

目录

前言	1.1
Electron简介	1.2
应用举例	1.2.1
安装	1.3
基本使用	1.4
打包和部署	1.5
python支持	1.6
心得	1.7
开发	1.7.1
Web技术	1.7.1.1
Log日志	1.7.1.2
打包	1.7.2
制作app的Logo	1.7.2.1
asar	1.7.2.2
files	1.7.2.3
electron-builder	1.7.2.4
electron-rebuild	1.7.2.5
不同系统	1.7.3
Win	1.7.3.1
Mac	1.7.3.2
不同语言	1.7.4
Python	1.7.4.1
electron-python-example	1.7.4.1.1
js	1.7.4.2
Node	1.7.5
electron和node版本对应关系	1.7.5.1
NODE_MODULE_VERSION和node版本对应关系	1.7.5.2
附录	1.8
文档和教程	1.8.1
参考资料	1.8.2

跨平台桌面应用框架：Electron

- 最新版本: v1.0
- 更新时间: 20200618

鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 crifan 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

简介

介绍跨平台桌面应用框架Electron的基本概念、基本原理、核心优势以及额外特性；介绍了用Electron开发出的常见应用有哪些；解释了如何安装Electron；以及如何快速上手使用；以及如何打包和部署；专门整理了如何让Electron支持Python以及其后的相关心得；总结了Electron的各种开发经验和心得，比如开发方面的Web技术、Log日志等；打包方面的如何制作app的logo、asar加密压缩、files参数、打包工具electron-builder、编译工具electron-rebuild等，以及常见的系统如Windows和Mac系统中常见问题和解决方案；以及不同语言，比如Python、js等的支持，整理了Node相关信息，最后附上相关文档和教程等资料。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

Gitbook源码

- [crifan/desktop_app_framework_electron: 跨平台桌面应用框架：Electron](#)

如何使用此Gitbook源码去生成发布为电子书

详见：[crifan/gitbook_template: demo how to use crifan gitbook template and demo](#)

在线浏览

- [跨平台桌面应用框架：Electron book.crifan.com](#)
- [跨平台桌面应用框架：Electron crifan.github.io](#)

离线下载阅读

- 跨平台桌面应用框架: [Electron PDF](#)
- 跨平台桌面应用框架: [Electron ePub](#)
- 跨平台桌面应用框架: [Electron Mobi](#)

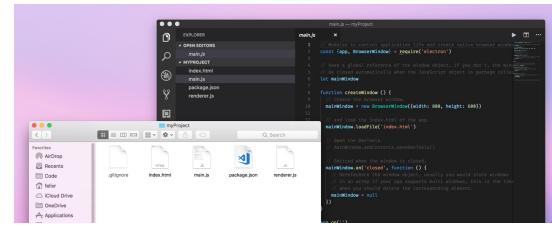
版权说明

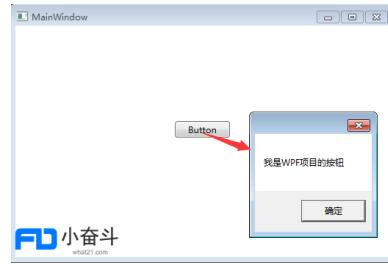
此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您版权，请通过邮箱联系我 `admin` 艾特 `crifan.com`，我会尽快删除。谢谢合作。

[crifan.com](#), 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-06-19 09:27:06

Electron简介

- Electron概述
 - 所属技术领域:
 - 跨平台的桌面端应用开发
 - 谁开发的: Github
 - 旧称: Atom Shell
 - 历史
 - 2013年作为构建Github上可编程的文本编辑器Atom的框架而被开发出来, 这两个项目在2014春季开源
 - 一句话描述: 一个用纯Web技术来构建跨平台桌面应用程序的开源框架
 - Web技术: HTML、CSS 和 JavaScript
 - 对比: 传统桌面应用都是非Web技术开发的
 - 跨平台: Win / Mac / Linux 等多个平台
 - 桌面应用: 主要用来开发桌面端应用
 - 而不是Web应用
 - 基本原理
 - 将Chromium和Node.js合并到同一个运行时环境中
 - 让你使用纯 JavaScript 调用丰富的原生(操作系统) APIs
 - 并将其打包为 Mac、Windows 和 Linux 系统下的应用
 - 可看成
 - 一个 Node.js 的变体
 - 专注于桌面应用而不是 Web Server 端
 - 使用 web 页面作为它的 GUI
 - 一个被 JavaScript 控制的, 精简版的 Chromium 浏览器
 - 主页
 - [Electron | 使用 JavaScript, HTML 和 CSS 构建跨平台的桌面应用程序](#)
 - 竞品
 - nw.js
 - 现状
 - 已成为开源开发者、初创企业和老牌公司常用的开发工具
 - 详见后续章节: [应用举例](#)
 - 优势
 - 颜值高=界面美观
 - 原因: Web 技术 (HTML + CSS + JS) 天生适合页面信息展示
 - 截图



- 对比其他技术：很多是基于各种图形库开发出的桌面应用，很多颜值一般
- 界面统一
 - Win / Mac / Linux 中显示效果几乎完全一样
 - 典型例子：[VSCode](#)
 - 对比其他技术：界面效果多数不太一样
 - 开发难度低 = 入门快 = 上手方便
 - = 像开发网站一样开发（跨平台桌面）应用
 - 比传统方式要简单很多
 - 说明
 - 传统方式：只支持某个平台，特定语言和框架，才能开发出该平台中的桌面端应用
 - 举例：
 - Win
 - IDE: Visual Studio + 语
言: C# + 框架: WinForm / WPF
 - 截图
 - Expenselt application window titled 'View Expense Report' showing a list of names: Mike, Lisa, John.
 - Mac
 - IDE: XCode + 语
言: Objective-C / Swift + 框
架: Cocoa
 - 截图



- 额外特性
 - 自动更新
 - 支持平台
 - 不支持 Linux
 - 支持 Mac 和 Win
 - 都是基于Squirrel去实现的
 - 原生的菜单和通知
 - 崩溃报告
 - 调试和性能分析
 - Windows 安装程序

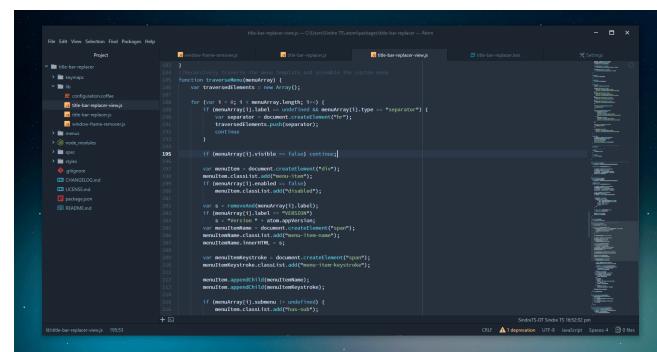
crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-05-29 17:52:20

应用举例

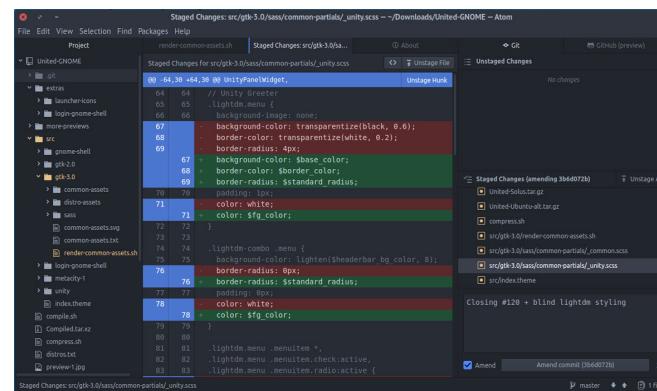
目前已经有众多很出名的跨平台桌面应用都是用 Electron 开发的。

举例如下：

- Atom
 - GitHub 用 Electron 开发了 Atom
 - 截图
 - Win

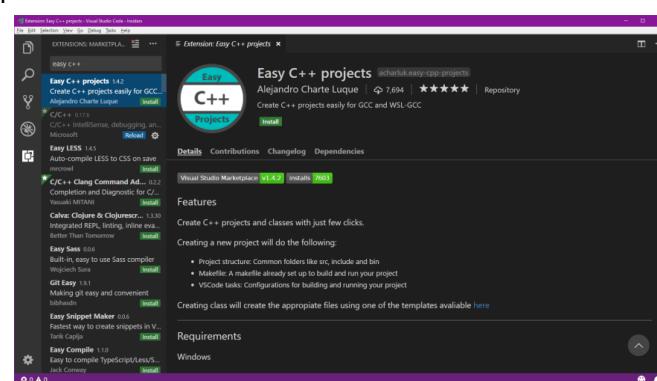


■ Mac

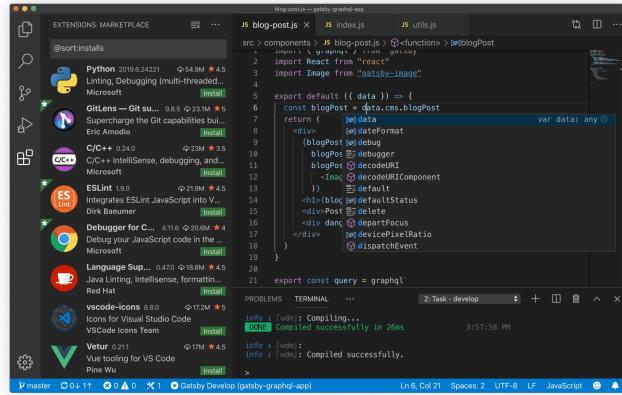


- Visual Studio Code
 - 微软用 Electron 开发了 [Visual Studio Code](#)
 - 截图

■ Win



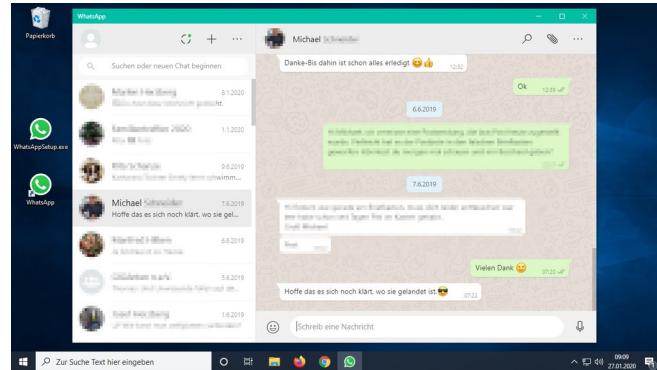
■ Mac



- WhatsApp 桌面版

○ 截图

■ Win

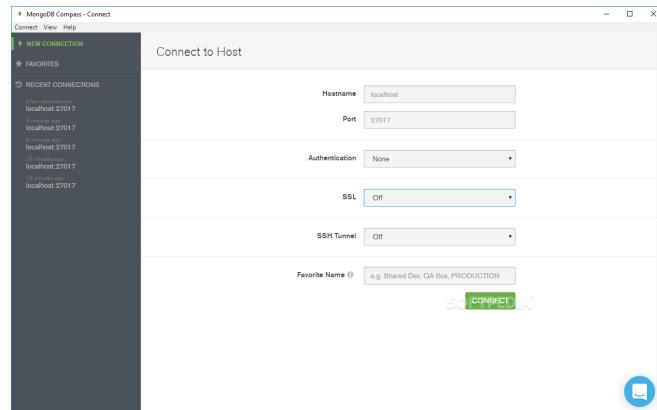


- MongoDB Compass

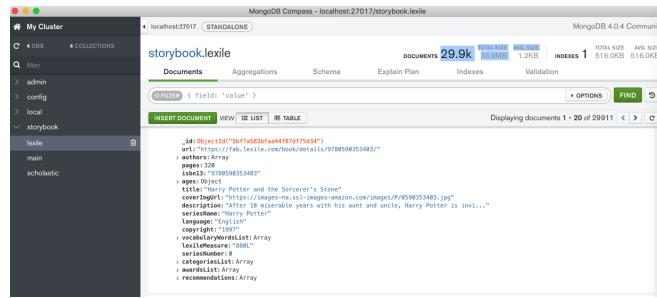
- MongoDB 用Electron开发了 [MongoDB Compass](#)

○ 截图

■ Win



■ Mac



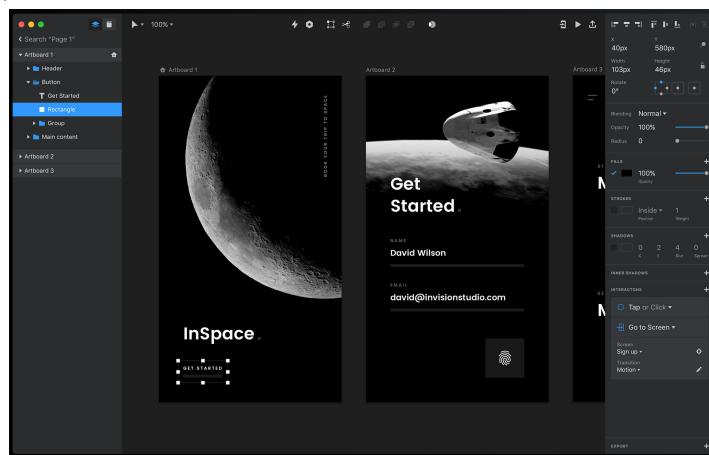
- 迅雷X

- 发布于2018年的迅雷X



- InVision Studio

- 截图



- Slack

- Slack 用 Electron 开发了 Slack

- Microsoft Teams

- Twitch

更多例子详见官网

- 看看谁在使用Electron

- <https://electronjs.org/apps>
 - <https://github.com/electron/apps/tree/master/apps>

单独举例：用Electron可以开发出足够复杂的应用

之前某人用Electron开发出一款足够复杂的应用。举例如下，供了解：

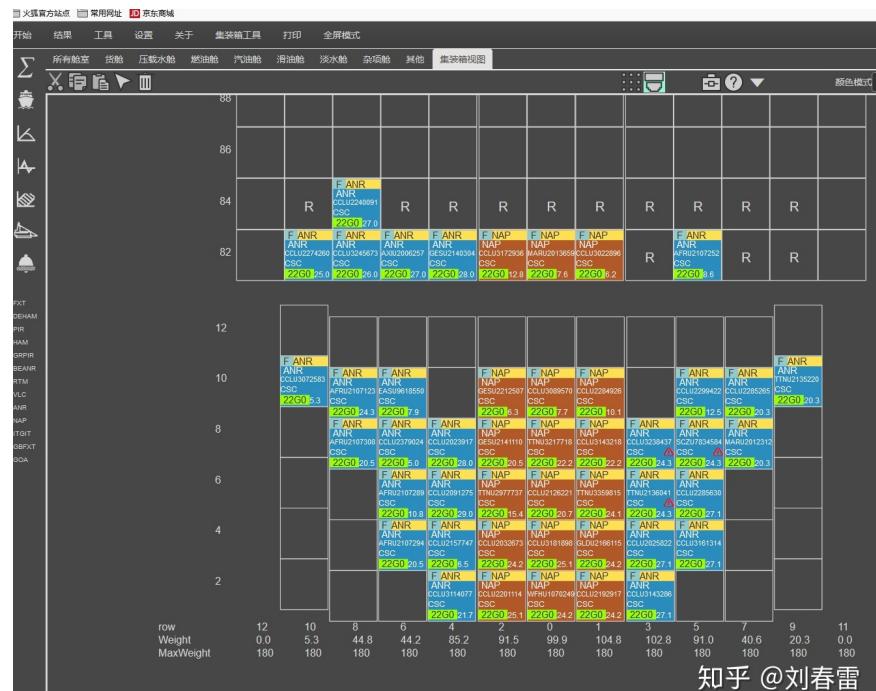
多皮肤实时切换：

The top screenshot shows the "SMARTLOAD" interface. It includes a toolbar with icons for file operations, settings, and printing. The main area displays a "校验结果" (Validation Results) table with various status indicators (OK, 良好, 良好). Below this is a "载重量汇总" (Total Weight Summary) table showing details for different holds. A graph at the bottom left shows weight distribution across the ship's length. On the right, there are sections for "主要尺度及要素" (Main Dimensions and Factors), "排水量" (Displacement), and "GM/GZ" (GM/GZ) calculations.

The bottom screenshot shows a similar interface for the "ZEBUNG" system. It has a similar layout with a toolbar, validation results, and a total weight summary table. The graph and dimension sections are also present, though some labels are in Chinese.

