API 说明文档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **时间** | **起草** | **修订** | **版本** | **说明** |
| 2016/10/9 | Mingao | Lewis | V0.0.1 | 初步完成用户相关操作 |
| 2016/10/12 | Lewis |  | V0.0.2 | 加入Websocket机制 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**1． 总述**

为了让用户实时感知设备的状态，将对设备管理模式进行调整。由初始HTTP请求的方式，更换为Websocket通信方式。

客户端请求连接地址格式为：ws://IP:PORT/ws; 基中IP指连接家庭网关的地址， PORT指家庭网关开放端口，默认为8360。

为了操作方便， 当用户登录成功后，客户端可以实时通过HTTP请求Websocket的地址。当用户获得到家庭网关的Websocket地址后，可以正常请求Websocket连接。 建立连接后，可以正常通信。

**1.1 请求地址**

请求方式：POST

请求命令： /ws?uid=xxxxxx

a)：成功登录，回复数据

{

“error” : 0

“action” : REPLY

“data”: {

“uri” = ws://IP:PORT/ws

}

}

实例：

{

"error": 0,

" action ": "reply",

"data": {

“uri”= ws://192.168.10.99:8360/ws

｝

}

b)： 登录失败，回复数据

{

“error”:errno

“action”: REPLY

}

**1.2 建立连接**

客户端websocket 连接，获得句柄，建立通信。

浏览器利用websocket建立连接的示例：

**<script** type="text/javascript"**>**

**var** sock = null;

**var** wsuri = "ws://127.0.0.1:8360/ws";

    $(document).ready(**function**(){

        $.post("/ws",{

            uid: 1474274034,

        },

**function**(data,status){

            wsuri = data.data;

        });

});

window.onload = **function**() {

console.log("onload");

sock = **new** WebSocket(wsuri);

sock.onopen = **function**() {

console.log("connected to " + wsuri);

}

sock.onclose = **function**(e) {

console.log("connection closed (" + e.code + ")");

}

sock.onmessage = **function**(e) {

console.log("message received: " + e.data);

          alert(e.data)

}

};

**function** send() {

**var** msg = document.getElementById('message').value;

sock.send(msg);

}

**</script>**

**2. 数据格式**

**a) 请求数据格式：**

**type RequestInfo struct{**

**Event string `json:"event"`**

**Action** **string `json:"action"`**

**Data interface{} `json:"data"`**

**｝**

**其中action表示动作为：**

const (

REQUEST = “request”

REPLY = “reply”

NOTIFY = “notify”

)

（注： 通常NOTIFY表示由服务端主动发起通知，而客户端无需对此类消息进行响应； REQUEST表示数据请求，REPLY表示数据响应）

**其中Event 表示事件：**

const (

CONNECT = “connect”

DISCONNECT = “disconnect”

ADDNODE = “addnode”

REMOVENODE = “removenode”

GETGATEWAYLIST = “getgatewaylist”

GETNODELIST = “getnodelist”

GETNODEADDRI=”getnodeaddri”

GETNDOEDATA = “getnodedata”

CONTRLNODE = “contrlnode”

…

)

**b) 回复数据结构为：**

**type** **MessageInfo struct {**

**ErrorNumber int `json:"error"`**

**Action string `json:"action"`**

**Data** **interface{} `json:"data"`**

**}**

**3. 消息通知**

通知消息格式为：

{

"error": 0,

" action ": NOTIFY,

“data”: {

“notify”: NOTIFY\_INFO

}

}

其中， NOTIFY\_INFO定义如下：

const

{

NOTIFY\_WELCOME = 0x00 //“welcome information”

NOTIFY\_USER\_OFFLIEN = 0x10 //“user offline”

NOTIFY\_GATEWAY\_CHANGED = 0x20 //“gateway changed”

NOTIFY\_NODE\_CHANGED = 0x30 //“node changed”

