常见踩坑

串口使用问题

- 环境安装
    1. 前置条件: 安装python2.7(一定要安装2.7版本)和VS2017
    2. 安装windows-build-tools

```
npm install --global --production windows-build-tools
// 可能会遇到安装python安装时挂起问题
```

    3. 安装node-gyp

```
npm install -g node-gyp
```

    4. 配置python和vscode

```
npm config set python python2.7
npm config set msvs_version 2019
npm config set python C:\Python27 (指定python2的路径)
```

    5. 测试node-gyp是否安装完成

```
node-gyp
// 成功显示则回复
Usage:node-gyp [options]  where is one of:    -build - Invokes `msbuild`and
builds the module    - clean - Removes any generatedbuild filesand
the"out"dir-
configure - Generates MSVC project files for the current module    - rebuild
-
Runs"clean","configure"and "build"allatonce    -install -Install node
development files for the specified node version.    - list - Prints a
listing
of the currentlyinstalled node development files    - remove - Removes the
node
development files for the specified versionnode-gyp@6.1.
0H:\npm\node_global\node_modules\node-gypnode@8.16.2
```

    6. 安装serialport和electron-rebuild

```
npm install serialport
npm install --save-dev electron-rebuild // 我是用的2.3.5版本
```

7. 使用electorn-rebuild编译

```
.\node_modules\.bin\electron-rebuild.cmd
// 提示Rebuild Complete则编译成功
```

UDP使用问题

- 环境安装
  1. 无需安装, 直接使用nodejs的dgram模块
  2. 文档链接 http://nodejs.cn/api/dgram.html
- 使用
  1. 创建UDP对象

```
const server = dgram.createSocket('udp4')
```

2. 绑定监听端口

```
// SEND_PORT 为要监听的端口, ipAddress为网卡的ip地址, setBroadcast只能在绑定套接字的时候使用
server.bind(SEND_PORT, ipAddress, () => {
  server.setBroadcast(true)
})
```

3. 利用message事件接收客户端返回的数据

```
server.on('message', (msg, rinfo) => {
  // 做回调操作
})
```

4. 发送广播数据

```
// SEND_BROADCAST_CONTENT为广播内容, SEND_PORT为发送端口, BROA_ADDRESS为广播地址
server.send(SEND_BROADCAST_CONTENT, SEND_PORT, BROA_ADDRESS)
```

TCP的使用问题

- 环境安装

    1. 无需安装, 直接使用node.js的net模块的的Socket

- 使用

    1. 引入net模块并实例化socket对象,创建连接

```
import { Socket } from 'net'

const socket = new Socket()

socket.connect({ port: 8000, host: '192.168.1.100' }, (err) => {
  if (err) {
    // 连接失败
  } else {
    // 连接成功
  }
}) // 创建连接
```

    2. 创建数据监听

```
import { StringDecoder } from 'string_decoder'

const decoder = new StringDecoder('utf8')

socket.on('data', (data) => {
  // data为buffer数据, 使用nodejs的string_decoder模块将buffer数据转成json对象
  const decoderData = decoder(data) // 解析后的json数据
})
```

    3. 发送数据

```
const sendData = { a: 0, b: '', c: [] }
socket.write(JSON.stringify(sendData))
```

- 踩到的坑(重点看!!!!!!踩了好久的坑)

    1. 在渲染进程中,使用TCP接收不到数据的问题,

```
// AddGroupWindow.vue
import { Socket } from 'net'

export default {
```

```
  data () {
    // ...
  },
  methods () {
    createTcpConnect () {
      const socket = new Socket()

      socket.connet({ port: 8000, host: '192.168.1.100' }, (e) => {
        if (e) {
          throw new Error(e)
        } else {
          console.log('连接成功')
        }
      })
      socket.on('data', (data) => {
        console.log(data)
      })

      const sendData = {
        guiheda: 'addGroupWindow'
        groupId: 0
        // ...
      }
      socket.write(JSON.stringify(sendData) + '\r')
    }
  }
}
```

此时调用CreateTcpConnect时, 通过抓包工具我们可以看到有发送数据, 并且有数据回复,但是我们在socket.on('data')的回调函数里监听函数并不会触发, 这让我百思不得其解, 最后经过尝试各种方法得到以下结论: 1.Socket模块在渲染进程中会接收不到数据, 在主进程中正常接收数据

解决方案: 在渲染进程中使用remote模块来使用主进程的模块,从而解决在渲染进程中接收不到数据的问题

```
import { ipcRenderer, remote } from 'electron'
import { Socket } from 'net'

const { ipcMain } = remote
const listenerKey = 'addGroupWindow'
export default {
  data () {
    // ...
  },
  created () {
    this.createTcpConnect()
  },
  methods () {
    createTcpConnect () {
      ipcMain.on(listenerKey, (e, data) => {
```

```
          const socket = new Socket()
          socket.connet({ port: 8000, host: '192.168.1.100' }, (e) => {
            if (e) {
              throw new Error(e)
            } else {
              console.log('连接成功')
            }
          })
          socket.on('data', (data) => {
            console.log(data)
            // 校验好数据后将socket连接关闭
            socket.destroy()
          })

          const sendData = {
            guiheda: 'addGroupWindow'
            groupId: 0
            // ...
          }
          socket.write(JSON.stringify(sendData) + '\r')
        })
      }
    },
    destroyed () {
      ipcMain.removeAllListeners(listenerKey) // 注意 一定要记得移除 否则会重复触
发
    }
  }
```

注意: 一定要移除主进程的事件监听, 和TCP的事件监听, 否则会成幂的触发你的事件监听函数!!!!