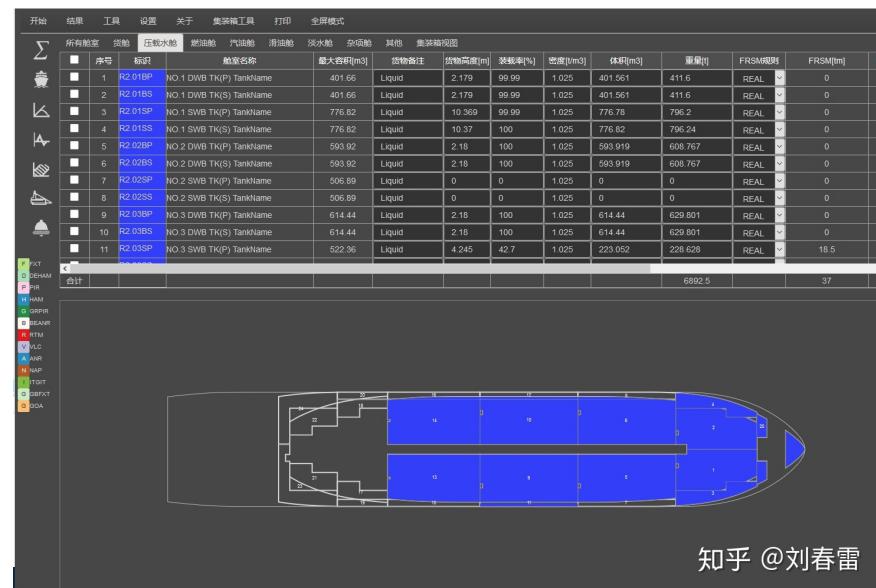
集装箱船 任意放大缩小 拖动：

This screenshot shows a detailed stowage plan for a container ship. The interface includes a toolbar and a navigation menu. The central part of the screen displays a large grid representing the ship's deck, with numerous shipping containers represented by colored rectangles. A sidebar on the left lists various container types and their quantities. On the right, there are several input fields and dropdown menus for specifying shipping details like position, needed, weight, and dimensions. A legend at the bottom provides key for the container colors.

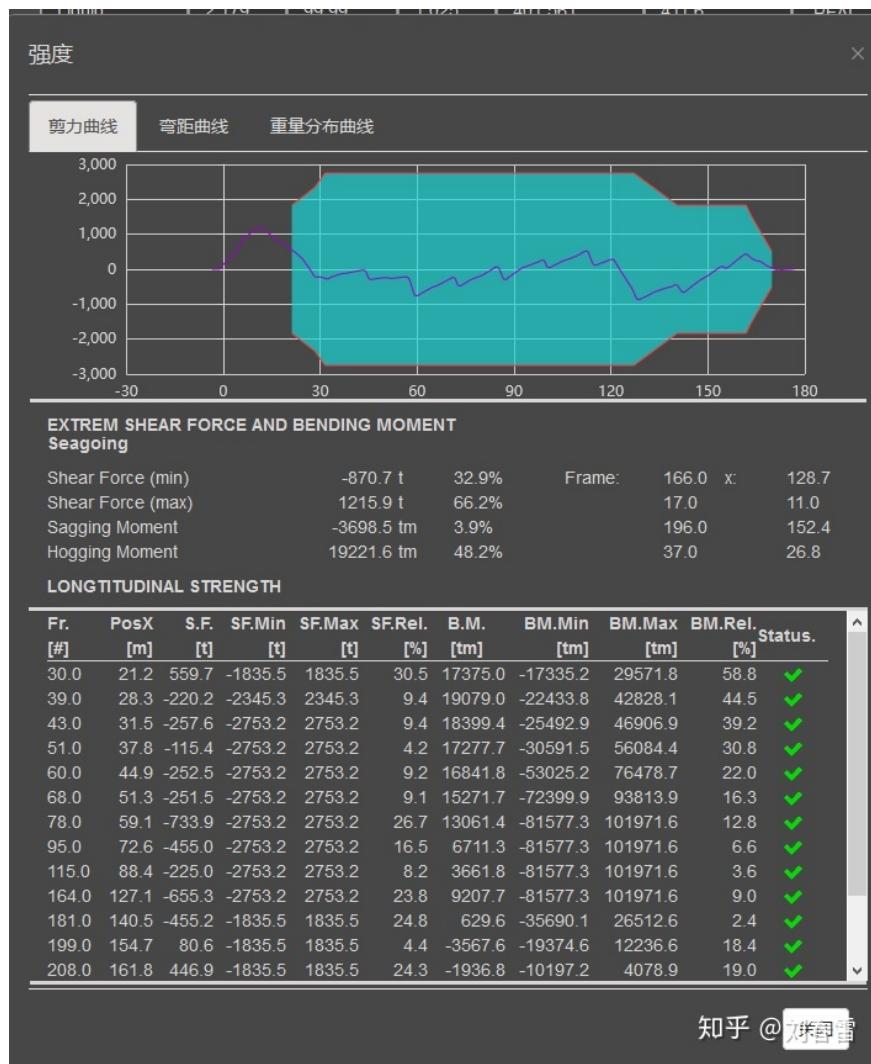


知乎 @刘春雷

为提高用户体验，Firefox 将发送部分功能的使用情况给我们，用于进一步优化火狐浏览器的易用性。您可以自由选择是否向我们分享数据。



知乎 @刘春雷



IMDG隔离表

类别描述	类别	1.1,1.2.1,5	1.3,1.6	1.4	2.1	2.2	2.3	3	4.1	4.2	4.3	5.1	5.2	6.1	6.2	7	8	9
易爆物	1.1,1.2.1.5	*	*	*	4	2	2	4	4	4	4	4	4	2	4	2	4	X
易爆物	1.3,1.6	*	*	*	4	2	2	4	3	3	4	4	4	2	4	2	2	X
易爆物	1.4	*	*	*	2	1	1	2	2	2	2	2	2	2	X	4	2	2
易燃气体	2.1	4	4	2	X	X	X	2	2	2	2	2	2	X	4	2	1	X
无毒	2.2	2	2	1	X	X	X	2	X	X	X	X	X	2	1	X	X	X
有毒气体	2.3	2	2	1	X	X	X	2	X	2	X	X	X	2	1	X	X	X
易燃液体	3	4	4	2	2	1	2	X	X	2	2	2	2	X	3	2	1	X
易燃固体	4.1	4	3	2	2	1	2	2	X	X	X	2	2	X	3	2	1	X
自燃物质	4.2	4	3	2	2	1	2	2	X	X	X	2	2	3	2	1	1	X
潮湿环境下危险物	4.3	4	4	2	2	X	X	2	X	2	X	2	2	X	2	2	1	X
氧化物	5.1	4	4	2	2	X	X	2	1	2	2	2	2	X	3	1	2	X
有机过氧化物	5.2	4	4	2	2	1	2	2	2	2	2	2	2	X	1	3	2	2
有毒物质	6.1	2	2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
传染物	6.2	4	4	4	4	2	2	3	3	3	2	3	3	1	3	2	2	X
放射材料	7	2	2	2	2	2	1	2	2	2	2	2	2	X	3	X	2	X
腐蚀性物品	8	4	2	2	2	X	X	X	X	X	X	X	X	X	3	2	X	X
混合物	9	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

* - 参考易爆物特殊规定 (IMDG准则第7.2.7.2条款)
X - 除非于危险货物清单中特殊声明, 否则无需隔离
1 - 远离
2 - 隔离
3 - 用一个于中间的整个舱室或货舱隔离
4 - 用一个于中间的整个舱室或货舱作纵向隔离

1.1,1.2.1.5 1.1,1.2.1.5

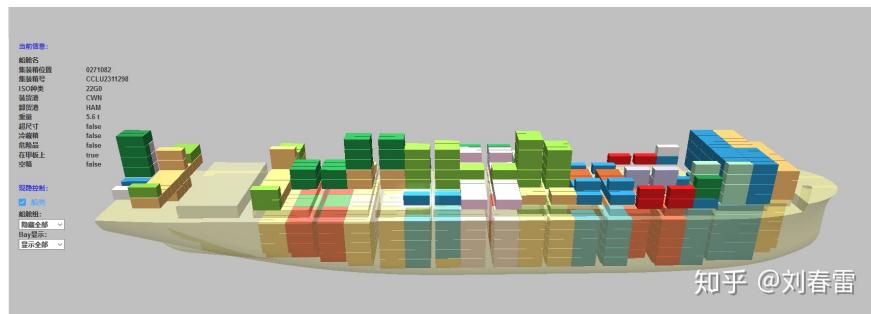
知乎 @刘雷

html5 网页版 安卓版：

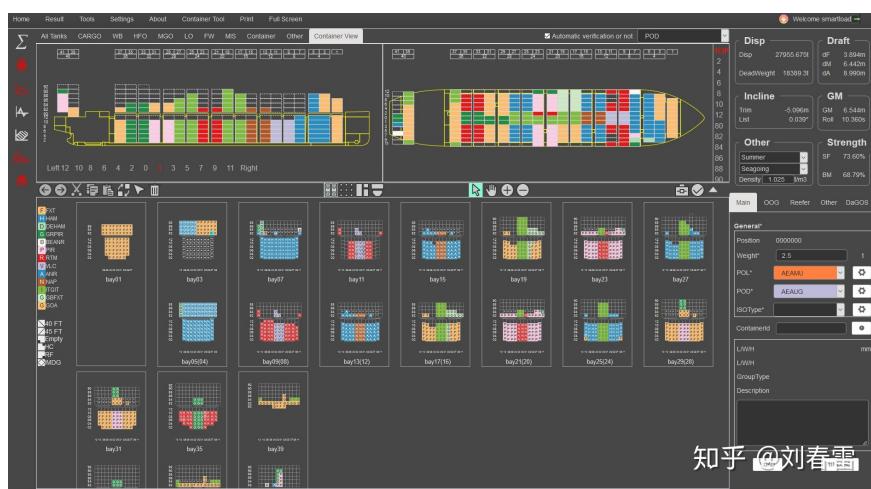


知乎 @刘春雷

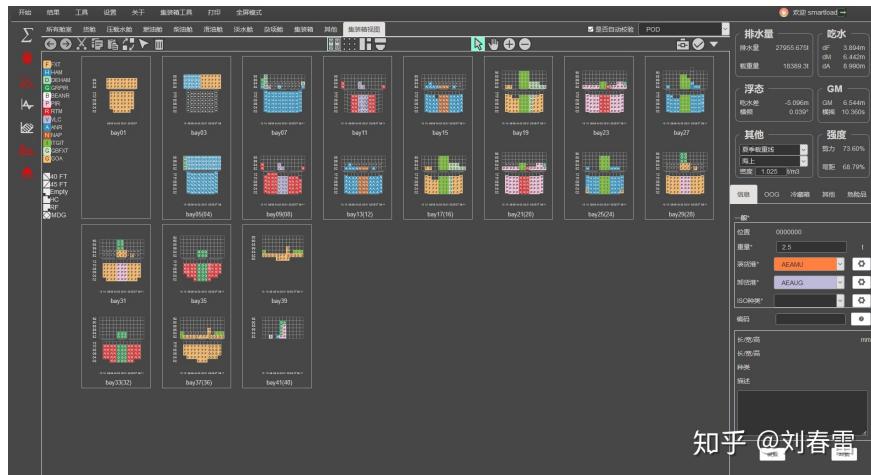
增加了Web 3D显示，放大缩小旋转拾取：



主界面功能布局做了调整：



窗口折叠：



crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新: 2020-05-29 18:17:31

Electron安装

初始化环境

用Electron之前，需要确保本地环境已安装好 npm 和 node

- 确保已安装了 Node.js：

```
node -v
```

如果没有安装，则先去安装：

```
brew install node
```

- 确保已安装了 npm：

```
npm -v
```

安装Electron

```
npm i -D electron@latest
```

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-05-29 23:21:44

Electron基本使用

官网入门教程

官网已有很好的入门教程：

[electron/electron-quick-start: Clone to try a simple Electron app](https://github.com/electron/electron-quick-start)

跟着教程一步步操作即可。

核心步骤：

```
# 克隆仓库
$ git clone https://github.com/electron/electron-quick-start
# 进入仓库
$ cd electron-quick-start
# 安装依赖库
$ npm install
# 运行应用
$ npm start
```

基本效果：



crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-05-29 22:50:13

Electron打包和部署

用Electron开发完桌面应用后，接着需要把代码打包在一起发布出来，叫做：**打包或部署或发布**

常见打包packaging工具

目前主要有3种可以用来打包Electron应用的工具：

- `electron-forge`
 - <https://github.com/electron-userland/electron-forge>
- `electron-builder`
 - 官网
 - [electron-builder](#)
 - GitHub
 - [electron-userland/electron-builder: A complete solution to package and build a ready for distribution Electron app with 'auto update' support out of the box](#)
 - 文档
 - 不同部分，分页显示的
 - 比如：
 - [PlatformSpecificBuildOptions - electron-builder](#)
 - 全部文档在一起的，单页显示的
 - [Common Configuration - electron-builder](#)
- `electron-packager`
 - [electron/electron-packager: Customize and package your Electron app with OS-specific bundles \(.app, .exe, etc.\) via JS or CLI](#)

更多内容详见官网：

- [应用程序打包 | Electron](#)
- [应用部署 | Electron](#)

用`electron-builder`打包

下面举例用 `electron-builder` 去打包Electron的应用。

Mac中用`electron-builder`打包

此处介绍如何在Mac中用 `electron-builder` 去打包Electron的应用，生成Mac中的 `app` 和 `pkg` 文件。

安装electron-builder

```
npm install -D electron-builder@21.2.0
```

添加build配置

以及给 `package.json` 加上必要的 `build` 的配置，主要增加的部分是：

```
{
  "scripts": {
    "pack": "electron-builder --dir",
    "dist": "electron-builder",
    "dist_all": "electron-builder -mw"
  },

  "build": {
    "productName": "ElectronDemo",
    "appId": "com.crifan.electronDemo",
    "copyright": "Copyright © 2020 ${author} String",
    "directories": {
      "output": "release"
    },
    "mac": {
      "category": "public.app-category.developer-tools",
      "type": "distribution",
      "target": [
        "pkg",
        "zip"
      ]
    }
  }
}
```

更新后 `package.json` 完整配置如下：

```
{  
  "name": "electron-quick-start",  
  "version": "1.0.0",  
  "description": "A minimal Electron application",  
  "main": "main.js",  
  "scripts": {  
    "start": "electron .",  
    "pack": "electron-builder --dir",  
    "dist": "electron-builder",  
    "dist_all": "electron-builder -mw"  
  },  
  "repository": "https://github.com/electron/electron-quick-start",  
  "keywords": [  
    "Electron",  
    "quick",  
    "start",  
    "tutorial",  
    "demo"  
  ],  
  "author": "Crifan Li",  
  "license": "CC0-1.0",  
  "devDependencies": {  
    "electron": "^9.0.0",  
    "electron-builder": "^21.2.0"  
  },  
  "build": {  
    "productName": "ElectronDemo",  
    "appId": "com.crifan.electronDemo",  
    "copyright": "Copyright © 2020 ${author} String",  
    "directories": {  
      "output": "release"  
    },  
    "mac": {  
      "category": "public.app-category.developer-tools",  
      "type": "distribution",  
      "target": [  
        "pkg",  
        "zip"  
      ]  
    }  
  },  
  "dependencies": {}  
}
```

打包

然后去打包

```
npm run dist
```

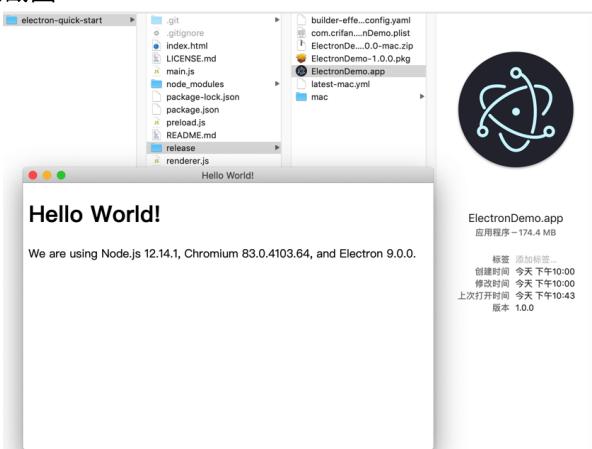
打包后输出文件：

```
□ ~/dev/src/electron/electron-quick-start/release □ □ mast
└─ ElectronDemo-1.0.0-mac.zip
└─ ElectronDemo-1.0.0.pkg
└─ builder-effective-config.yaml
└─ com.crifan.electronDemo.plist
└─ latest-mac.yml
└─ mac
    └─ ElectronDemo.app

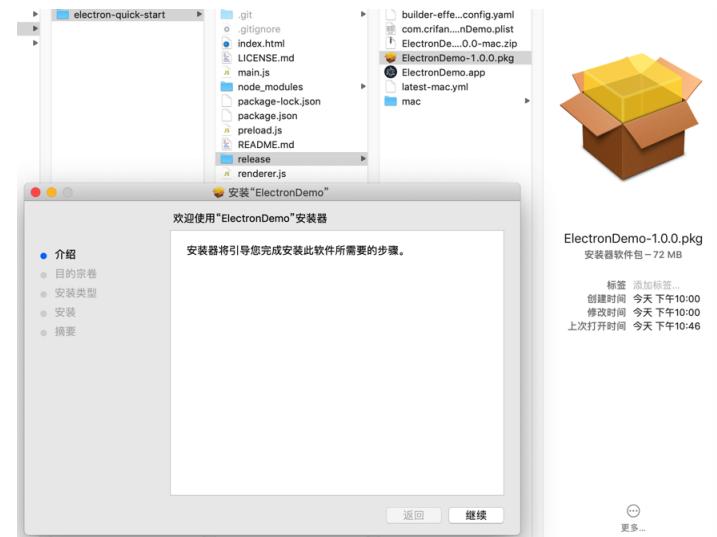
2 directories, 5 files
```