}

**3.1** **欢迎消息**

{

"error": 0,

" action ": NOTIFY,

“data”: {

“notify”: NOTIFY\_WELCOME

}

}

**3.2** **通知用户下线**

{

"error": 0,

" action ": NOTIFY,

“data”: {

“notify”: NOTIFY\_USER\_OFFLIEN

}

}

**3.3 通知通信链路断开**

{

"error": 0,

" action ": NOTIFY,

“data”: {

“notify”: NOTIFY\_USER\_OFFLIEN

}

}

**3.3 通知网关状态改变**

{

"error": 0,

" action ": NOTIFY,

“data”: {

“notify”: NOTIFY\_NODE\_CHANGED

}

}

**4. 设备管理**

**4.1 获得网关信息**

**1) 请求命令格式：**

{

"event"：GATEWAYLIST

"action"：REQUEST

}

**2) 响应数据格式**

{

"error": 0,

" action ": REPLY,

“data”: [

….

]

}

**实例：**请求设备信息和响应数据

{

"action"："request"

"event"："gatewaylist"

｝

{

"error": 0,

"action ": "reply",

"data": [

{

"id": 0,

"category": "gateway",

"gatewayName": "gateway1",

"address": "192.168.10.99",

"protocol": "SIMULATE",

},

{

"id": 1,

"category": "gateway",

"gatewayName": "gateway2",

"address": "192.168.10.227",

"protocol": "SIMULATE",

},

{

"id": 2,

"category": "gateway",

"gatewayName": "gateway3",

"address": "192.168.10.1",

"protocol": "SIMULATE",

}

]

}

｝

**4.2 请求结点信息**

**1) 请求命令格式：**

{

"action"："request"

"event":"getnodelist"

“data”:

{

“gid”: 0

}

}

（注： gid 为对应网关ID）

**2) 响应数据格式**

**a) 正常响应**

{

"error": 0,

"action": "reply",

"data": [

{

"id": 0,

"category": "switch",

"nodename": "switch1",

"address": "fe80::19c0:d694:85c1:b81",

"protocol": "SIMULATE"

},

{

"id": 1,

"category": "switch",

"nodename": "switch2",

"address": "fe80::19c0:d694:85c1:b81",

"protocol": "SIMULATE"

},

{

"id": 2,

"category": "test",

"nodename": "test3",

"address": "fe80::19c0:d694:85c1:b81",

"protocol": "SIMULATE"

}

]

}

**b)错误响应**

{

"error": 1,

"action": "reply",

“data”: “xxxxxxxxxx”

}

**4.3 获得节点详细信息 (属性等)**

**1) 请求命令格式：**

{

"action"："request"

"event"：" getnodeaddri"

“data”:

{

“gid”: 0

“nid”:0

}

}

（注： gid 为对应网关ID， 而uid为指定网关下面对应的节点id）

**2) 响应数据格式**

{

"error": 0,

"action": "reply",

"data": [

{

"id": 0,

"category": "switch",

"nodename": "switch1",

"address": "fe80::19c0:d694:85c1:b81",

"protocol": "SIMULATE"

}

]

}

**4.4 请求控制节点**

对于控制节点，需要传入相关的控制命令。

**1) 请求命令格式：**

{

"action"："request"

"event"："controlnode"

“data”:

{

“gid”: 0

“nid”: 0

“cmd”:xxxxx

paras:{

“para1”:xxxx

“para2”:xxxxx

….

}

}

}

**2) 响应数据格式(待完善)**

4.5 获得节点数据

command的数据格式设计：

cmd:COMAND

para:{

para1:PARA1

para2:PARA2

}

个人觉得，命令应该与底层ZWAVE相对应，以兼容ZWAVE的控制命令。

其中，cmd的定义列表如下：

const

{

GETCURDATA = “getcurdata”

GETHISTORYDATA = “gethistorydata”

….

}

**1) 请求命令格式：**

{

"action"："request"

"event"："getnodedata"

“data”:

{

“gid”: 0

“nid” : 0

“cmd”:xxxxx

paras:{

“para1”:xxxx

“para2”:xxxxx

….

}

}

}

**2) 响应数据格式(待完善)**