Joylink2.0 指令转换及 lua 脚本示例代码

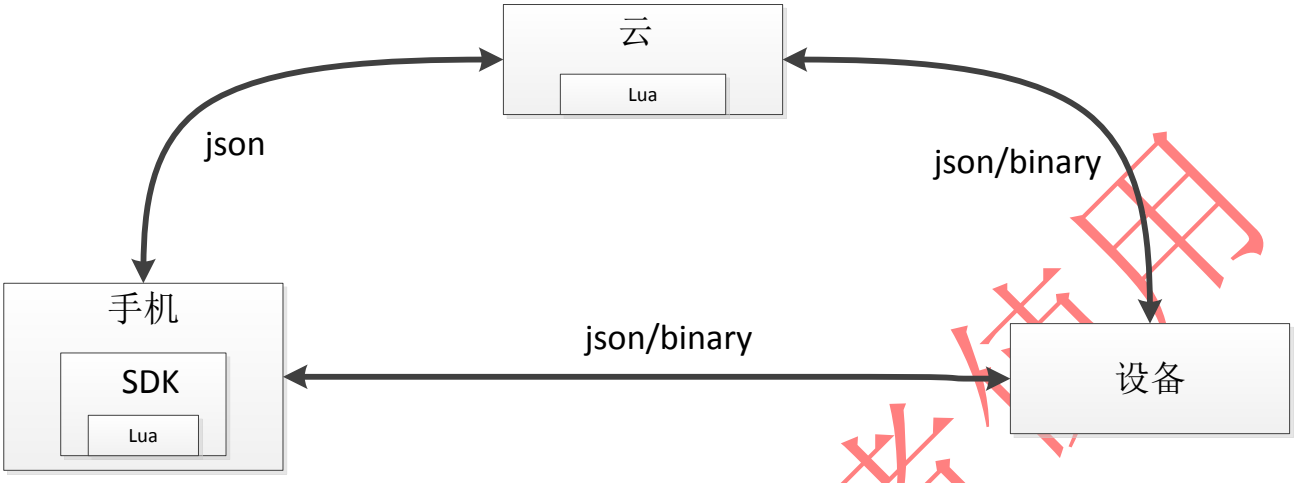
说明文档

京东智能协议组

目录

1. 系统架构

在说明指令转换和 lua 脚本之前，先介绍一下系统架构。



2. 京东标准指令

京东标准指令：在京东 Joylink 协议里所有的事物（包括设备 device，设备快照 snapshot，操作属性 stream_id，操作指令 cmd 等）均使用 json 语言描述。现在只举设备控制指令的例子，更多指令见协议文本：

设备控制请求

```
{
    “cmd”:5,
    “data”:{
        “streams”: [
            { “stream_id”: “switch”, “current_value”: “1” }
        ],
        "snapshot": [  //控制时下发全量快照
            {"stream_id":"switch", "current_value":"0"}
        ]
    }
}
```

```
{
    "streams": [{"current_value":"1","stream_id":"switch"}], //  控制命令
    "snapshot": [{"current_value":"0","stream_id":"switch"}] //  当前快照，包含全属性
}
```

设备控制响应

```
{
    "code": 0, // 响应结果
    "streams": [   // 快照
        {
            "current_value": "1",
            "stream_id": "switch"
        }
    ],
    "msg": "控制成功"
}
```

3. 厂家私有指令

厂家私有指令:指的是厂家设备 MCU 中的 firmware 能够理解和执行的控制指令，一般都由厂商自己定义。举例如下：

1、采用异步串口通信：波特率＝12600bps、1 个起始位、8 个数据位、1 个停止位、无奇偶校验位。

2、通信数据格式：每一帧数据包括 12 个字节，每个字节 8 位。
....(因为涉及厂商敏感信息，以下从略)