其中：

- ElectronDemo-1.0.0-mac.zip
 - 解压得到 ElectronDemo.app
 - 双击即可运行
 - 效果截图



- ElectronDemo-1.0.0.pkg
 - 双击即可安装



crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-06-18 09:32:08

Electron对python支持

Electron本身官网并没有支持Python。

不过网上有人给出了可行的方案：

- 英文
 - [fyears/electron-python-example: Electron as GUI of Python Applications](#)
- 中文翻译
 - [用 Electron 作为 Python 的 GUI 界面 - 知乎](#)

下面介绍环境搭建的详细过程：

搭建能运行Python的Electron环境

下载代码

```
git clone https://github.com/fyears/electron-python-example
```

得到文件：

```
□ tree -CF
.
├── electron-python-example/
│   ├── LICENSE
│   ├── README.md
│   ├── index.html
│   ├── main.js
│   ├── old-post-backup/
│   │   ├── README.md
│   │   ├── hello.py
│   │   ├── main.js
│   │   └── package.json
│   ├── package.json
│   ├── pycalc/
│   │   ├── api.py
│   │   ├── calc.py
│   │   └── requirements.txt
│   └── renderer.js
└── electron-quick-start/
    ├── LICENSE.md
    ├── README.md
    ├── index.html
    ├── main.js
    ├── package-lock.json
    ├── package.json
    ├── preload.js
    └── renderer.js

4 directories, 21 files
```

用VSCode打开，效果是：



初始化node环境

node要安装版本v8

之前：

```
brew install node
```

安装出的node是 13.5.0，会导致后续编译electron有问题，所以需要换 8.x 版本的 node：

```
brew install node@8
```

安装后使其生效：

把路径加到启动脚本的PATH中：

```
echo 'export PATH="/usr/local/opt/node@8/bin:$PATH"' >>
~/.zshrc
```

立刻生效：

```
source ~/.zshrc
```

安装后的版本是：

- node: 8.17.0
- npm: 6.13.4

根目录 electron-python-example 中：

创建文件 .npmrc，内容为：

```
npm_config_target="2.0.18" # electron version
npm_config_runtime="electron" # 为electron编译
npm_config_disturl="https://atom.io/download/electron" # 资源URL
npm_config_build_from_source="true" # 从源码编译
```

以及配置文件： electron-python-example/package.json

```
{
  "name": "pretty-calculator",
  "version": "1.0.0",
  "description": "A minimal Electron and Python - based calculator",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "repository": "https://github.com/fyears/electron-python-example",
  "keywords": [
    "Electron",
    "Python",
    "zerorpc",
    "demo"
  ],
  "author": "fyears",
  "license": "MIT",
  "dependencies": {
    "zeromq": "^6.0.0-beta.6",
    "zerorpc": "^0.9.8"
  },
  "devDependencies": {
    "electron": "^2.0.18",
    "electron-builder": "^21.2.0",
    "electron-rebuild": "^1.8.8"
  }
}
```

再去安装：

```
npm install
```

初始化Python开发环境

虚拟环境

此处Python的虚拟环境选择用： virtualenv

- 安装：

```
pip install virtualenv
```

- 新建：

```
virtualenv venv
```

- 进入：

```
source venv/bin/activate
```

安装依赖的库

安装负责 js 和 python 之间的通信的 zerorpc：

```
pip install zerorpc
```

启动Electron

每次启动前最好清除缓存

根据[作者](#)解释，每次启动之前，最好去清除一下各种缓存，防止干扰：

```
rm -rf ~/.node-gyp  
rm -rf ~/.electron-gyp  
rm -rf ./node_modules  
rm -f package-lock.json
```

启动electron

```
./node_modules/.bin/electron
```

界面效果：



测试zerorpc通信

再去测试zerorpc通信是否正常，Python代码是否可用：

打开某个终端去：

```
python pycalc/api.py
```

正常输出是：

```
start running on tcp://127.0.0.1:4242
```

再去另外一个终端：

```
virtualenv venv
source venv/bin/activate

zerorpc tcp://localhost:4242 calc "1 + 1"
```

正常输出：

```
connecting to "tcp://localhost:4242"
2.0
```



即表示 zerorpc 通信正常，js 可以和 Python 代码交互了。

然后再去启动electron：

```
./node_modules/.bin/electron .
```

即可看到计算器的界面：



至此即在mac中跑通此处的 electron-python-example 了。

剩下的就是自己实现自己需要的逻辑和功能了。

打包和发布

在实现了自己的业务逻辑和功能后，再去打包和发布。

python的打包

先用 PyInstaller 打包：

```
cd electron-python-example

rm -rf build
rm -rf *.spec
rm -rf pymitmdumpstartdist
rm -rf pymitmdumpotherdist

pyinstaller pymitmdump/mitmdumpStartApi.py --distpath pymitmdump_start
pyinstaller pymitmdump/mitmdumpOtherApi.py --distpath pymitmdump_other
```

此处会生成的2个dist目录：

- pymitmdumpstartdist
 - 二进制文件： electron-python-example/pymitmdumpstartdist/mitmdumpStartApi/mitmdumpStartApi
 - 以及我们拷贝的：
 - 整个目录： electron-python-example/pymitmdumpstartdist/mitmdumpStartApi/mitmdump_executable
 - 单个py文件： electron-python-example/pymitmdumpstartdist/mitmdumpStartApi/mitmdumpUrlSaver.py
- pymitmdumpotherdist
 - 二进制文件： electron-python-example/pymitmdumpotherdist/mitmdumpOtherApi/mitmdumpOtherApi

electron的打包

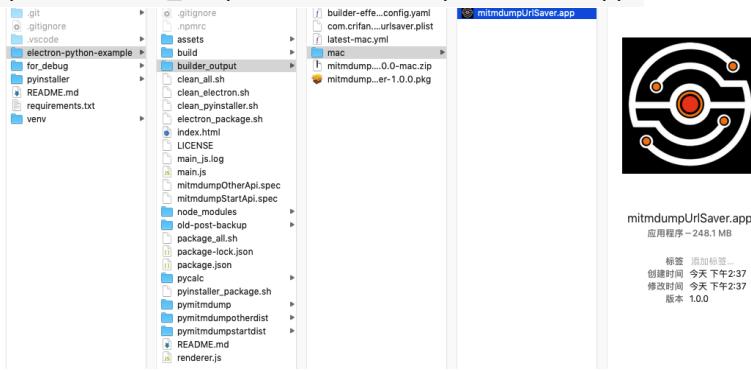
再去用 electron-builder 打包：

```
rm -rf builder_output

npm run dist
```

生成：

- app: electron-python-example/builder_output/mac/mitmdumpUrlSaver.app



- app的zip压缩包: electron-python-example/builder_output/mitmdumpUrlSaver-1.0.0-mac.zip
- pkg: electron-python-example/builder_output/mitmdumpUrlSaver-1.0.0.pkg
 - 注: Mac中安装后,但在应用程序中找不到对应的app。暂未深究,原因未知,可能和code sign有关系。

运行

双击打包出来的app即可运行:



其他说明

package.json 最新完整配置及说明

此处最后的 package.json 配置为:


```

    "!build",
    "!pymitmdump",
    "!pycalc",
    "!old-post-backup",
    "!*.*log",
    "!*.*txt",
    "!*.*md",
    "!*.*spec"
  ],
  "asar": false,
  "mac": {
    "category": "public.app-category.developer-tools",
    "type": "distribution",
    "icon": "assets/icon/icon.icns",
    "target": [
      "pkg",
      "zip"
    ]
  },
  "win": {
    "target": "nsis",
    "icon": "assets/icon/logo.png"
  }
}
}

```

(1) file

其中

```

"**/*",
"!**/node_modules/*/{CHANGELOG.md,README.md,README,readme.r
"!**/node_modules/*/{test,__tests__,tests,powered-test,exar
"!**/node_modules/*.d.ts",
"!**/node_modules/.bin",
"!**/*.{iml,o,hprof,orig,pyc,pyo,rbc,swp,csproj,sln,xproj}"
"!.editorconfig",
"!**/._*",
"!**/.DS_Store,.git,.hg,.svn,CVS,RCS,SCCS,.gitignore,.gita
"!**/{__pycache__,thumbs.db,.flowconfig,.idea,.vs,.nyc_out}
"!**/{appveyor.yml,.travis.yml,circle.yml}",
"!**/{npm-debug.log,yarn.lock,.yarn-integrity,.yarn-metadata

```

是从官网 [Application Contents - electron-builder](#) 拷贝出来的默认的配置：

余下部分：

```
  "!build",
  "!pymitmdump",
  "!pycalc",
  "!old-post-backup",
  "!.log",
  "!.txt",
  "!.md",
  "!.spec"
```

是自己加的，用于排除此处不需要打包的文件。

(2) asar

asar 参数默认是开启的，其会导致此处二进制可执行文件：

- mitmdumpUrlSaver.app/Contents/Resources/app/pymitmdumpstartd
ist/mitmdumpStartApi/mitmdumpStartApi
 - 无法运行
- mitmdumpUrlSaver.app/Contents/Resources/app/pymitmdumpother
dist/mitmdumpOtherApi/mitmdumpOtherApi
 - 可以运行

根本原因：未知

规避办法：只能去关闭掉asar：

```
"asar": false,
```

才能正常运行mitmdumpStartApi和mitmdumpOtherApi

更多细节

详见：

【已解决】Mac中运行跑通electron-python-example的示例

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 13:54:29

Electron心得

此处整理Electron开发期间的一些心得。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-06-17 16:39:48

开发

此处整理关于Electron开发期间的一些心得。

Uncaught ReferenceError process is not defined

- 问题：文件： `electron-python-example/index.html` 的代码：

```
<script>
  require('./renderer.js')
</script>
```

报错：`Uncaught ReferenceError process is not defined`

- 原因：Electron 5.0 之后，把之前默认是开启的 `nodeIntegration` 关闭了，导致找不到进程而报错
- 解决办法：加上参数去开启：`nodeIntegration:true`

文件：`electron_python/electron-python-example/main.js`

```
const createWindow = () => {
  mainWindow = new BrowserWindow(
    {
      width: 800,
      height: 600,
      webPreferences: {
        nodeIntegration: true,
      }
    }
  )
}
```

textarea右键复制选中内容

- 背景：希望 `textarea` 区域可以右键显示按钮，带复制选项，用于复制所选内容
- 解决办法：

安装插件 [sindresorhus/electron-context-menu: Context menu for your Electron app](#)，即可实现此功能。

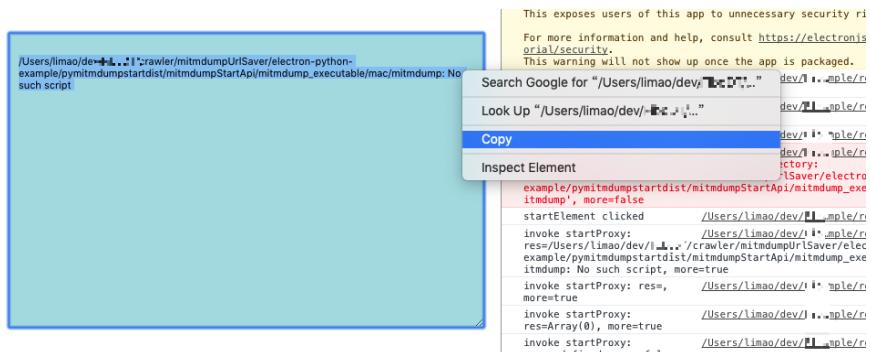
- 步骤：

```
npm install electron-context-menu
```

使用：

```
const contextMenu = require('electron-context-menu');
contextMenu({
  prepend: (defaultActions, params, browserWindow) => [
    {
      label: 'Rainbow',
      // Only show it when right-clicking images
      visible: params.mediaType === 'image'
    },
    {
      label: 'Search Google for "{selection}"',
      // Only show it when right-clicking text
      visible: params.selectionText.trim().length > 0
      click: () => {
        shell.openExternal(`https://google.com/search?q=${params.selectionText}`);
      }
    }
  ]
});
```

Electron中的textarea，即可支持右键，出现菜单，选择文字后，即可复制：



进一步的需求：禁止右键中除了 Copy 外其他按钮菜单

去加上配置：

```
const contextMenu = require('electron-context-menu');
contextMenu({
  showLookUpSelection: false,
  showCopyImage: false,
  showInspectElement: false,
});
```

效果：



crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-06-18 15:21:58

Web技术

由于Electron的开发，关于内容显示方面，来说就是Web技术的开发。所以常会涉及到一些Web技术的使用。现整理部分如下：

html的文本区域界面用于显示js中获取到的字符串

文件： `electron-python-example/index.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Mitmdump代理保存URL</title>

    <style>

      h1 {
        text-align: center;
      }

      #mitm {
        text-align: center;
      }

      #startSaver {
        font-size: 14px;
        height: 40px;
        background-color: mediumspringgreen;
        border-radius: 6px;
      }

      #output {
        margin-top: 20px;

        background-color: powderblue;

        border: 1px solid #ccc;
        padding: 10px;
        /* width: 400px; */
        /* min-width: 800px; */
        width: 500px;
        min-height: 300px;
        /* height: auto; */
        /* 可以设置一个最大高度，超出时滚动，否则，高度会被撑开 */
        /* max-height: 300px; */
        overflow: auto;
      }
    </style>

  </head>

  <body>
    <h1>Mitmdump代理保存URL</h1>
    <ul>
      <li>Node.js: <script>document.write(process.versions.
      <li>Chromium: <script>document.write(process.versions.
      <li>Electron: <script>document.write(process.versions.
    </ul>
```

```
<div id="mitm">
  <button id="startSaver">启动代理</button>
  <div>
    <!-- <textarea readonly id="output" rows="100" cols="100"></textarea>
    <textarea readonly id="output"></textarea>
  </div>
</div>

</body>

<script>
  require('./renderer.js')
  // require('electron-python/electron-python-example/renderer.js')
</script>
</html>
```

文件: electron-python-example/renderer.js

```

const zerorpc = require("zerorpc")
// let client = new zerorpc.Client()
const constLargeEnoughHeartbeat = 60 * 60 * 24 * 30 * 12 /,
clientOptions = {
  "heartbeatInterval": constLargeEnoughHeartbeat,
}
let client = new zerorpc.Client(clientOptions)

client.connect("tcp://127.0.0.1:4242")

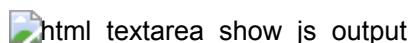
client.invoke("echo", "server ready", (error, res) => {
  if(error || res !== 'server ready') {
    console.error(error)
  } else {
    console.log("server is ready")
  }
})

let startElement = document.querySelector('#startSaver')
let outputElement = document.querySelector('#output')

startElement.addEventListener('click', () => {
  console.log("startElement clicked")
  client.invoke("startSaver", (error, res, more) => {
    if(error) {
      console.error("invoke startSaver: error=%s, more=%s",
    } else {
      console.log("invoke startSaver: res=%s, more=%s", res,
      outputElement.textContent += "\n" + res
      // outputElement.textContent += res
      outputElement.scrollTop = outputElement.scrollHeight
    }
  })
})
})

```

效果:



crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新: 2020-06-18 15:20:37

Log日志

此处整理一些和log日志, `print`, `console.log` 等相关的内容:

让main.js即js输出log日志文件到当前文件夹

文件: `electron-python-example/main.js`

```
const path = require('path')

let logFilename = path.join(__dirname, "main_js.log")
let fs = require('fs');
let util = require('util');

var logFile = fs.createWriteStream(logFilename, { flags: 'a' });
// Or 'w' to truncate the file every time the process starts
var logStdout = process.stdout;
console.log = function () {
  logFile.write(util.format.apply(null, arguments) + '\n');
  logStdout.write(util.format.apply(null, arguments) + '\n'
}
console.error = console.log;
```

实现了之后的`console.log`:

```
console.log("main.js: __dirname=%s", __dirname)
```

可以同时输出到:

- `console`终端
- log文件

让main.js中的`console.log`输出到Electron的devTool中的`console`的log中

- 背景:

此处已有:

文件: `electron-python-example/main.js`

```
...
const createWindow = () => {
  mainWindow = new BrowserWindow(
    {
      // width: 800,
      // height: 600,
      width: 1024,
      height: 768,
      // transparent: true,
      webPreferences: {
        nodeIntegration: true,
      }
    }
  )
  mainWindow.loadURL(require('url').format({
    pathname: path.join(__dirname, 'index.html'),
    protocol: 'file:',
    slashes: true
  }))
  mainWindow.webContents.openDevTools()

  mainWindow.on('closed', () => {
    mainWindow = null
  })
}
```

和文件: `electron-python-example/renderer.js`

其中

文件: `electron-python-example/index.html`

