

## 盈客通维码器 socket 协议文档 v1.3

### 测试服务：

地址 (Ip: 端口) 类型: TCP:

47.94.208.29:7701 (协议 1.1 该服务不支持)

备注: 最好自行搭建服务 (较容易调试),

后台发送指令接口 (供测试客户端接收指令)

客户端相关: url: <http://47.94.208.29:5543/api/client/>

查看所有客户端: 方式: get

参数: 无

授权客户端: 方式: Put

参数 (x-www-form-urlencoded) : id(客户端列表中客户端的 ID)

取消授权客户端: 方式: Delete

参数 (x-www-form-urlencoded) : id(客户端列表中客户端的 ID)

出币: url: <http://47.94.208.29:5543/api/coin> 方式: Post

参数 (x-www-form-urlencoded) : CoinCount (出币数量)、clientid (客户端 ID, 识别号)、chance (概率)、voltage (电压)

设置电压: url: <http://47.94.208.29:5543/api/voltage> 方式: Put

参数 (x-www-form-urlencoded) : clientid (客户端 ID, 识别号)、voltage (电压)

设置概率: url: <http://47.94.208.29:5543/api/chance> 方式: Put

参数 (x-www-form-urlencoded) : clientid (客户端 ID, 识别号)、chance (概率)

### 数据头和尾

名称	数据	数据类型
头	+cmd+	string
尾	-end-	string

### 指令编码:

心跳	1000
接受出币	2000
出币完成	2001
执行失败	4000

设置默认概率	9000
设置默认概率成功	9001
上报信息	7000
上报故障	7001
设置电压	9002
设置电压完成	9003

指令直接转为 byte[]