4. 指令转换 demo

指令转换：指的是**京东标准指令**转换为**厂家私有指令**；或者**厂家私有指令**转换为**京东标准指令**。

4.1. Demo1：京东标准指令转换为厂家私有指令

这是某开关的关闭指令：
京东标准指令：

```
{
    "cmd":5,
    "localkey":"c389eb817be7cd16a19c11592acf2748",
    "sublocalkey":"25e35c9691ee91296facd5d28a4822bc",
    "script":"abc.lua",
```

```
"device":{
    "version":1,
    "mac":"AC-CF-23-B8-78-5F",
    "ip":"192.168.99.242",
    "port":80,
    "productuuid":"111abc",
    "lancon":1,
    "trantype":1,
    "feedid":"147244371169429480",
    "devkey":"03fd4cea9ede125201528a98bc8fa4c81add0eee3b",
    "dbg":null,
    "opt":null,
    "devtype":0,
    "protocol":0,
    "parentmac":"",
    "state":0
},
"data":{
    "streams":[
        {
            "stream_id":"power",
            "current_value":"0"
        },
        {
            "stream_id":"Light1",
            "current_value":"100"
        }
    ],
    "snapshot":[
        {
            "stream_id":"power",
            "current_value":"1"
        },
        {
            "stream_id":"power2",
            "current_value":"0"
        },
        {
            "stream_id":"power3",
            "current_value":"0"
        },
        {
            "stream_id":"Light1",
            "current_value":"100"
```

```
            },
            {
                "stream_id":"Light2",
                "current_value":"100"
            },
            {
                "stream_id":"Light3",
                "current_value":"100"
            },
            {
                "stream_id":"switch1",
                "current_value":"3"
            },
            {
                "stream_id":"Lightable",
                "current_value":"0"
            },
            {
                "stream_id":"switchable",
                "current_value":"1"
            }
        ]
    }
}
```


转换得到的厂家私有指令：

00 ff ff 64 ff ff ff ff

4.2. Demo2：厂家私有指令转换为京东标准指令

开关关闭指令的设备响应：
厂家私有指令：
00 00 00 64 64 64 03 00 01
转化后得到的京东标准指令：


```
{
"msg":"done",
"streams":[
{
"stream_id":"power",
"current_value":0
},
{
"stream_id":"power2",
```

```
"current_value":0
},
{
"stream_id":"power3",
"current_value":0
},
{
"stream_id":"Light1",
"current_value":100
},
{
"stream_id":"Light2",
"current_value":100
},
{
"stream_id":"Light3",
"current_value":100
},
{
"stream_id":"switch1",
"current_value":3
},
{
"stream_id":"Lightable",
"current_value":0
},
{
"stream_id":"switchable",
"current_value":1
}
],
"code":0
}
```