```
...
<script>
  require('./renderer.js')
</script>
</html>
```

可见: `index.html`中引用了`renderer.js`

去

```
./node_modules/.bin/electron
```

调试`electron`时, 有2种log:

- Electron界面中, 右边可以看到html调用的 `renderer.js` 输出的log

- 原因：由于 `main.js` 中的 `mainWindow.webContents.openDevTools()` 打开了 DevTools
- 调试终端中：可以看到 `main.js` 的 log
 - 因为是在VSCode的终端中运行的



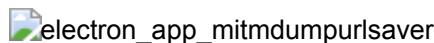
而mac中打包后的Electron的app中，只能看到 Electron界面中（`render.js` 打印）的log，看不到`main.js`中的log：



而此处希望，对于production生产模式，即mac中打包electron后得到的app文件 `mitmdumpUrlSaver.app` 在其运行时也可以看到 `main.js` 中的log，以便于后续调试，当出错时找到原因。

- **最终方案：**

从界面中启动打包后的app内部的二进制文件，比如我的 `mitmdumpUrlSaver.app`



对应内部的二进制文件

```
./mitmdumpUrlSaver.app/Contents/MacOS/mitmdumpUrlSaver
```



即可启动终端，运行此处的APP。终端中可以看到我们要的 `main.js` 中的log



- **进一步：**如果想要在当前终端中看到Electron的`render.js`中的log

则可以启动之前加上环境变量：

```
export ELECTRON_ENABLE_LOGGING=true  
./mitmdumpUrlSaver.app/Contents/MacOS/mitmdumpUrlSaver
```

注：可以用echo确认变量是否已设置成功

```
□ echo $ELECTRON_ENABLE_LOGGING  
true
```

然后就也能看到 `render.js` 中的log了：



相关说明：

此处2个js文件，对应着2个js线程：

- main.js -> node.js 的主线程
 - -» main.js 中 console.log 输出到的地方
- renderer.js -> 显示界面的渲染线程 = Electron界面所在的线程
 - -» (调试时) Electron界面开启了 DevTools 后右边就能看到 console.log 输出的内容了
 -  see_log_after_toggle_devtools

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 15:22:22

打包

关于Electron打包方面，总结一些心得和坑。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-17 17:23:01

制作app的Logo

给Electron打包生成可执行文件期间，会涉及到给打包后的app弄个Logo图标。

Mac中可以用工具：

- [Image2icon](#)
 - 官网
 - [Image2icon - Your Mac. Your Icons](#)

去制作app的logo图标。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新： 2020-06-17 17:21:23

asar

- asar简介
 - 是什么：一种压缩文件格式
 - 目的=为何要压缩：
 - 减小文件体积
 - 打包后的应用文件尽量小，不要太大
 - 避免暴露源码
 - 特点
 - Electron 可以无需解压整个文件，即可从其中读取任意文件内容

如何使用=开启asar

为了使用一个 asar 档案文件代替 app 文件夹，你需要修改这个档案文件的名字为 app.asar，

然后将其放到 Electron 的资源文件夹下，然后 Electron 就会试图读取这个档案文件并从中启动。

asar 目录结构：

- Mac

```
electron/Electron.app/Contents/Resources/  
└── app.asar
```

- Window 和 Linux

```
electron/resources/  
└── app.asar
```

更多细节详见：

[应用程序打包 | Electron](#)

asar使用心得

开启asar但不压缩部分文件和目录

有些时候需要实现，虽然整体开启asar，但是不想asar压缩部分内容，则可以用： `asarUnpack`

举例：

```
"asar": true,  
"asarUnpack": [  
    "main.js",  
    "pymitmdumpstartdist",  
    "pymitmdumpotherdist"  
],
```

确保预期效果：打包后的mac的app中有了2个dist目录和一个文件：

- electron-python-example/builder_output/mac/mitmdumpUrlSaver.app/Contents/Resources/app.asar.unpacked/
 - pymitmdumpstartdist
 - pymitmdumpotherdist
 - main.js

从而确保 main.js 运行期间去判断对应的PyInstaller打包后的python的二进制文件

- xxx/builder_output/mac/mitmdumpUrlSaver.app/Contents/Resources/app.asar/
 - pymitmdumpstartdist/mitmdumpStartApi/mitmdumpStartApi
 - pymitmdumpotherdist/mitmdumpOtherApi/mitmdumpOtherApi

都可以被

```
isPackaged = require('fs').existsSync(fullPath)
```

识别到，确保后续代码逻辑可以正常运行。

详见：

【已解决】 Mac中electron-python用electron-builder打包后app运行失败： Error Lost remote after 10000ms

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新： 2020-06-18 14:07:44

files

`package.json` 配置参数中有个 `files`，用来控制要打包的文件。此处整理相关心得。

Application entry file `main.js` in the `.app/Contents/Resources/app.asar` does not exist

- 问题：在改了配置为

```
"files": [  
    "pymitmdumpstartdist",  
    "pymitmdumpperdist"  
,
```

后，打包报错：

```
x Application entry file "main.js" in the "xxx/mac/mitmdump"
```

- 原因：根据官网

`files` - Application Contents - electron-builder 解释，默认 `files` 配置是：

```
"files": [  
    "**/*",  
    "!**/node_modules/*/{CHANGELOG.md,README.md,README,readme,  
    !**/node_modules/*/{test,__tests__,tests,powered-test,  
    !**/node_modules/*.d.ts",  
    !**/node_modules/.bin",  
    !**/*.{iml,o,hprof,orig,pyc,pyo,rbc,swp,csproj,sln,so,  
    ".editorconfig",  
    !**/._*",  
    !**/{.DS_Store,.git,.hg,.svn,CVS,RCS,SCCS,.gitignore,  
    !**/{__pycache__,__thumbs.db,.flowconfig,.idea,.vs,.nyc,  
    !**/{appveyor.yml,.travis.yml,circle.yml",  
    !**/{npm-debug.log,yarn.lock,.yarn-integrity,.yarn-r  
,
```

其中的 `**/*` 包含了根目录下的各种核心文件，包括入口文件 `main.js`

而前面的配置：

```
"files": [  
    "pymitmdumpstartdist",  
    "pymitmdumpotherdist"  
],
```

表示：去掉其他文件，只包含上述2个文件夹

导致找不到必要的入口文件 `main.js`（等其他文件）

- 解决办法：把自己的配置 加到 默认配置上：

即可。

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新: 2020-06-18 16:22:40

electron-builder

signing Error Command failed codesign sign

- 问题：用 `npm run dist` 打包报错：

```
signing           file=release/mac/ElectronDemo.app ident  
Error: Command failed: codesign --sign xxx --force
```

- 原因：code sign方面的错误。
- 解决办法：暂未深入研究，所以只是去禁止code sign，以规避此问题。
 - 禁止code sign的方法
 - 先去：`export CSC_IDENTITY_AUTO_DISCOVERY=false`
 - 再继续打包：`npm run dist`

make-dir.js 86 catch SyntaxError Unexpected token {

- 问题：打包：

```
npm run dist
```

报错：

```
/Users/limao/dev/src/electron/electron-quick-start/node  
} catch {  
^  
SyntaxError: Unexpected token {
```

- 原因：之前20200529，用：

```
npm install -D electron-builder
```

安装出的是最新版本的 `electron-builder`：

```
"dependencies": {  
  "electron-builder": "^22.6.1"  
}
```

而新版 `22.6.1` 有bug。

- 解决办法：换个别的稍微旧一点的版本，比如 20.44.4：

```
npm uninstall electron-builder
npm install -D electron-builder@20.44.4
```

Error loading Python lib /private/var/folders .Python dlopen image not found

- 问题：electron-builder 去打包mac的app后运行报错

```
curPyProc.spawnargs=/var/folders/gt/5868sbcd1jq4rxvryqhy2_:
curPyProc.pid=81849
curPyProc stderr data: chunk=[81849] Error loading Python :
curPyProc exit: exitCode=255
```

- 原因：此处app中用到了已打包好的 .Python ，但是却没有被打包进去，所以运行报错找不到。
- 解决办法：去把缺少的东西打包进来，相关参数： extraFiles 。

此处额外的所需要的事，项目根目录下的（之前用PyInstaller打包后的）2个dist文件夹：

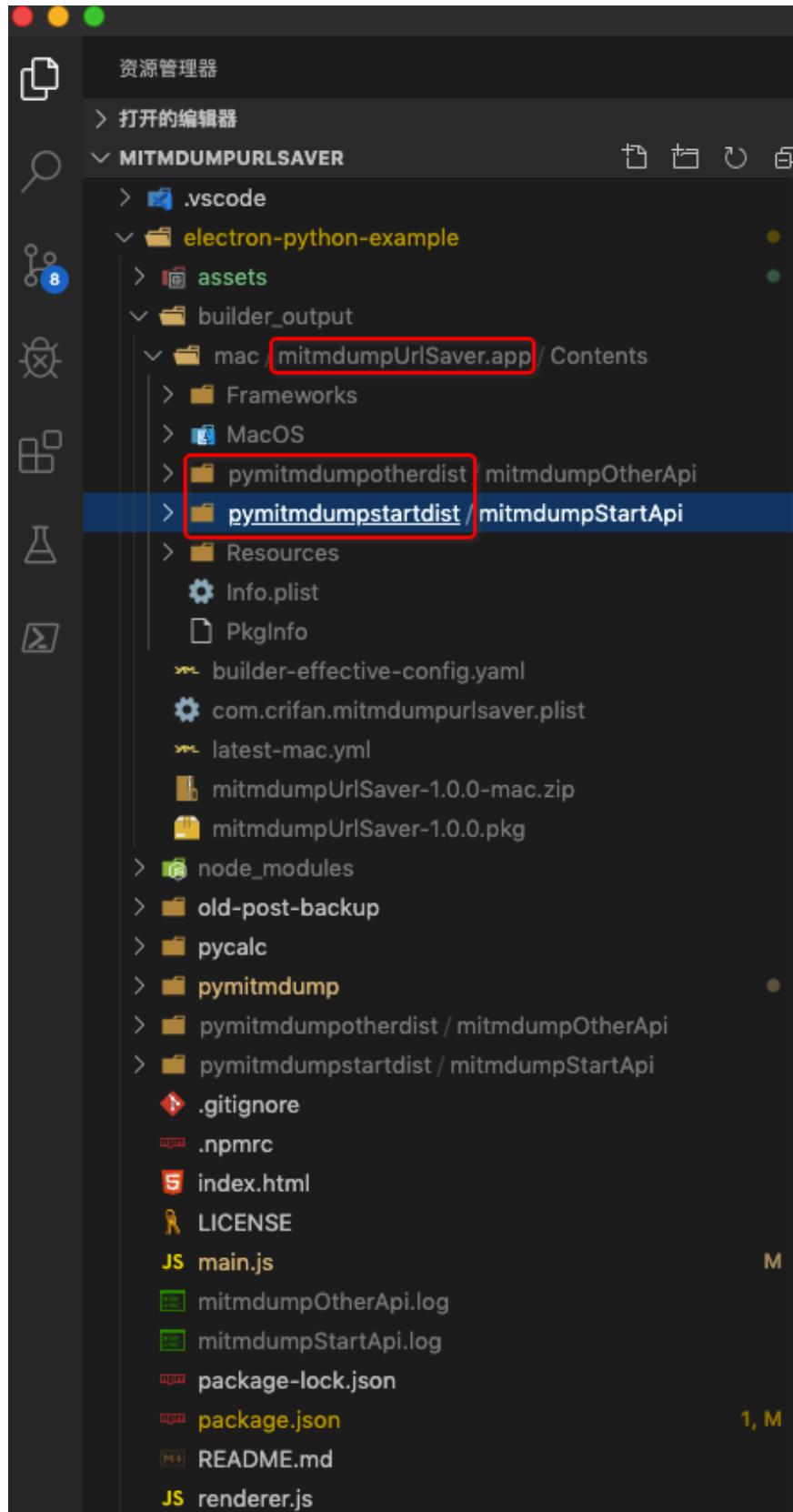
- pymitmdumpotherdist
- pymitmdumpstartdist

具体配置是： build 中 mac 的 extraFiles

文件： electron-python-example/package.json

```
"build": {
...
  "mac": {
...
    "extraFiles": [
      "pymitmdumpstartdist",
      "pymitmdumpotherdist"
    ],
    "target": [ "pkg", "zip" ]
  }
}
```

如此打包后的app，其中就包括了额外的目录：



-》即可正常运行，没了 .Python 找不到的问题。

打包期间实现拷贝整个目录和其中二进制文件到打包后的特定目录

- 背景：希望实现把打包前的整个目录 `electron-python-example/pymitmdump/mitmdump_executable` 及其中的子文件夹和文件，拷贝到打包后的位置 `electron-python-example/builder_output/mac/mitmdumpUrlSaver.app/Contents/Resources/app/pymitmdumpstartdist/mitmdumpStartApi/mitmdump_executable`，从而希望后续代码能识别到。
- 解决办法：通过 `extraResources` 设置

文件：`electron-python-example/package.json`

```
"build": {  
  ...  
  "extraResources": [  
    {  
      "from": "pymitmdump/mitmdump_executable",  
      "to": "app/pymitmdumpstartdist/mitmdumpStartApi/mitmdump_executable",  
      "filter": ["**/*"]  
    }  
  ],
```

路径中包含中文会无法启动mitmdump代理

- 问题：Electron打包的app放到别的mac中测试时，路径中有中文，导致无法正常启动mitmdump代理。
- 原因：路径中包含中文，内部启动mitmdump代理的代码，不支持非中文字符。
- 解决办法：加上兼容不同编码的字符的路径，即可：

```
import locale

curPlatformEncoding = locale.getpreferredencoding()
logging.info("curPlatformEncoding=%s", curPlatformEncoding)
# Mac: UTF-8
# Wind: cp936 ~= GBK ~= GB2312

curProcess = subprocess.Popen(
    shellCmd,
    stdout=subprocess.PIPE,
    stderr=subprocess.STDOUT,
    universal_newlines=True,
    # encoding="UTF-8",
    encoding=curPlatformEncoding,
    shell=isUseShell,
)
logging.debug("curProcess=%s", curProcess)

while True:
    curLineOutput = curProcess.stdout.readline()
```

实现了：自动判断当前系统编码

- Mac: UTF-8
- Win: cp936
 - = GBK

实现 `subprocess` 的 `readline` 内部自动解码，而不会出现解码报错问题。

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新: 2020-06-18 16:37:21

electron-rebuild

build.sh cmake command not found make libzmq Error 127

- 问题: 用 electron-rebuild 编译报错: build.sh cmake command not found make libzmq Error 127
- 原因: 缺少 cmake
- 解决办法:

```
brew install cmake
```

zeromq vendor napi.h error unknown type name napi_callback_scope

- 问题: 用 electron-rebuild 编译报错: zeromq vendor napi.h error unknown type name napi_callback_scope
- 原因: 未深究
- 解决办法: 无需解决, 后续继续:

```
npm install  
./node_modules/.bin/electron-rebuild
```

是可以正常启动Electron的。

.electron-gyp node v8.h error unterminated conditional directive #ifndef INCLUDEV8_H

- 问题:

用:

```
./node_modules/.bin/electron-rebuild
```

报错:

```

../binding.cc:1610:23: error: expected '}'  

/Users/limao/.electron-gyp/7.1.7/include/node/v8.h:37:14: namespace v8 {  

^  

1 warning and 13 errors generated.  

make: *** [Release/obj.target/zmq/binding.o] Error 1  

gyp ERR! build error  

gyp ERR! stack Error: `make` failed with exit code: 2  

gyp ERR! stack     at ChildProcess.onExit (/Users/limao/dev/crifan/mitmdump/node-gyp/lib/gyp/runner.js:102:19)  

gyp ERR! stack     at emitTwo (events.js:126:13)  

gyp ERR! stack     at ChildProcess.emit (events.js:214:7)  

gyp ERR! stack     at Process.ChildProcess._handle.onexit (internal/child_process.js:271:12)  

gyp ERR! System Darwin 18.7.0  

gyp ERR! command "/usr/local/Cellar/node@8/8.17.0/bin/node"  

gyp ERR! cwd /Users/limao/dev/crifan/mitmdump/mitmdumpUrls  

gyp ERR! node -v v8.17.0  

gyp ERR! node-gyp -v v6.0.1  

gyp ERR! not ok

```

- 原因:

- 表面上是node的v8.h头文件有问题
 - 实际上：的确也是v8.h文件内容有问题
 - 此处内容不全，只有7000多行，实际上完整的文件大概1万多行
- 但是根本原因在于：
 - 此处的node版本是v8.17.0
 - 最高支持的electron版本是2.0.18
 - 而当前设置的electron的版本是7.1.2

- 解决办法：去把 electron 版本从 7.1.2 改为 2.0.18

修改后：

文件： electron_python/electron-python-example/.npmrc