指令及多包粘连实例（代码为 C#）

代码

```
static void Main(string[] args)
{
    #region 指令转byte[]

    ///数据头
    string header = "+-cmd-+",
        ///数据尾
        footer = "-end+-";
    ///将数据头和尾转为byte[]
    var bHeader = Encoding.UTF8.GetBytes(header);
    var bFooter = Encoding.UTF8.GetBytes(footer);

    ///指令头
    int m1 = 1000, ///心跳
        m2 = 2000, ///接受出币
        m3 = 9000;///设置默认概率

    ///将指令头转为byte[]
    var m1bs = BitConverter.GetBytes(m1);
    var m2bs = BitConverter.GetBytes(m2);
    var m3bs = BitConverter.GetBytes(m3);

    ///接受出币数据
    var v2 =
        "{ \"MessageID\": \"5af8ffa42157bc10d8ec90e3\", \"ClientID\": \"99999\", \"CoinCount\": 1000, \"Chance\": 100 }";
    ///设置默认概率数据
    var v3 =
        "{ \"MessageID\": \"5af8ffa92157bc10d8ec90f3\", \"ClientID\": \"99999\", \"Chance\": 100 }";

    Console.WriteLine($"测试数据:
```

```
{header}{m1}{footer}{header}{m2}{v2}{footer}{header}{m3}{v3}{footer}");
```

```
///将数据转为byte[]
```

```
var b2v = Encoding.UTF8.GetBytes(v2);
```

```
var b3v = Encoding.UTF8.GetBytes(v3);
```

```
///将数据进行打包
```

```
var b1h = GetHeaderBytes(m1bs, bHeader, bFooter);
```

```
var b2h = GetHeaderBytes(m2bs, bHeader, bFooter, b2v);
```

```
var b3h = GetHeaderBytes(m3bs, bHeader, bFooter, b3v);
```

```
///模拟粘连后的总数据
```

```
var bSum = new byte[b1h.Length + b2h.Length + b3h.Length];
```

```
b1h.CopyTo(bSum, 0);
```

```
b2h.CopyTo(bSum, b1h.Length);
```

```
b3h.CopyTo(bSum, b2h.Length + b1h.Length);
```

```
Console.WriteLine("\nbyte[]数组输出:");
```

```
for (int i = 0; i < bSum.Length; i++)
```

```
{
```

```
    Console.WriteLine($"{i}:{bSum[i]}");
```

```
}
```

```
#endregion
```

```
#region byte[]转指令
```

```
Console.WriteLine("\n所有数据直接转UTF8字符串: " +  
Encoding.UTF8.GetString(bSum));
```

```
///数据包头部下标集合
```

```
var bheaderLastIndexs = new List<int>();
```

```
///数据包尾部下标集合
```

```
var bFooterFirstIndexs = new List<int>();
```

```
///找到所有数据包的头部和尾部
```

```
for (int i = 0; i < bSum.Length; i++)
```

```
{
```

```
    if (HasData(bHeader, bSum, i))
```

```
    {
```

```
        i = i + bHeader.Length - 1;
```

```
        bheaderLastIndexs.Add(i + 1);
```

```
    }
```

```
    if (HasData(bFooter, bSum, i))
```

```
    {
```

```
        bFooterFirstIndexs.Add(i);
```

```
        i = i + bFooter.Length - 1;
```

```
    }
```

```
}
```

```
///将数据包去除头部和尾部的数据包列表
```

```
var dataList = new List<byte[]>();
```

```
if (bheaderLastIndexs.Count == bFooterFirstIndexs.Count)
```

```
{
```

```
    for (int i = 0; i < bheaderLastIndexs.Count; i++)
```

```
    {
```

```
dataList.Add(bSum.Skip(bheaderLastIndexs[i]).Take(bFooterFirstIndexs[i] -
```

```

bheaderLastIndexs[i]).ToArray());
    }
}
else///如果头部数量不等于尾部数量则为错误数据, 不进行处理
    return;
///获取真是数据
Console.WriteLine("\n从byte[]转为真实数据: ");
foreach (var item in dataList)
{
    var ht = GetHeaderValue(new byte[][] { m1bs, m2bs, m3bs }, item);
    var v = "";
    if (ht.Length < item.Length)
    {
        v = Encoding.UTF8.GetString(item, ht.Length, item.Length -
ht.Length);
    }
    Console.WriteLine($"{BitConverter.ToInt32(ht, 0)}:{v}");
}
#endregion

Console.ReadLine();
}
/// <summary>
/// 从数据包中获取指令
/// </summary>
/// <param name="v">所有指令数组</param>
/// <param name="data">当前数据包</param>
/// <returns></returns>
private static byte[] GetHeaderValue(byte[][] v, byte[] data)
{
    byte[] b = null;
    foreach (var item in v)
    {
        var flag = true;
        for (int i = 0; i < item.Length; i++)
        {
            if (item[i] != data[i])
            {
                flag = false;
                break;
            }
        }
        if (flag)
        {
            b = item;
            break;
        }
    }
    return b;
}
/// <summary>
/// 判断是否为数据头或者尾部
/// </summary>
/// <param name="bHeader"></param>
/// <param name="bSum"></param>

```

```

    /// <param name="index"></param>
    /// <returns></returns>
    private static bool HasData(byte[] bHeader, byte[] bSum, int index)
    {
        var flag = true;
        for (int i = 0; i < bHeader.Length; i++)
        {
            if (bHeader[i] != bSum[index + i])
            {
                flag = false;
                break;
            }
        }
        return flag;
    }
    /// <summary>
    /// 对数据进行打包
    /// </summary>
    /// <param name="b1hs">指令代码byte[]</param>
    /// <param name="bHeader">数据头</param>
    /// <param name="bFooter">数据尾</param>
    /// <param name="b2v">数据</param>
    /// <returns></returns>
    private static byte[] GetHeaderBytes(byte[] b1hs, byte[] bHeader, byte[]
bFooter, byte[] b2v = null)
    {
        var b1h = b2v == null ? new byte[bHeader.Length + b1hs.Length +
bFooter.Length] : new byte[bHeader.Length + b1hs.Length + b2v.Length +
bFooter.Length];
        bHeader.CopyTo(b1h, 0);
        b1hs.CopyTo(b1h, bHeader.Length);
        if (b2v != null)
        {
            b2v.CopyTo(b1h, b1hs.Length + bHeader.Length);
        }
        bFooter.CopyTo(b1h, b1h.Length - bFooter.Length);
        return b1h;
    }
}

```

输出内容:

测试数据: +-cmd-+1000-+end+-+cmd-  
+2000{"MessageID":"5af8ffa42157bc10d8ec90e3","ClientID":"99999","CoinCount":1000,"  
Chance":100}-+end+-+cmd-  
+9000{"MessageID":"5af8ffa92157bc10d8ec90f3","ClientID":"99999","Chance":100}-  
+end+-

byte[]数组输出:

0:43  
1:45  
2:99  
3:109  
4:100  
5:45

6:43  
7:232  
8:3  
9:0  
10:0  
11:45  
12:43  
13:101  
14:110  
15:100  
16:43  
17:45  
18:43  
19:45  
20:99  
21:109  
22:100  
23:45  
24:43  
25:208  
26:7  
27:0  
28:0  
29:123  
30:34  
31:77  
32:101  
33:115  
34:115  
35:97  
36:103  
37:101  
38:73  
39:68  
40:34  
41:58  
42:34  
43:53  
44:97  
45:102  
46:56  
47:102  
48:102  
49:97  
50:52  
51:50  
52:49  
53:53  
54:55  
55:98  
56:99  
57:49  
58:48  
59:100  
60:56  
61:101  
62:99  
63:57

Create By Alair Eileen

64:48  
65:101  
66:51  
67:34  
68:44  
69:34  
70:67  
71:108  
72:105  
73:101  
74:110  
75:116  
76:73  
77:68  
78:34  
79:58  
80:34  
81:57  
82:57  
83:57  
84:57  
85:57  
86:34  
87:44  
88:34  
89:67  
90:111  
91:105  
92:110  
93:67  
94:111  
95:117  
96:110  
97:116  
98:34  
99:58  
100:49  
101:48  
102:48  
103:48  
104:44  
105:34  
106:67  
107:104  
108:97  
109:110  
110:99  
111:101  
112:34  
113:58  
114:49  
115:48  
116:48  
117:125  
118:45  
119:43  
120:101  
121:110

Create By Alair Eileen

122:100  
123:43  
124:45  
125:43  
126:45  
127:99  
128:109  
129:100  
130:45  
131:43  
132:40  
133:35  
134:0  
135:0  
136:123  
137:34  
138:77  
139:101  
140:115  
141:115  
142:97  
143:103  
144:101  
145:73  
146:68  
147:34  
148:58  
149:34  
150:53  
151:97  
152:102  
153:56  
154:102  
155:102  
156:97  
157:57  
158:50  
159:49  
160:53  
161:55  
162:98  
163:99  
164:49  
165:48  
166:100  
167:56  
168:101  
169:99  
170:57  
171:48  
172:102  
173:51  
174:34  
175:44  
176:34  
177:67  
178:108  
179:105



180:101  
181:110  
182:116  
183:73  
184:68  
185:34  
186:58  
187:34  
188:57  
189:57  
190:57  
191:57  
192:57  
193:34  
194:44  
195:34  
196:67  
197:104  
198:97  
199:110  
200:99  
201:101  
202:34  
203:58  
204:49  
205:48  
206:48  
207:125  
208:45  
209:43  
210:101  
211:110  
212:100  
213:43  
214:45

所有数据直接转UTF8字符串: +-cmd-+?\_\_\_\_\_ -+end+--+cmd-+?  
{ "MessageID": "5af8ffa42157bc10d8ec90e3", "ClientID": "99999", "CoinCount": 1000, "Chance": 100 } -+end+--+cmd-+ (#  
{ "MessageID": "5af8ffa92157bc10d8ec90f3", "ClientID": "99999", "Chance": 100 } -+end+-

从byte[]转为真实数据:

1000:  
2000: { "MessageID": "5af8ffa42157bc10d8ec90e3", "ClientID": "99999", "CoinCount": 1000, "Chance": 100 }  
9000: { "MessageID": "5af8ffa92157bc10d8ec90f3", "ClientID": "99999", "Chance": 100 }

## 特定数据规定

### 数据类型最大长度

数据类型	长度
String	100
json	

## 消息关键词解释

数据名称	解释	数据类型	长度		
ClientID	客户端 ID	String			
MessageID	消息 ID	String	24(max)		
CoinCount	出币数量	Int	1-99		
Chance	概率	Int	1-99		
Voltage	电压	Int	1-99		

### 概率解释 (Chance)

取值范围为 0 到 100

0 为永不中奖, 100 为每次都中奖

#### 1. 心跳

过程解释:

当完成后, 马上开启心跳线程 (注意: 服务端也会向客户端发起心跳, 客户端必须处理服务端的心跳) 心跳速度为 10s/次

指令发送:

传递[心跳]指令和设备 ID 数据为:

```
{  
  "ClientID": "3234"  
}
```

将数据转为 byte[] 后追加到指令后面

指令接收:

接收服务端[心跳]指令(服务器只发送指令)

#### 2. 接收出币

过程解释:

服务器会随时的向客户端[接受出币]指令, 客户端需要按照该设备做出出币的动作

注意: 这里的概率是指当前投币后生效的概率, 当前投币结束后再回到默认的概率

指令接收:

[接受出币]+携带数据

从 byte[] 转 string 为:

```
{
  "MessageID": "jfooejewfjiojfioeie",
  "ClientID": "3234",
  "CoinCount": 32,

  "Voltage": 32,
  "Chance": 80
}
```

注意：接收到该数据后要核实该 ClientID 是否与本客户端 ID 一致，如果不一致则发送失败信息到服务端

CoinCount、Voltage、Chance 最值都为 1-99

### 3. 出币完成

过程解释：

当出币完成后，马上发送[出币完成]指令到服务端

指令发送：

传递[出币完成]指令+携带数据

```
{
  "MessageID": "jfooejewfjiojfioeie",
  "ClientID": "3234"
}
```

将数据转为 byte[]后追加到指令后面)

### 4. 设置默认概率

过程解释：

服务器向客户端发送[设置默认概率]指令，客户端需要按照该设备设置默认概率

指令接收：

[设置默认概率]指令+携带数据

从 byte[]转 string 为：

```
{
  "MessageID": "jfooejewfjiojfioeie",
  "ClientID": "3234",
  "Chance": 100
}
```

注意：接收到该数据后要核实该 ClientID 是否与本客户端 ID 一致，如果不一致则发送失败信息到服务端

Chance 最值为 1-99

## 5. 设置默认概率成功

过程解释:

当概率设置完成后, 马上发送[设置默认概率成功]指令到服务端

指令发送:

传递[设置默认概率成功]指令+携带数据

```
{  
  "MessageID": "jfooejewfjiojfioeie",  
  "ClientID": "3234"  
}
```

将数据转为 byte[]后追加到指令后面)

## 6. 设置默认电压

过程解释:

服务器向客户端发送[设置默认电压]指令, 客户端需要按照该设备设置默认电压

指令接收:

[设置默认电压]指令+携带数据

从 byte[]转 string 为:

```
{  
  "MessageID": "jfooejewfjiojfioeie",  
  "ClientID": "3234",  
  "Voltage": 10  
}
```

注意: 接收到该数据后要核实该 ClientID 是否与本客户端 ID 一致, 如果不一致则发送失败信息到服务端

Voltage 最值为 1-99

## 7. 设置默认电压成功

过程解释:

当电压设置完成后, 马上发送[设置默认电压成功]指令到服务端

指令发送:

传递[设置默认电压成功]指令+携带数据

```
{  
  "MessageID": "jfooejewfjiojfioeie",
```

```
    "ClientID": "3234"  
  }
```

将数据转为 byte[] 后追加到指令后面)

## 8. 执行失败

过程解释:

当执行失败后, 马上发送[执行失败]指令到服务端

指令发送:

传递[执行失败]指令+携带数据

```
{  
  "MessageID": "jfooejewfjiojfoeiie",  
  "ClientID": "3234"  
}
```

将数据转为 byte[] 后追加到指令后面)

## 9. 上报信息

过程解释:

当客户端有特殊日志信息后, 马上发送[上报信息]指令到服务端

指令发送:

传递[上报信息]指令+携带数据

```
{  
  "ClientID": "3234",  
  "LogInfo": {  
    "Content": "日志详情内容",  
  }  
}
```

将数据转为 byte[] 后追加到指令后面)

## 10. 上报故障

过程解释：

当客户端有特殊故障信息后，马上发送[上报故障]指令到服务端

指令发送：

传递[上报故障]指令+携带数据

```
{  
  "ClientID": "3234",  
  "BugInfo": {  
    "Content": "日志详情内容",  
  }  
}
```

将数据转为 byte[]后追加到指令后面)

## 11. 断开自动连接

过程解释：

当连接中断时，尝试再次连接，3 次后 3 分钟后再执行该任务