5. Lua 脚本示例代码(注释说明)

实现指令转化是靠 Lua 脚本实现的。Lua 脚本的解析运行是使用开源项目 LuaC 5.3.3。添加了注释说明(见红色字体)。

```
function tableToString(cmd)        --函数，将表数据转为字符串
    local strcmd=""      --变量定义
    local I      --变量定义
    for i=1,#cmd do
        strcmd=strcmd..string.char(cmd[i])        --表中的每个成员转化为 char，添加到串后面
    end
```

```
    return strcmd     --返回串
end

function decode( cmd)
    local tb     --定义局部变量
    if cjson == nil then  --如果全局变量 cjson 是 nil
        cjson = (require 'JSON')     --加载外部对象 JSON 为全局变量 cjson 赋值
--为了方便代码管理，通常会把 lua 代码分成不同的模块，然后在通过 require 函数把它们加载进来
        tb = cjson:decode(cmd)     --使用外部对象的 decode 函数处理 cmd
    else
        tb = cjson.decode(cmd)     --使用自定义对象的 decode 函数处理 cmd
    end
    return tb        --返回 tb
end

function jds2pri( code, cmd )   --code:biz_code；cmd: 待转换的 json 串
    local bin;        --定义局部变量
    if code == 1002 then     --如果 code 是 1002
        bin = {        --局部表变量，赋初值
            0xFF,          --power
            0xFF,          --power2
            0xFF,          --power3
            0xFF,          --Light1
            0xFF,          --Light2
            0xFF,           --Light3
            0xFF,           --switch1
            0xFF,           --Lightable
        }
        local json = decode(cmd);   --定义局部变量，并赋值
        local streams = json["streams"];   --定义局部变量，并赋值
        for i=1, #streams do     --for 循环，从 1 到表 streams 的最大键值
            local key = streams[i]['stream_id']   --定义局部变量，值为 streams[i]的['stream_id']
            local val = streams[i]['current_value'] --定义局部变量，值为 streams[i]的[' current_value ']
            if (key == "power") then    --如果 key 是 power 的话
                if(val == 0x01 or val == "1") then   --如果 val 是 1 或者‘1’
                    bin[1] = 0x01;        --对 bin[1]赋值为 1
                elseif(val == 0x00 or val == "0") then   --如果 val 是 0 或者‘0’
                    bin[1] = 0x00;        --对 bin[1]赋值为 0
                end
            elseif (key == "power2") then    --同理，根据”power2”转换出 bin[2]
                if(val == 0x01 or val == "1") then
                    bin[2] = 0x01;
                elseif(val == 0x00 or val == "0") then
                    bin[2] = 0x00;
```

```
                end
            elseif (key == "power3") then          --同理，根据”power3”转换出 bin[3]
                if(val == 0x01 or val == "1") then
                    bin[3] = 0x01;
                elseif(val == 0x00 or val == "0") then
                    bin[3] = 0x00;
                end
            elseif (key == "Light1") then          --同理，根据” Light1”转换出 bin[4]
                bin[4] = val;
            elseif (key == "Light2") then          --同理，根据” Light2”转换出 bin[5]
                bin[5] = val;
            elseif (key == "Light3") then          --同理，根据” Light3”转换出 bin[6]
                bin[6] = val;
            elseif (key == "switch1") then         --同理，根据” switch1”转换出 bin[7]
                bin[7] = val;
            elseif (key == "Lightable") then       --同理，根据” Lightable”转换出 bin[8]
                if(val == 0x01 or val == "1") then
                    bin[8] = 0x01;
                elseif(val == 0x00 or val == "0") then
                    bin[8] = 0x00;
                end
            end
        end

    end

    local ret = tableToString(bin)          --Lua 表数据转化为普通字符串
    return 0, string.len(ret), ret          --返回成功码，转化得到的字符串长度；字符串
end

function pri2jds( code, length, bin )       --code:biz_code; length:脚本长度；bin:待转换的脚本
    local retTable = {};    --定义局部变量
    -- cloudmenu
    if code == 150 then     --云菜谱响应
        --no payload
    else
        -- control 102 ; report 103 ; snapshot 104
        local streams = {};     --定义局部变量
        streams[1] = {};        --赋初值
        streams[1]["stream_id"] = "power";   --赋初值
        if(bin:byte(1) == 0x01) then        --根据 bin[1]给 streams[1]["current_value"]赋值
            streams[1]["current_value"] = 0x01;
        elseif(bin:byte(1) == 0x00)     then
            streams[1]["current_value"] = 0x00;
```

```
end
streams[2] = {};        --根据 bin[2]给 streams[2]赋值
streams[2]["stream_id"] = "power2";
if(bin:byte(2) == 0x01) then
    streams[2]["current_value"] = 0x01;
elseif(bin:byte(2) == 0x00)    then
    streams[2]["current_value"] = 0x00;
end
streams[3] = {};       --根据 bin[3]给 streams[3]赋值
streams[3]["stream_id"] = "power3";
if(bin:byte(3) == 0x01) then
    streams[3]["current_value"] = 0x01;
elseif(bin:byte(3) == 0x00)    then
    streams[3]["current_value"] = 0x00;
end
streams[4] = {};       --根据 bin[4]给 streams[4]赋值
streams[4]["stream_id"] = "Light1";
streams[4]["current_value"] = bin:byte(4);

streams[5] = {};          --根据 bin[5]给 streams[5]赋值
streams[5]["stream_id"] = "Light2";
streams[5]["current_value"] = bin:byte(5);

streams[6] = {};          --根据 bin[6]给 streams[6]赋值
streams[6]["stream_id"] = "Light3";
streams[6]["current_value"] = bin:byte(6);

streams[7] = {};        --根据 bin[7]给 streams[7]赋值
streams[7]["stream_id"] = "switch1";
streams[7]["current_value"] = bin:byte(7);

streams[8] = {};     --根据 bin[8]给 streams[8]赋值
streams[8]["stream_id"] = "Lightable";
if(bin:byte(8) == 0x01) then
    streams[8]["current_value"] = 0x01;
elseif(bin:byte(8) == 0x00) then
    streams[8]["current_value"] = 0x00;
end

streams[9] = {};          --根据 bin[9]给 streams[9]赋值
streams[9]["stream_id"] = "switchable";
if(bin:byte(9) == 0x01) then
    streams[9]["current_value"] = 0x01;
elseif(bin:byte(9) == 0x00) then
```

```
                streams[9]["current_value"] = 0x00;
            end

            retTable["streams"] = streams;      --赋值
        end
        retTable["code"] = 0;      --赋值
        retTable["msg"] = "done";      --赋值
        local jsonStr = cjson.encode(retTable);          --将表数据转化为标准 json
        return 0, jsonStr, code;       --返回成功码，转化得到的 json 串，biz_code
end
```