```

npm_config_target="2.0.18" # electron version
npm_config_runtime="electron" # 为electron编译
npm_config_disturl="https://atom.io/download/electron" # 资源
npm_config_build_from_source="true" # 从源码编译

```

文件： electron_python/electron-python-example/package.json

```
{  
...  
  "devDependencies": {  
    "electron": "^2.0.18",  
  ...  
  }  
}
```

即可。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-06-17 18:19:03

不同系统

Electron支持不同的系统的开发和打包。

常见的有：

- Win
- Mac
- Linux

等，下面分别总结相关经验心得。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 10:21:52

Win系统

此处整理关于Windows平台中开发Electron的相关心得。

Win中搭建Electron-Python开发环境

把 node 10 换成 node 8 :

【已解决】windows10中重新安装node把node 10换成node 8

```
npm install
```

去初始化

运行Electron本身demo：

```
.\node_modules\.bin\electron
```

运行当前Electron项目：

```
.\node_modules\.bin\electron
```

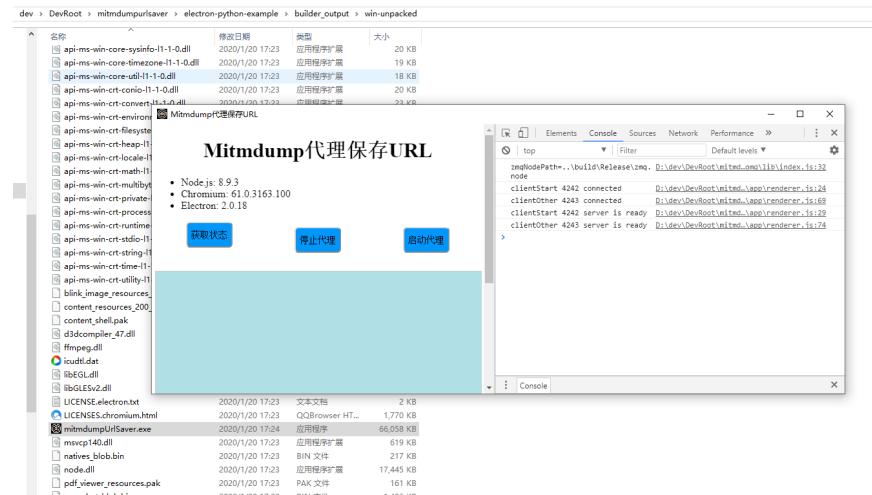
打包：

```
npm run dist
```

注：无需 npm run dist_all

即可打包出exe：

双击运行： D:\dev\DevRoot\mitmdumpurlsaver\electron-python-example\builder_output\win-unpacked\mitmdumpUrlSaver.exe



windows中启动Electron-python报错： Uncaught Error A dynamic link library DLL initialization routine failed zerorpc zeromq zmq.node

- 问题：

文件： electron-python-example\node_modules\zerorpc\node_modules\zeromq\lib\index.js

```
var EventEmitter = require('events').EventEmitter
, zmq = require('../build/Release/zmq.node')
, util = require('util');
```

中的：

```
var zmq = require('../build/Release/zmq.node');
```

运行报错：

```
Uncaught Error: A dynamic link library (DLL) initialization
\\?\D:\dev\DevRoot\mitmdumpurl saver\electron-python-example
```

- 原因：

经过一番网上搜索，没人给出解决办法。

后来自己是从

```
Uncaught Error: A dynamic link library (DLL) initialization
routine failed.
```

加上输出的错误路径

```
\?\D:\dev\DevRoot\mitdumpurl saver\electron-python-example\node_modules\zerorpc\node_modules\zeromq\build\Release\zmq.node
```

中的后面的部分

```
D:\dev\DevRoot\mitdumpurl saver\electron-python-example\node_modules\zerorpc\node_modules\zeromq\build\Release\zmq.node
```

看出是正确的路径，所以怀疑是否是路径方面的问题。后经研究的确就是路径问题。

- **解决办法：**

之前写法是linux中写法，改为同样适配windows的写法：

```
let path = require('path')
let zmqNodePath = path.join("../", "build", "Release", "zmq")
console.log("zmqNodePath=%s", zmqNodePath)
var zmq = require(zmqNodePath)
```

或：

```
let osIsWin = os.platform() === "win32"
let zmqNodePath = undefined
if (osIsWin){
  let zmqNodePath = '../\\build\\Release\\zmq.node'
} else {
  let zmqNodePath = '../build/Release/zmq.node'
}
var zmq = require(zmqNodePath)
```

即可。

运行时Win的exe所在路径

背景：Win中Electron打包后的可执行文件格式一般是 `exe`。

需求：希望在运行期间，获取到当前的实际的exe所在目录的路径。

心得：双击 `exe` 运行后，其内部会解压到临时目录，一般是当前用户的 `AppData` 目录，类似于这种：

```
C:\Users\xxx\AppData\Local\Programs\
```

-» 如果想要获取原本（双击以启动运行的）`exe`所在目录，则就很困难了。

期间去试过参数 `--win portable`，对应配置：

```
"win": {
  "target": "portable",
  "icon": "assets/icon/logo.png"
}
```

结果却是，虽然打包出的exe无需安装，双击即可运行，但是运行期间解压的目录是这种更彻底的临时目录：

`C:\Users\xx~1\AppData\Local\Temp\1Wgi1npnbDTNi0qe7pBCQL1FL0Z`

名称	修改日期	类型	大小
api-ms-win-crt-process-l1-1-0.dll	2020/1/21 11:30	应用程序扩展	20 KB
api-ms-win-crt-runtime-l1-1-0.dll	2020/1/21 11:30	应用程序扩展	22 KB
api-ms-win-crt-studio-l1-1-0.dll	2020/1/21 11:30	应用程序扩展	25 KB
api-ms-win-crt-string-l1-1-0.dll	2020/1/21 11:30	应用程序扩展	25 KB
api-ms-win-crt-time-l1-1-0.dll	2020/1/21 11:30	应用程序扩展	21 KB
api-ms-win-crt-utility-l1-1-0.dll	2020/1/21 11:30	应用程序扩展	19 KB
blink_image_resources_200_percent.pak	2020/1/21 11:30	PAK 文件	27 KB
content_resources_200_percent.pak	2020/1/21 11:30	PAK 文件	1 KB
content_shell.pak	2020/1/21 11:30	PAK 文件	8,480 KB
d3dcompiler_47.dll	2020/1/21 11:30	应用程序扩展	4,077 KB
ffmpeg.dll	2020/1/21 11:30	应用程序扩展	1,910 KB
icudtl.dat	2020/1/21 11:30	DAT 文件	9,959 KB
libEGL.dll	2020/1/21 11:30	应用程序扩展	18 KB
libGLESv2.dll	2020/1/21 11:30	应用程序扩展	3,602 KB
LICENSE.electron.txt	2020/1/21 11:30	文本文档	2 KB
LICENSES.chromium.html	2020/1/21 11:30	QQBrowser HT...	1,770 KB
mitmdumpUrlSaver.exe	2020/1/21 11:30	应用程序	66,058 KB
msvcprt40.dll	2020/1/21 11:30	应用程序扩展	619 KB
natives_blob.bin	2020/1/21 11:30	BIN 文件	217 KB
node.dll	2020/1/21 11:30	应用程序扩展	17,445 KB
pdf_viewer_resources.pak	2020/1/21 11:30	PAK 文件	161 KB
snapshot_blob.bin	2020/1/21 11:30	BIN 文件	1,496 KB
ucrbase.dll	2020/1/21 11:30	应用程序扩展	978 KB
ui_resources_200_percent.pak	2020/1/21 11:30	PAK 文件	76 KB
vcruntime140.dll	2020/1/21 11:30	应用程序扩展	86 KB
views_resources_200_percent.pak	2020/1/21 11:30	PAK 文件	57 KB

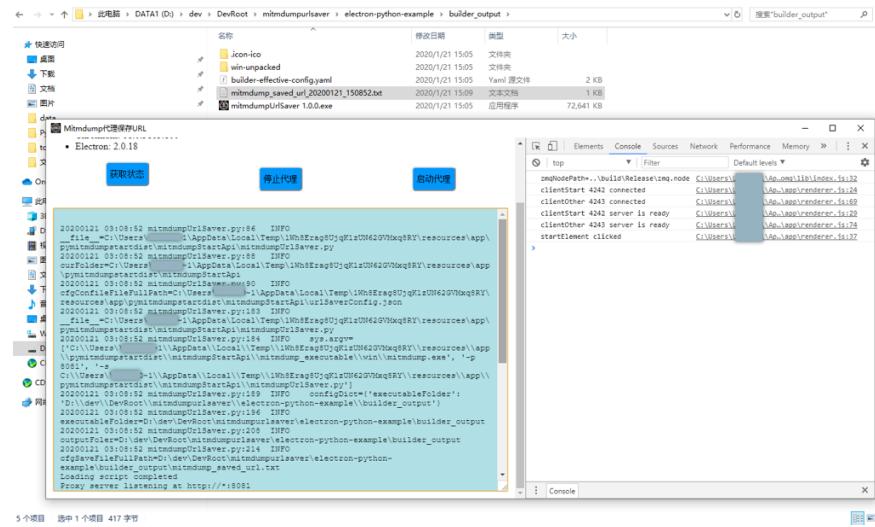
更加没法找到exe所在目录。

后来去试了试别人[提到](#)的相关参数：

```
process.env.PORTABLE_EXECUTABLE_DIR
```

最终是可以获取到当前exe所在目录的：

exe所在目录：



相关代码：

```
let isWin = process.platform === "win32"

if (isWin){
  let portableExecutableDir = process.env.PORTABLE_EXECUTABLE_DIR
  console.log("portableExecutableDir=%s", portableExecutableDir)
}
```

输出：

```
portableExecutableDir=D:\dev\DevRoot\mitmdumpurlsaver\electron-py
```

供参考。

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 17:17:36

Mac系统

此处整理关于Mac系统中开发Electron的相关心得。

npm安装electron速度太慢

mac中npm install很慢，则可以去全局设置源，换成淘宝的源：

```
npm config set registry https://registry.npm.taobao.org/
npm config set ELECTRON_MIRROR http://npm.taobao.org/mirrors/electron/
```

且单独给electron的变量，加到启动脚本中：

```
vi ~/.zshrc
```

最后加上：

```
export ELECTRON_MIRROR=http://npm.taobao.org/mirrors/electron/
```

保存退出后，去立刻生效：

```
source ~/.zshrc
```

再去用 echo 验证：

```
# echo $ELECTRON_MIRROR
http://npm.taobao.org/mirrors/electron/
```

即可极大地提高 npm install 去安装electron相关东西的速度。

比如此处是从 <100KB/s 变成 > 1MB/s。

获取不到process.env的值

经过之前折腾：

【未解决】 Mac中electron打包后process.env是undefined获取不到有效值

确定是：

Mac中，打包后的 app 在运行期间，无法获取到env变量：

```
process.env
```

或：

```
const remote = electron.remote;
remote.process.env
```

的值。-》也就无法获取到

```
process.env.NODE_ENV
```

或：

```
remote.process.env.NODE_ENV
```

的值了 -》 后续就没法通过别人说的：

```
const remote = electron.remote
let nodeEnv = remote.process.env.NODE_ENV

const isProd = remote.process.env.NODE_ENV === 'production'

let portableExecutableDir = remote.process.env.PORTABLE_EXECUTABLE_DIR

let initCwd = remote.process.env.INIT_CWD
```

去判断是否是生产环境，以及获取到有效的app可执行文件所在目录了。

获取Electron的app的根目录

此处是

文件： `electron-python-example/main.js`

```

const electron = require('electron')
const app = electron.app
console.log("app=%s", app)

console.log("main.js: __dirname=%s", __dirname)
console.log(" getAppPath=%s", app.getAppPath())
// let curAppPath = app.getAppPath()
let curAppPath = __dirname
console.log("curAppPath=%s", curAppPath)

let rootPath = curAppPath
// // for debug:
// let rootPath = "/Users/limao/Downloads/electron_app/mitm
console.log("rootPath=%s", rootPath)
let removeAppSuffixP = /\//[^\/]+\.app\Content\Resourc
rootPath = rootPath.replace(removeAppSuffixP, "")
console.log("rootPath=%s", rootPath)

```

通过调用：

```

const createSinglePyProc = (curScript, curPort, curDistFolder) {
    if (guessPackaged(curDistFolder)) {
        curPyProc = childProcess.execFile(curScript, [curPort,
            console.log('execFile curPyProc=%s, from curScript=%s',
        } else {
            curPyProc = childProcess.spawn('python', [curScript, curPort,
                console.log('spawn python curPyProc=%s, from curScript=%s',
            }

```

传入到Python端：

文件： electron-python-
example/pymitmdump/mitmdumpStartApi.py

```
class MitmdumpStartApi(object):
    def __init__(self, rootPath):
        self.appRootPath = rootPath
        logging.debug("self.appRootPath=%s", self.appRootPath)

    @zerorpc.stream
    def startProxy(self):
        """Start mitmdump proxy"""
        logging.debug("startProxy")
        from mitmdumpManage import startMitmdumpSaver
        startResp = startMitmdumpSaver(self.appRootPath)

    def main():
        rootPath = sys.argv[2]
        logging.debug("rootPath=%s", rootPath)
        zerorpcServer = zerorpc.Server(MitmdumpStartApi(rootPath))
        zerorpcServer.serve()
```

传递给：

文件： electron-python-example/pymitmdump/mitmdumpManage.py

```
def startMitmdumpSaver(appRootPath):
    """Start mitmdump saver"""
    logging.debug("startMitmdumpSaver")
    logging.debug("appRootPath=%s", appRootPath)
    # electron not package: /Users/limao/dev/xxx/crawler/mitmdump
    # electron packaged: /Users/limao/Downloads/electron_ap

    ...
    shellCmdList = ['%s' % gConfig["mitmdumpExecutable"],
```

在启动了mitmdump后，是通过：

文件： electron-python-example/pymitmdump/mitdumpUrlSaver.py

```

def getOutputFolder():
    print("sys.argv=%s" % sys.argv)
    argvLen = len(sys.argv)
    print("argvLen=%s" % argvLen)

    if argvLen >= 4:
        sysArgv3 = sys.argv[3]
        print("sysArgv3=%s" % sysArgv3)
        appRootPath = sysArgv3
        print("appRootPath=%s" % appRootPath)
        outputFoler = appRootPath
    else:
        outputFoler = os.path.dirname(__file__)

    return outputFoler

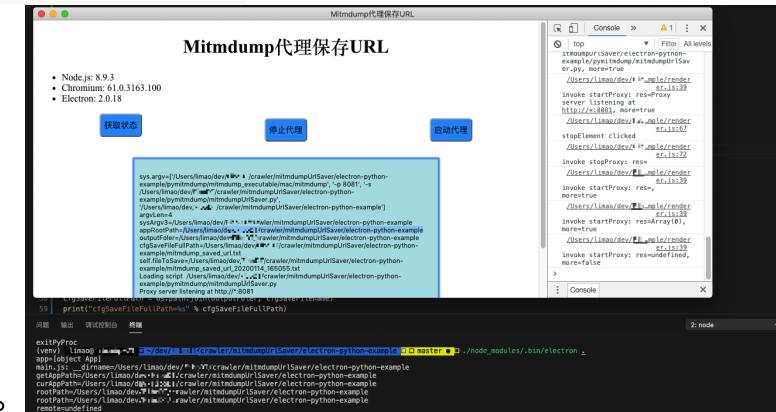
outputFoler = getOutputFolder()
print("outputFoler=%s" % outputFoler)

```

中的 `sys` 的 `argv[3]` , 最终得到了此处的app的root的path路径:

- 开发环境=本地调试环

境: /Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example



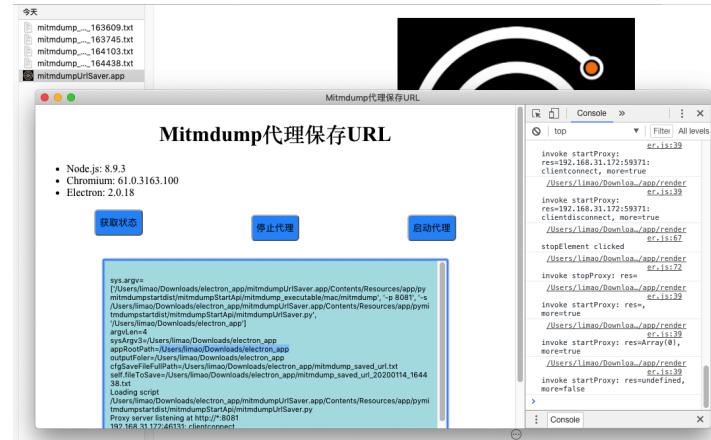
- 生产环境=打包发布app后:

- 当前app运行路

径: /Users/limao/Downloads/electron_app

- app的root路径: `app.getAppPath` =

/Users/limao/Downloads/electron_app/mitmdumpUrlSaver
.app/Contents/Resources/app



crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新: 2020-06-18 17:40:06

不同语言

Electron除了支持本身的最基本的 Web语言 = HTML+CSS+JS 之外，也可以通过其他方式支持其他编程语言和框架。

现整理如下：

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 10:25:29

Python

此处总结关于Electron支持Python期间遇到的经验。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-06-17 18:12:27

electron-python-example

在参考 `electron-python-example` 给Electron添加Python支持期间，遇到很多问题。整理如下：

npm install时zeromq会报错

- 解决办法：把node从 `13.5.0` 换成 `8.x` 的版本，比如此处的

```
brew install node@8
```

安装出来后的版本是：`8.17.0`

详见：

【已解决】mac中npm install报错：make
Release/obj.target/zmq/binding.o Error 1

能看到electron界面，但是输入计算表达式后，却看不到输出结果

- 原因：可能有多种
- 思路：通过electron的默认打开的console的log输出的错误信息中去找原因
 - 如果找不到合适错误原因，则可以尝试去给python的进程加上`stdout`和`stderr`

```
electron-python-example/main.js
```

```

const createPyProc = () => {
  let script = getScriptPath()
  let port = '' + selectPort()

  if (guessPackaged()) {
    pyProc = require('child_process').execFile(script, [port])
  } else {
    pyProc = require('child_process').spawn('python', [script])
  }

  if (pyProc != null) {
    console.log('child process success on port ' + port)

    pyProc.stdout.on('data', function(data){
      var textChunk = data.toString('utf8') // buffer to string
      console.log("textChunk=%s", textChunk)
    })

    pyProc.stderr.on('data', function(data){
      console.log("pyProc error=%s", data)
    });
  }
}

```

用于查看到输出的普通信息和错误信息，以便于找到错误现象和原因

详见：

【已解决】electron-python-example运行后输入计算内容但是不显示结果

Uncaught ReferenceError process is not defined

解决办法：加上参数配置 nodeIntegration: true

electron-python-example/main.js

```
const createWindow = () => {
  mainWindow = new BrowserWindow(
    {
      width: 800,
      height: 600,
      webPreferences: {
        nodeIntegration: true,
      }
    }
  )
}
```

详见：

【已解决】electron-python-example运行出错：Uncaught
ReferenceError process is not defined

renderer.js Error Lost remote after 10000ms

- 原因：python的zerorpc没有安装，或者安装了zerorpc但是python进程没有正常启动
- 解决办法：确保已安装zerorpc，且的确启动了对应进程

注：此处在虚拟环境virtualenv中

```
pip install zerorpc
```

然后确保

```
electron-python-example/main.js

pyProc = require('child_process').spawn('python', [script,
```

正常启动了。

此处有2种确认方式：

- 方式1：系统中查看是否有对应python进程

```
ps aux | grep 61737
```

其中的61737是python进程PID

是通过js中加了调试代码

```
console.log('spawn: pyProc=%s', pyProc)
```

后，启动

```
./node_modules/.bin/electron .
```

时，看到console的log中的

```
spawnfile: 'python',
_handle:
Process {
  owner: [Circular],
  onexit: { [Function] [length]: 2, [name]: '' },
  pid: 61737
```

有对应python的pid的。

如果有进程，说明python进程没问题，否则说明进程没正常启动。

- 方式2：加上 stdout、 stderr 等监听

```
if (pyProc != null) {
  //console.log(pyProc)
  console.log('child process success on port ' + port)

  pyProc.stdout.on('data', function(data){
    var textChunk = data.toString('utf8') // buffer to string
    console.log("textChunk=%s", textChunk)
  })

  pyProc.stderr.on('data', function(data){
    console.log("pyProc error=%s", data)
  });
}
```

确认没有error等异常错误信息

此处自己就是看到有

```
pyProc error=Traceback (most recent call last):
  File "/xxx/electron_python/electron-python-example/pycall.py", line 1, in <module>
    import zerorpc
ModuleNotFoundError: No module named 'zerorpc'
```

从而发现：

是没有进入virtualenv的venv中，导致没有找到zerorpc

该确保进入虚拟环境：

```
source venv/bin/activate
```

即可找到 `zerorpc`，正常启动python进程了。

zerorpc.exceptions.LostRemote: Lost remote after 10s heartbeat

- 问题：electron的python，用了stream后，运行了10秒后，报错：
 - js中：`invoke startSaver: error=Error: Lost remote after 10000ms`
 - python中：`zerorpc.exceptions.LostRemote: Lost remote after 10s heartbeat`
- 原因：js和python端，底层都有heartbeat心跳机制，每隔一段时间，都要检测是否服务还有效alive，否则就报错。

而此处我的情况特殊：

js端，通过zerorpc通讯到python端

但是python端是 `while True` 的无限循环，除非进程被干掉，否则不返回

所以肯定无法满足heartbeat的10秒之内返回的要求

- 解决办法：故意设置一个足够大的心跳时间

文件：`electron-python-example/renderer.js`

把：

```
let client = new zerorpc.Client()
```

改为：

```
const constLargeEnoughHeartbeat = 60 * 60 * 24 * 30 * 12
clientOptions = {
  "heartbeatInterval": constLargeEnoughHeartbeat,
}
let client = new zerorpc.Client(clientOptions)
```

以此规避此问题。

Uncaught Error The module zeromq zmq.node was compiled against a different Node.js version

- 问题: Electron运行

```
./node_modules/.bin/electron
```

报错:

```
Uncaught Error: The module node_modules/zeromq/build/Re  
was compiled against a different Node.js version using  
NODE_MODULE_VERSION 57. This version of Node.js require  
NODE_MODULE_VERSION 75. Please try re-compiling or re-i  
the module (for instance, using `npm rebuild` or `npm i
```

- 原因: 此处安装的zeromq的库是针对于

```
NODE_MODULE_VERSION 57 = Node.js Node.js 8.x
```

编译出来的, 而当前用的node版本是:

```
NODE_MODULE_VERSION 75 = Node.js Node.js 12.7.0
```

所以不匹配, 没法使用。

- 解决办法: 去重新编译出对应的匹配的 zeromq 库即可:
- 注意事项和准备工作:

需要说明的是, 此处当时node版本, 已经是为了解决

```
npm install
```

期间 zerorpc出错, 而去降低了node版本, 从之前的 node: 13.5.0 降低到了: node: 8.17.0

且对于node和electron本身, 不同版本之间是有依赖关系的, 所以此处 node 8.17.0 , 只能用 electron 2.0.18

所以此处配置是:

文件: electron_python/electron-python-example/package.json

```
"devDependencies": {  
  "electron": "^2.0.18",
```

文件: electron_python/electron-python-example/.npmrc

```
npm_config_target="2.0.18" # electron version
```

- 解决步骤

安装electron-rebuild:

```
npm install --save-dev electron-rebuild
```

重新去rebuild:

```
./node_modules/.bin/electron-rebuild
```

即可。

注:

编译之前，需要安装 cmake

```
brew install cmake
```

详见：

【已解决】Electron报错：Uncaught Error The module zeromq
zmq.node was compiled against a different Node.js version

优化：被调用python代码的print为输出日志到log文件

- 背景：希望被调用的python代码的print的日志输出改为输出到log日志文件中，便于后期调试查看log
- 解决办法：

文件： electron-python-
example/pymitmdump/mitmdumpSaverApi.py

```
from utils import loggingInit
logFilename = "mitmdumpSaverApi.log"
loggingInit(logFilename)

# from __future__ import print_function
from mitmdumpLauncher import startMitmdumpSaver, getMitmdur
import sys
import os
import zerorpc
import logging

"""

def main():
    logging.debug(__file__)

    addr = 'tcp://127.0.0.1:' + parse_port()
```

相关的logging的初始化：

文件： electron-python-example/pymitmdump/utils.py

```

#_*_ coding: utf-8 _*
# Function: Crifan's utils functions, copy from https://github.com/crifan/crifan-utils
# Update: 20200103
# Author: Crifan Li

import os
import sys
from datetime import datetime, timedelta
import logging
# print("sys.executable=%s" % sys.executable)
# sys.executable=/Users/limao/dev/xxx/crawler/mitmdumpUrls.py

...
CURRENT_LIB_FILENAME = "crifanLogging"

LOG_FORMAT_FILE = "%(asctime)s %(filename)s:%(lineno)-4d %(levelname)-8s"
LOG_LEVEL_FILE = logging.DEBUG
LOG_FORMAT_CONSOLE = "%(asctime)s %(filename)s:%(lineno)-4d %(levelname)-8s"
LOG_LEVEL_CONSOLE = logging.INFO

def loggingInit(filename = None,
                fileLogLevel = LOG_LEVEL_FILE,
                fileLogFormat = LOG_FORMAT_FILE,
                fileLogDateFormat = '%Y/%m/%d %I:%M:%S',
                enableConsole = True,
                consoleLogLevel = LOG_LEVEL_CONSOLE,
                consoleLogFormat = LOG_FORMAT_CONSOLE,
                consoleLogDateFormat = '%Y-%m-%d %I:%M:%S',
                ):
    """
    init logging for both log to file and console

    :param logFilename: input log file name
                        if not passed, use current lib filename
    :return: none
    """

    logFilename = ""
    if filename:
        logFilename = filename
    else:
        # logFilename = __file__ + ".log"
        # '/Users/crifan/dev/dev_root/xxx/crifanLogging.py'
        logFilename = CURRENT_LIB_FILENAME + ".log"

    # logging.basicConfig(
    #     level      = fileLogLevel,
    #     format    = fileLogFormat,
    #     datefmt  = fileLogDateFormat,
    #     filename = logFilename,

```

```

#           encoding = "utf-8",
#           filemode = 'w')

# rootLogger = logging.getLogger()
rootLogger = logging.getLogger("mitmproxy")
rootLogger.setLevel(fileLogLevel)
fileHandler = logging.FileHandler(
    filename=logFilename,
    mode='w',
    encoding="utf-8")
fileHandler.setLevel(fileLogLevel)
fileFormatter = logging.Formatter(
    fmt=fileLogFormat,
    datefmt=fileLogDateFormat
)
fileHandler.setFormatter(fileFormatter)
rootLogger.addHandler(fileHandler)

if enableConsole :
    # define a Handler which writes INFO messages or higher to the console
    console = logging.StreamHandler()
    console.setLevel(consoleLogLevel)
    # set a format which is simpler for console use
    formatter = logging.Formatter(
        fmt=consoleLogFormat,
        datefmt=consoleLogDateFormat)
    # tell the handler to use this format
    console.setFormatter(formatter)
    rootLogger.addHandler(console)
    . . .

```

被调用的python

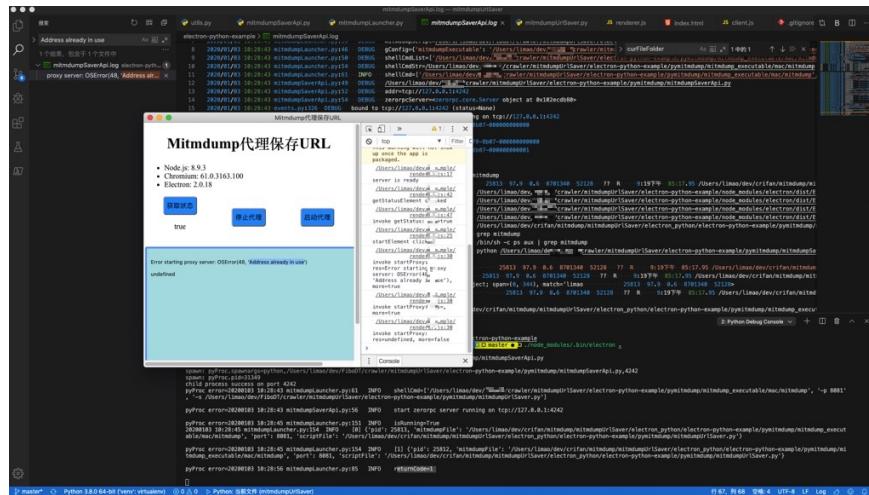
文件: electron-python-example/pymitmdump/mitmdumpLauncher.py

```

. . .
import subprocess
import os
import re
import logging
from utils import osIsMacOS, osIsWindows
. . .
curFileFolder = os.path.dirname(__file__)
logging.debug("curFileFolder=%s", curFileFolder) # curFileFolder
. . .

```

即可正常输出log日志到文件中了:



获取Mac中electron-builder打包出的app通过双击启动时app文件所在的文件夹路径

- 背景：Mac中带Python的Electron打包后是app文件，双击app运行，希望在运行期间获取app所在目录

文件： electron-python-example/pymitmdump/mitmdumpStartApi.py

```

import os
import sys
import logging
from utils import getFilenameNoPointSuffix, loggingInit

sysMeipass = None
sysArgv0 = sys.argv[0]
hasMeipass = hasattr(sys, '_MEIPASS')
isBundled = hasMeipass
if isBundled:
    # /xxx/pymitmdumpstartdist/mitmdumpStartApi/mitmdumpSta
    logFilenameNoSuffix = sysArgv0.split(os.path.sep)[-1]
    logFilename = "%s.log" % logFilenameNoSuffix
    sysMeipass = sys._MEIPASS
    logPath = sysMeipass
else:
    logFilename = "%s.log" % getFilenameNoPointSuffix(__file__)
    logPath = os.path.dirname(__file__)
logFullPath = os.path.join(logPath, logFilename)
# logFullPath = "/Users/limao/Downloads/electron_app/mitmd
loggingInit(logFullPath)

logging.debug("sysArgv0=%s", sysArgv0)
logging.debug("hasMeipass=%s", hasMeipass)
logging.debug("sysMeipass=%s", sysMeipass)

logging.debug("logFilename=%s", logFilename)
logging.debug("logPath=%s", logPath)
logging.debug("logFullPath=%s", logFullPath)

sysExecutable = sys.executable
logging.debug("sysExecutable=%s", sysExecutable)
sysArgv = sys.argv
logging.debug("sysArgv=%s", sysArgv)
hasFrozen = hasattr(sys, 'frozen')
logging.debug("hasFrozen=%s", hasFrozen)
if hasFrozen:
    sysForzen = sys.frozen
    logging.debug("sysForzen=%s", sysForzen)

logging.debug("os.getcwd=%s", os.getcwd())
logging.debug("os.curdir=%s", os.curdir)
logging.debug("__file__=%s", __file__)
logging.debug("__name__=%s", __name__)

```

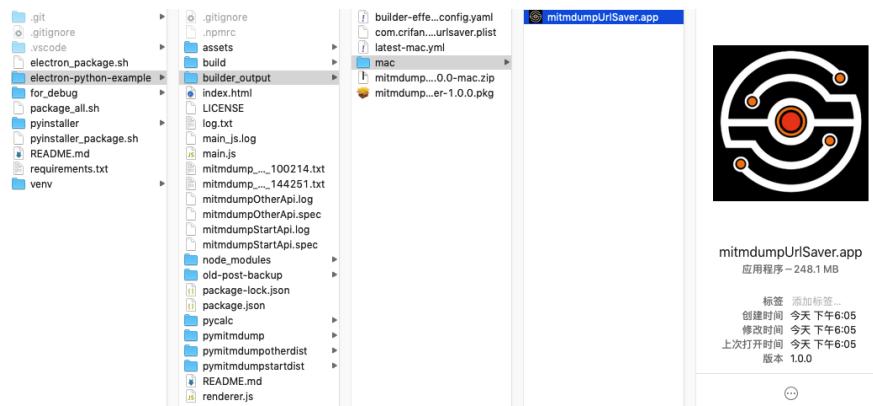
可以获取到对应路径：

- 打包后：bundle的=是二进制 的路径

(PyInstaller)打包后的（且后来再用electron-builder打包在一起的，所生成的）app文件，双击运行后对应输出是：

```
2020/01/13 06:05:51 mitmdumpStartApi.py:23 DEBUG sysArgv0=
2020/01/13 06:05:51 mitmdumpStartApi.py:24 DEBUG hasMeipass
2020/01/13 06:05:51 mitmdumpStartApi.py:25 DEBUG sysMeipass
2020/01/13 06:05:51 mitmdumpStartApi.py:27 DEBUG logFilenar
2020/01/13 06:05:51 mitmdumpStartApi.py:28 DEBUG logPath=/l
2020/01/13 06:05:51 mitmdumpStartApi.py:29 DEBUG logFullPat
2020/01/13 06:05:51 mitmdumpStartApi.py:32 DEBUG sysExecute
2020/01/13 06:05:51 mitmdumpStartApi.py:34 DEBUG sysArgv=[]
2020/01/13 06:05:51 mitmdumpStartApi.py:36 DEBUG hasFrozen_
2020/01/13 06:05:51 mitmdumpStartApi.py:39 DEBUG sysForzen_
2020/01/13 06:05:51 mitmdumpStartApi.py:41 DEBUG os.getcwd_
2020/01/13 06:05:51 mitmdumpStartApi.py:42 DEBUG os.curdir_
2020/01/13 06:05:51 mitmdumpStartApi.py:43 DEBUG __file__=_
2020/01/13 06:05:51 mitmdumpStartApi.py:44 DEBUG __name__=_
2020/01/13 06:05:51 mitmdumpStartApi.py:101 DEBUG mitmdumpS
.
.
```

其中此处app本身所在位置是：



输出的路径是：

- `sysArgv0` :
 - `/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/elect
 ron-python-
 example/builder_output/mac/mitmdumpUrlSaver.app/Cont
 ents/Resources/app/pymitdumpstartdist/mitmdumpStart
 Api/mitmdumpStartApi`
- `logPath` :
 - `/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/elect
 ron-python-
 example/builder_output/mac/mitmdumpUrlSaver.app/Cont
 ents/Resources/app/pymitdumpstartdist/mitmdumpStart
 Api`

- 其实也就是 sysMeipass 的值
- 没打包之前：是源码=是xxx.py=此处是 mitmdumpStartApi.py 的路径

注意：此处先要删除2个dist目录（pymitmdumpstartdist和 pymitmdumpotherdist）

运行

```
./node_modules/.bin/electron
```

输出：

```
2020/01/13 06:13:55 mitmdumpStartApi.py:23 DEBUG sysArgv0=,
2020/01/13 06:13:55 mitmdumpStartApi.py:24 DEBUG hasMeipass=False
2020/01/13 06:13:55 mitmdumpStartApi.py:25 DEBUG sysMeipass=None
2020/01/13 06:13:55 mitmdumpStartApi.py:27 DEBUG logFilename=None
2020/01/13 06:13:55 mitmdumpStartApi.py:28 DEBUG logPath=/Users/limao/.pyenv/versions/3.8.0/lib/python3.8
2020/01/13 06:13:55 mitmdumpStartApi.py:29 DEBUG logFullPath=None
2020/01/13 06:13:55 mitmdumpStartApi.py:32 DEBUG sysExecutable=None
2020/01/13 06:13:55 mitmdumpStartApi.py:34 DEBUG sysArgv=[],
2020/01/13 06:13:55 mitmdumpStartApi.py:36 DEBUG hasFrozen=False
2020/01/13 06:13:55 mitmdumpStartApi.py:41 DEBUG os.getcwd()
2020/01/13 06:13:55 mitmdumpStartApi.py:42 DEBUG os.curdir=None
2020/01/13 06:13:55 mitmdumpStartApi.py:43 DEBUG __file__=,
2020/01/13 06:13:55 mitmdumpStartApi.py:44 DEBUG __name__=
2020/01/13 06:13:55 mitmdumpStartApi.py:101 DEBUG /Users/limao/.pyenv/versions/3.8.0/lib/python3.8
...
.
```

可见： sys 没有 frozen，也没有 _MEIPASS 值。

打包electron-python的单个主文件报错： OSError Python library not found libpython3.8m.dylib

- 问题：mac中用PyInstaller打包python报错 OSError Python library not found libpython3.8m.dylib
- 原因：此处PyInstaller打包需要动态库，类似于 libpython3.8m.dylib，而默认安装的python是静态库 /Users/limao/.pyenv/versions/3.8.0/lib/libpython3.8.a
- 解决办法：重新安装带动态库的python 3.8
- 具体步骤：加上参数重新编译：

最好先去删除之前旧的python：

```
pyenv uninstall 3.8.0  
rm -rf /Users/limao/.pyenv/versions/3.8.0/
```

然后再去重新安装：

```
env PYTHON_CONFIGURE_OPTS="--enable-framework" pyenv install
```

即可正常使用 PyInstaller。

其他说明：

(1) 此处安装后对应的动态库是

```
limao@xxx ~ ll /Users/limao/.pyenv/versions/3.8.0/Python.framework/Versions/3.8/lib/python3.8/lib-dynload/_ctypes.so  
total 0  
lrwxr-xr-x 1 limao CORP\Domain Users 9B 1 6 10:34 ,
```

软链接对应的文件是：3.4MB的 Python

```
~ ll /Users/limao/.pyenv/versions/3.8.0/Python.framework/Versions/3.8/lib/python3.8/lib-dynload/_ctypes.so  
-rwxr-xr-x 1 limao CORP\Domain Users 3.4M 1 6 10:34 ,
```

(2) 此处安装后，还需要安装项目所依赖的一些库

比如：

```
pip install virtualenv  
  
virtualenv venv  
source venv/bin/activate  
  
pip install zerorpc
```

zerorpc的stream持续输出无返回导致后续函数stopProxy无法调用

- 问题：mitmdumpSaverApi.py中调用启动代理后，代理没有输出。
- 原因：由于启动代理，内部逻辑是，没有立刻返回，而是通过stream持续返回输出，导致进程被占用，所以stopProxy都没有被执行到
- 解决办法：新增一个进程，额外单独的去处理关闭代理（以及获取代理状态）
- 具体步骤：

js中createPyProc创建了2个线程：

文件: `electron-python-example/main.js`

```

const electron = require('electron')
const app = electron.app
const BrowserWindow = electron.BrowserWindow
const path = require('path')

// *****
// * py process
// *****

// const PY_DIST_FOLDER = 'pycalcdist'
const PY_DIST_FOLDER = 'pymitmdumpdist'

// const PY_FOLDER = 'pycalc'
// const PY_MODULE = 'api' // without .py suffix

const PY_FOLDER = 'pymitmdump'
// const PY_MODULE = 'mitmdumpSaverApi' // without .py suffix
const PY_MODULE_START = 'mitmdumpStartApi' // without .py suffix
const PY_MODULE_OTHER = 'mitmdumpOtherApi' // without .py suffix

PORT_START = 4242
PORT_OTHER = 4243

// let pyProc = null
let pyProcStart = null
let pyProcOther = null
// let pyPort = null

const guessPackaged = () => {
    // console.log("guessPackaged")
    const fullPath = path.join(__dirname, PY_DIST_FOLDER)
    // console.log("fullPath=%s", fullPath)
    isPackaged = require('fs').existsSync(fullPath)
    // console.log("isPackaged=%s", isPackaged)
    return isPackaged
}

// const getScriptPath = () => {
//     if (!guessPackaged()) {
//         return path.join(__dirname, PY_FOLDER, PY_MODULE +
//     }
//     if (process.platform === 'win32') {
//         return path.join(__dirname, PY_DIST_FOLDER, PY_MODULE +
//     }
//     return path.join(__dirname, PY_DIST_FOLDER, PY_MODULE,
// }

const getScriptPath = (pyModule) => {
    if (!guessPackaged()) {
        return path.join(__dirname, PY_FOLDER, pyModule + '.py')
}

```

```

    }

    if (process.platform === 'win32') {
        return path.join(__dirname, PY_DIST_FOLDER, pyModule, py)
    }
    return path.join(__dirname, PY_DIST_FOLDER, pyModule, py)
}

const getScriptPathStart = () => {
    return getScriptPath(PY_MODULE_START)
}

const getScriptPathOther = () => {
    return getScriptPath(PY_MODULE_OTHER)
}

// const selectPort = () => {
//     pyPort = 4242
//     return pyPort
// }

const createSinglePyProc = (curScript, curPort) => {
    console.log('createSinglePyProc: curScript=%s, curPort=%o',
    let curPyProc = null

    if (guessPackaged()) {
        // curPyProc = require('child_process').execFile(curScript,
        curPyProc = require('child_process').execFile(curScript,
    } else {
        // curPyProc = require('child_process').spawn('python',
        curPyProc = require('child_process').spawn('python', [
    }

    console.log('curPyProc=%s', curPyProc)

    if (curPyProc != null) {
        console.log("success create child process")
        // console.log('curPyProc=%o', curPyProc)
        console.log('curPyProc.spawnargs=%s', curPyProc.spawnargs)
        console.log('curPyProc.pid=%s', curPyProc.pid)

        curPyProc.stdout.on('data', function(chunk){
            var chunkUtf8Str = chunk.toString('utf8') // buffer to string
            console.log("curPyProc stdout data: chunkUtf8Str=%s",
        })

        curPyProc.stderr.on('data', function(chunk){
            console.error("curPyProc stderr data: chunk=%s\r\n",
        })

        curPyProc.on('close', function(closeCode){
            console.log("curPyProc close: closeCode=%s\r\n", closeCode)
        })
    }
}

```

```

        })
        // for Windows:
        curPyProc.on('exit', function(exitCode){
            console.log("curPyProc exit: exitCode=%s\r\n", exitCode)
        })
    }

    return curPyProc
}

const createPyProc = () => {
    console.log('In createPyProc')
    // let script = getScriptPath()
    // console.log('script=%s', script)
    // let port = '' + selectPort()
    // console.log('port=%s', port)

    let scriptStart = getScriptPathStart()
    console.log('scriptStart=%s', scriptStart)
    pyProcStart = createSinglePyProc(scriptStart, PORT_START)

    let scriptOther = getScriptPathOther()
    console.log('scriptOther=%s', scriptOther)
    pyProcOther = createSinglePyProc(scriptOther, PORT_OTHER)
}

const exitPyProc = () => {
    // pyProc.kill()
    // pyProc = null
    console.log('exitPyProc')

    if (pyProcStart != null) {
        pyProcStart.kill()
        pyProcStart = null
    }

    if (pyProcOther != null) {
        pyProcOther.kill()
        pyProcOther = null
    }

    // pyPort = null
}

app.on('ready', createPyProc)
app.on('will-quit', exitPyProc)

...

```

文件: electron-python-example/renderer.js

```

const zerorpc = require("zerorpc")

// let client = new zerorpc.Client()

PORT_START = 4242
PORT_OTHER = 4243

/***** Client Start Server *****/
const constLargeEnoughHeartbeat = 60 * 60 * 24 * 30 * 12 // 12 days
clientOptions = {
    "heartbeatInterval": constLargeEnoughHeartbeat,
}
let clientStart = new zerorpc.Client(clientOptions)

clientStart.connect(`tcp://127.0.0.1:${PORT_START}`)
console.log(`clientStart ${PORT_START} connected`)
clientStart.invoke("echo", "server ready", (error, res) =>
    if(error || res !== 'server ready') {
        console.error(error)
    } else {
        console.log(`clientStart %d server is ready`, PORT_START)
    }
)

let startElement = document.querySelector('#startProxy')
let outputElement = document.querySelector('#output')

startElement.addEventListener('click', () => {
    console.log("startElement clicked")
    outputElement.textContent = "" // clear before start

    clientStart.invoke("startProxy", (error, res, more) => {
        if(error) {
            console.error("invoke startProxy: error=%s, more=%s", error, more)
        } else {
            console.log("invoke startProxy: res=%s, more=%s", res, more)
            if (res != undefined) {
                outputElement.textContent += "\n" + res
                // outputElement.textContent += res
            }
            outputElement.scrollTop = outputElement.scrollHeight
        }
    })
})

/***** Client Other Server *****/

```

```

let clientOther = new zerorpc.Client()

clientOther.connect(`tcp://127.0.0.1:${PORT_OTHER}`)
console.log(`clientOther ${PORT_OTHER} connected`)
clientOther.invoke("echo", "server ready", (error, res) =>
  if(error || res !== 'server ready') {
    console.error(error)
  } else {
    console.log("clientOther %d server is ready", PORT_OTHER)
  }
)

let stopElement = document.querySelector('#stopProxy')

stopElement.addEventListener('click', () => {
  console.log("stopElement clicked")
  clientOther.invoke("stopProxy", (error, res) => {
    if(error) {
      console.error("invoke stopProxy: error=%s", error)
    } else {
      console.log("invoke stopProxy: res=%s", res)
    }
  })
})

let getStatusElement = document.querySelector('#getStatus')
let statusElement = document.querySelector('#serverStatus')

getStatusElement.addEventListener('click', () => {
  console.log("getStatusElement clicked")
  clientOther.invoke("getStatus", (error, res) => {
    if(error) {
      console.error("invoke getStatus: error=%s", error)
    } else {
      console.log("invoke getStatus: res=%s", res)
      statusElement.textContent = res
    }
  })
})

```

分别调用的python脚本文件是：

文件： electron-python-example/pymitmdump/mitmdumpStartApi.py

```

from utils import getFilenameNoPointSuffix, loggingInit
loggingInit("%s.log" % getFilenameNoPointSuffix(__file__))

from mitmdumpManage import startMitmdumpSaver
import sys
import zerorpc
import logging

class MitmdumpStartApi(object):
    def echo(self, text):
        """Echo any text from zerorpc client"""
        logging.debug("text=%s", text)
        return text

    @zerorpc.stream
    def startProxy(self):
        """Start mitmdump proxy"""
        logging.debug("startProxy")
        startResp = startMitmdumpSaver()
        logging.debug("startResp=%s", startResp)
        return startResp

    def parse_port():
        port = 4242
        try:
            port = int(sys.argv[1])
        except Exception as e:
            pass
        curPortStr = '{}'.format(port)
        return curPortStr

    def main():
        logging.debug(__file__)

        addr = 'tcp://127.0.0.1:' + parse_port()
        logging.debug("addr=%s", addr)
        zerorpcServer = zerorpc.Server(MitmdumpStartApi())
        logging.debug("zerorpcServer=%s", zerorpcServer)
        zerorpcServer.bind(addr)
        logging.info('start zerorpc server running on %s', addr)
        zerorpcServer.run()

    if __name__ == '__main__':
        main()

```

和：

文件： electron-python-
example/pymitmdump/mitmdumpOtherApi.py

```

from utils import getFilenameNoPointSuffix, loggingInit
loggingInit("%s.log" % getFilenameNoPointSuffix(__file__))

from mitmdumpManage import getMitmdumpStatus, stopMitmdump
import sys
import zerorpc
import logging

class MitmdumpOtherApi(object):
    def echo(self, text):
        """Echo any text from zerorpc client"""
        logging.debug("text=%s", text)
        return text

    def getStatus(self):
        """Get mitmdump server status"""
        logging.debug("getStatus")
        isRunning, processList = getMitmdumpStatus()
        logging.debug("isRunning=%s, processList=%s", isRunning, processList)
        return isRunning
        # return isRunning, processList

    def stopProxy(self):
        """Stop mitmdump proxy"""
        logging.debug("stopProxy")
        stopResp = stopMitmdump()
        logging.debug("stopResp=%s", stopResp)
        return stopResp

    def parse_port():
        port = 4242
        try:
            port = int(sys.argv[1])
        except Exception as e:
            pass
        curPortStr = '{}'.format(port)
        return curPortStr

    def main():
        logging.debug(__file__)

        addr = 'tcp://127.0.0.1:' + parse_port()
        logging.debug("addr=%s", addr)
        zerorpcServer = zerorpc.Server(MitmdumpOtherApi())
        logging.debug("zerorpcServer=%s", zerorpcServer)
        zerorpcServer.bind(addr)
        logging.info('start zerorpc server running on %s', addr)
        zerorpcServer.run()

```

```
if __name__ == '__main__':
    main()
```

其中都调用了最底层实现逻辑的代码：

文件： `electron-python-example/pymitmdump/mitmdumpManage.py`

```

#_*_ coding: utf-8 _*
# Function: Launch mitmdump to start script service to proxy
# Update: 20191231
# Author: Crifan Li

import subprocess
import os
import re
import logging
from utils import osIsMacOS, osIsWindows

def startMitmdumpSaver():
    """Start mitmdump saver"""
    logging.debug("startMitmdumpSaver")

    isUseShell = False
    # isUseShell = True

    curFileFolder = os.path.dirname(__file__)
    logging.debug("curFileFolder=%s", curFileFolder) # curl

    Mitmdump_Mac = "mitmdump_executable/mac/mitmdump"
    Mitmdump_Win = "mitmdump_executable/win/mitmdump.exe"
    mitmdumpExecutablePath = None
    if osIsMacOS():
        mitmdumpExecutablePath = Mitmdump_Mac
    elif osIsWindows():
        mitmdumpExecutablePath = Mitmdump_Win
    logging.debug("mitmdumpExecutablePath=%s", mitmdumpExecutablePath)

    mitmdumpExecutableFullPath = os.path.join(curFileFolder, mitmdumpExecutablePath)
    logging.debug("mitmdumpExecutableFullPath=%s", mitmdumpExecutableFullPath)
    # mitmdumpExecutable = "mitmdump"
    # mitmdumpExecutable = "pyinstaller/mitmdump_executable"
    mitmdumpExecutable = mitmdumpExecutableFullPath
    logging.debug("mitmdumpExecutable=%s", mitmdumpExecutable)

    mitmdumpScriptFilename = "mitmdumpUrlSaver.py"
    logging.debug("mitmdumpScriptFilename=%s", mitmdumpScriptFilename)
    mitmdumpScriptFullPath = os.path.join(curFileFolder, mitmdumpScriptFilename)
    logging.debug("mitmdumpScriptFullPath=%s", mitmdumpScriptFullPath)
    mitmdumpScript = mitmdumpScriptFullPath
    logging.debug("mitmdumpScript=%s", mitmdumpScript)

    gConfig = {
        "mitmdumpExecutable": mitmdumpExecutable,
        "port": 8081,
        "mitmdumpScript": mitmdumpScript,
    }
    logging.debug("gConfig=%s", gConfig)

```

```

# shellCmdList = ['ping', '-c 4', 'www.crifan.com']
shellCmdList = ['%s' % gConfig["mitmdumpExecutable"],
logging.debug("shellCmdList=%s", shellCmdList)
# shellCmdList=['/Users/limao/dev/crifan/mitmdump/mitmdump']

shellCmdStr = " ".join(shellCmdList)
logging.debug("shellCmdStr=%s", shellCmdStr)
# shellCmdStr=/Users/limao/dev/crifan/mitmdump/mitmdump

if isUseShell:
    shellCmd = shellCmdStr
else:
    shellCmd = shellCmdList
logging.info("shellCmd=%s", shellCmd)

curProcess = subprocess.Popen(
    shellCmd,
    stdout=subprocess.PIPE,
    stderr=subprocess.STDOUT,
    universal_newlines=True,
    shell=isUseShell,
)
logging.debug("curProcess=%s", curProcess)

while True:
    curLineOutput = curProcess.stdout.readline()
    curLineOutput = curLineOutput.strip()
    logging.debug("curLineOutput=%s", curLineOutput)
    yield curLineOutput

# Do something else
returnCode = curProcess.poll()
if returnCode is not None:
    logging.info('returnCode=%s', returnCode)

    # Process has finished, read rest of the output
    # for eachLineOutput in curProcess.stdout.read():
    #     eachLineOutput = eachLineOutput.strip()
    #     logging.debug("eachLineOutput=%s", eachLineOutput)
    #     yield eachLineOutput
    respLineList = curProcess.stdout.readlines()
    logging.info('respLineList=%s', respLineList)
    yield respLineList

break

def getMitmdumpStatus():
    """Get current mitmdump status"""
    logging.debug("getMitmdumpStatus")

```

```

isRunning, processInfoList = False, []
if osIsMacOS():
    shellCmdStr = "ps aux | grep mitmdump"
elif osIsWindows():
    shellCmdStr = "tasklist | findstr mitmdump"
logging.debug("shellCmdStr=%s", shellCmdStr)
shellRespStr = subprocess.check_output(
    shellCmdStr,
    shell=True,
    universal_newlines=True,
)
logging.debug("shellRespStr=%s", shellRespStr)
singleLineList = shellRespStr.split(os.linesep)
logging.debug("singleLineList=%s", singleLineList)
for eachLineStr in singleLineList:
    # TODO: add windows support extract mitmdump from cmd output
    logging.debug("eachLineStr=%s", eachLineStr)
    # mitmdumpCmdPattern = "mitmdump\s+-p\s+\d+-s\s+\$"
    # mitmdumpCmdPattern = "mitmdump\s+-p\s+\d+\$+-\$"
    # mitmdumpCmdPattern = "mitmdump\s+-p\s+(?P<port>\d+)-s"
    # mitmdumpCmdPattern = "\s+(?P<mitmdumpFile>\S+m:\d+)-s"
    # mitmdumpCmdPattern = "\w+\s+(?P<pidStr>\d+)\.+"
    mitmdumpCmdPattern = "\w+\s+(?P<pidStr>\d+)\.+"
    foundMitmdump = re.match(mitmdumpCmdPattern, eachLineStr)
    foundMitmdump = re.search(mitmdumpCmdPattern, eachLineStr)
    logging.debug("foundMitmdump=%s", foundMitmdump)
    if foundMitmdump:
        isRunning = True
        matchedMitmdumpStr = foundMitmdump.group(0)
        logging.debug("matchedMitmdumpStr=%s", matchedMitmdumpStr)
        pidStr = foundMitmdump.group("pidStr")
        pidInt = int(pidStr)
        logging.debug("pidInt=%s", pidInt)
        mitmdumpFile = foundMitmdump.group("mitmdumpFile")
        logging.debug("mitmdumpFile=%s", mitmdumpFile)
        portStr = foundMitmdump.group("portStr")
        portInt = int(portStr)
        logging.debug("portInt=%s", portInt)
        scriptFile = foundMitmdump.group("scriptFile")
        logging.debug("scriptFile=%s", scriptFile)
        curProcessDict = {
            "pid": pidInt,
            "mitmdumpFile": mitmdumpFile,
            "port": portInt,
            "scriptFile": scriptFile,
        }
        logging.debug("curProcessDict=%s", curProcessDict)
        processInfoList.append(curProcessDict)

# logging.info("isRunning=%s, processInfoList=%s", isRunning, processInfoList)

```

```

logging.info("isRunning=%s", isRunning)
if processInfoList:
    for curIdx, eachProcess in enumerate(processInfoList):
        logging.info("[%d] %s", curIdx, eachProcess)
    return isRunning, processInfoList

def stopMitmdump():
    """Stop mitmdump"""
    # TODO: add windows support
    shellCmdStr = "killall mitmdump"
    logging.debug("shellCmdStr=%s", shellCmdStr)
    try:
        shellRespStr = subprocess.check_output(
            shellCmdStr,
            shell=True,
            universal_newlines=True,
        )
        logging.info("shellCmdStr=%s -> shellRespStr=%s",
    except subprocess.CalledProcessError as procErr:
        logging.error("shellCmdStr=%s -> procErr=%s", shellCmdStr, procErr)
        shellRespStr = None

    return shellRespStr

if __name__ == "__main__":
    from utils import loggingInit
    logFilename = "mitmdumpManager.log"
    loggingInit(logFilename)

    # getMitmdumpStatus()
    stopMitmdump()

```

以及，其中start部分是调用的是：

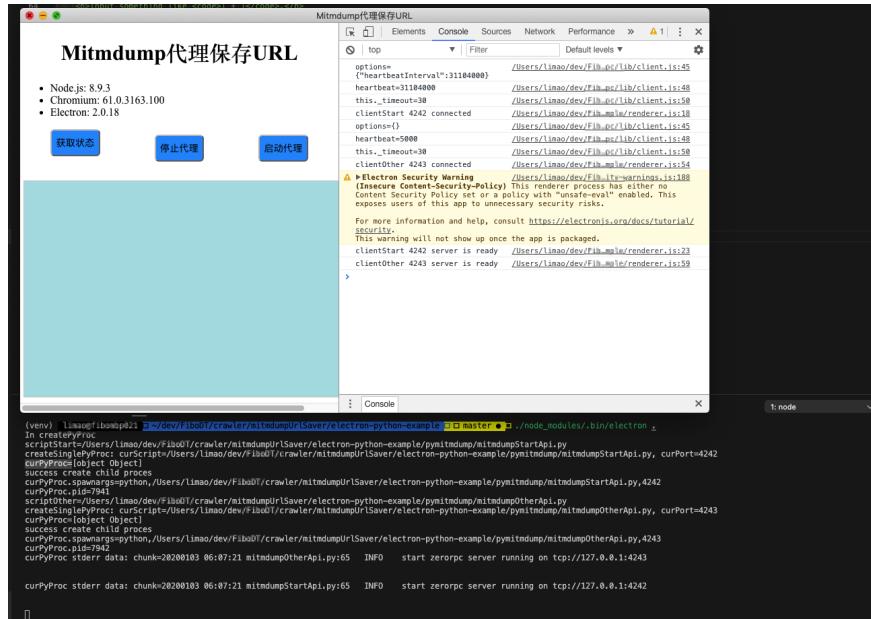
文件： electron-python-example/pymitmdump/mitmdumpUrlsaver.py

```

class Saver:
    def request(self, flow):
        ...
addons = [Saver()]

```

正常启动后：



输出：

```
□ ./node_modules/.bin/electron
In createPyProc
scriptStart=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example
createSinglePyProc: curScript=/Users/limao/dev/xxx/crawler, curPyProc=[object Object]
success create child proces
curPyProc.spawnargs=python,/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pynitdump/mitmdumpStartApi.py, curPort=4242
curPyProc.pid=7941
scriptOther=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pynitdump/mitmdumpOtherApi.py, curPort=4243
createSinglePyProc: curScript=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pynitdump/mitmdumpOtherApi.py, curPyProc=[object Object]
success create child proces
curPyProc.spawnargs=python,/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pynitdump/mitmdumpOtherApi.py, curPort=4243
curPyProc.pid=7942
curPyProc stderr data: chunk=20200103 06:07:21 mitmdumpOtherApi.py:65  INFO start zeropc server running on tcp://127.0.0.1:4243
curPyProc stderr data: chunk=20200103 06:07:21 mitmdumpStartApi.py:65  INFO start zeropc server running on tcp://127.0.0.1:4242
```

以及两个log文件：

文件： electron-python-example/mitmdumpOtherApi.log

```

2020/01/03 06:07:21 mitmdumpOtherApi.py:58 DEBUG /Users/
2020/01/03 06:07:21 mitmdumpOtherApi.py:61 DEBUG addr=1
2020/01/03 06:07:21 mitmdumpOtherApi.py:63 DEBUG zerorpc
2020/01/03 06:07:21 events.py:326 DEBUG bound to tcp://127.0.0.1:8080
2020/01/03 06:07:21 mitmdumpOtherApi.py:65 INFO start
2020/01/03 06:07:21 channel.py:125 DEBUG <-- new channel
2020/01/03 06:07:21 mitmdumpOtherApi.py:30 DEBUG text=<
2020/01/03 06:07:21 channel.py:139 DEBUG -x- closed char

```

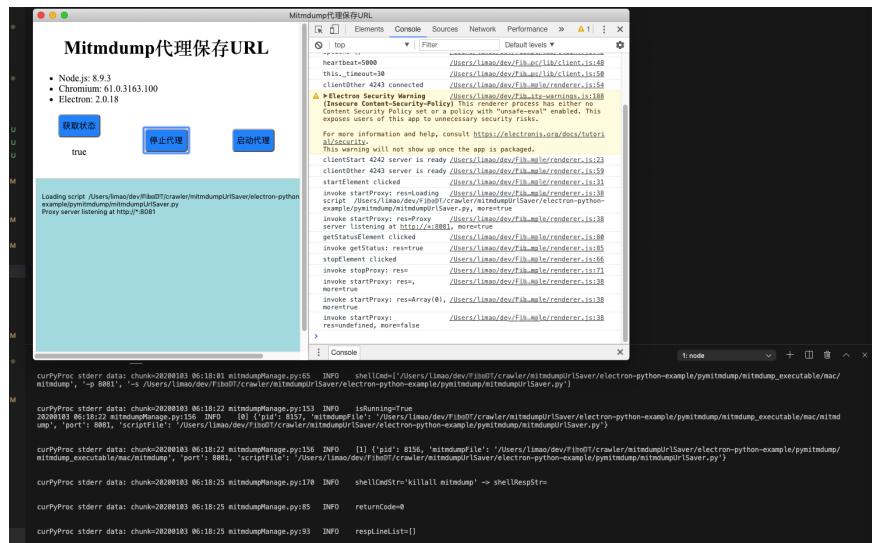
文件： electron-python-example/mitmdumpStartApi.log

```

2020/01/03 06:07:21 mitmdumpStartApi.py:58 DEBUG /Users/
2020/01/03 06:07:21 mitmdumpStartApi.py:61 DEBUG addr=1
2020/01/03 06:07:21 mitmdumpStartApi.py:63 DEBUG zerorpc
2020/01/03 06:07:21 events.py:326 DEBUG bound to tcp://127.0.0.1:8080
2020/01/03 06:07:21 mitmdumpStartApi.py:65 INFO start
2020/01/03 06:07:21 channel.py:125 DEBUG <-- new channel
2020/01/03 06:07:21 mitmdumpStartApi.py:16 DEBUG text=<
2020/01/03 06:07:21 channel.py:139 DEBUG -x- closed char

```

后续启动mitmdump后，再去get status或stop，可以正常调用了：



就不会被之前只有单一进程而阻塞和卡死了。

优化：js端通过zerorpc调用python代码的逻辑优化为支持返回连续输出结果

- 背景：Electron的js端通过zerorpc调用python代码，之前只能返回单次结果，现象希望支持返回连续的（mitmdump抓包）输出结果
- 解决办法：

(1) python中

文件: electron-python-example/pymitmdump/mitmdumpLauncher.py

```
def startMitmdumpSaver():
    """Start mitmdump saver"""
    print("mitmdumpLauncher: startMitmdumpSaver")

    curProcess = subprocess.Popen(
        shellCmd,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        universal_newlines=True,
        shell=isUseShell,
    )
    print("curProcess=%s" % curProcess)

    while True:
        curLineOutput = curProcess.stdout.readline()
        curLineOutput = curLineOutput.strip()
        # print("curLineOutput=%s" % curLineOutput)
        # print(curLineOutput)
        yield curLineOutput

        # Do something else
        returnCode = curProcess.poll()
        if returnCode is not None:
            print('returnCode=%s' % returnCode)

        # Process has finished, read rest of the output
        for eachLineOutput in curProcess.stdout.readlines():
            eachLineOutput = eachLineOutput.strip()
            print("eachLineOutput=%s" % eachLineOutput)

            yield eachLineOutput

    break
```

核心逻辑: 通过yield输出连续内容

对应调用:

文件: electron-python-example/pymitmdump/mitmdumpSaverApi.py

```

from mitmdumpLauncher import startMitmdumpSaver
import zerorpc

class MitmdumpSaverApi(object):
    @zerorpc.stream
    def startSaver(self):
        """Start mitmdump saver to save url to file"""
        print("MitmdumpSaverApi startSaver")

        respValue = startMitmdumpSaver()
        print("respValue=%s" % respValue)
        return respValue

```

(2) js中

文件: electron-python-example/renderer.js

```

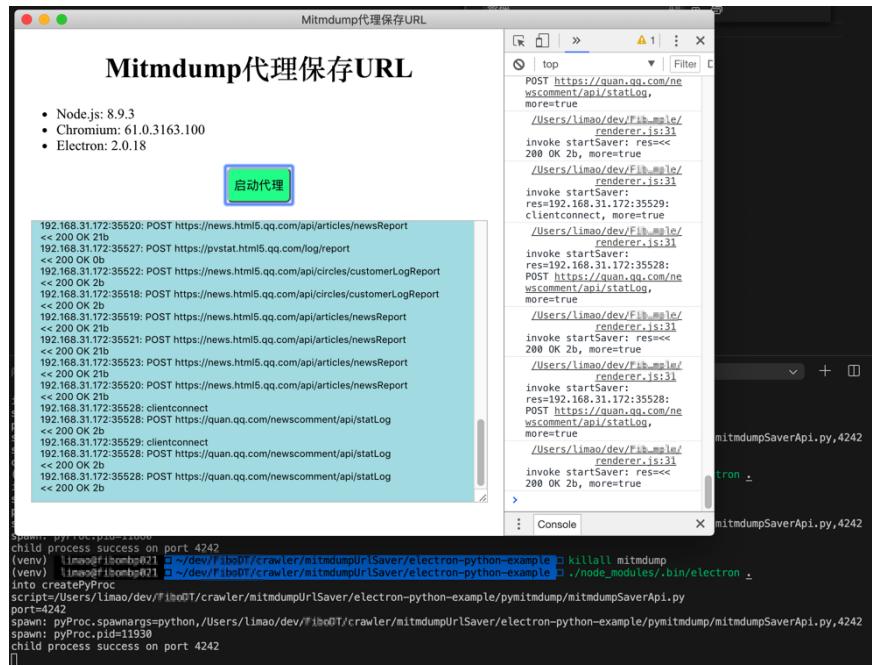
const zerorpc = require("zerorpc")
// let client = new zerorpc.Client()
const constLargeEnoughHeartbeat = 60 * 60 * 24 * 30 * 12 //,
clientOptions = {
    "heartbeatInterval": constLargeEnoughHeartbeat,
}
let client = new zerorpc.Client(clientOptions)

let startElement = document.querySelector('#startSaver')
let outputElement = document.querySelector('#output')

startElement.addEventListener('click', () => {
    console.log("startElement clicked")
    client.invoke("startSaver", (error, res, more) => {
        if(error) {
            console.error("invoke startSaver: error=%s, more=%s",
        } else {
            console.log("invoke startSaver: res=%s, more=%s", res
            outputElement.textContent += "\n" + res
            // outputElement.textContent += res
            outputElement.scrollTop = outputElement.scrollHeight
        }
    })
})

```

效果: 持续的输出mitmdump代理抓包到的url链接了



Electron打包双击app启动时python用正则 处理从sys的argv中解析出app文件所在根目 录路径

- 背景：对于支持Python的Electron打包后的app，希望在app启动运行期间，用Python从sys的argv中解析出app文件所在根目录路径
- 解决办法：

文件： electron-python-
example/pymitmdump/mitmdumpUrlsaver.py

```

def extractAppFolderFromArgv(curArgvList):
    """Extract app folder from argv list

    Examples:
        input: ['/Users/limao/Downloads/electron_app/mitmproxy.py']
        output: /Users/limao/Downloads/electron_app
    .....
    appFolder = None

    eachArgvStr = " ".join(curArgvList)
    foundScriptPy = re.search("-s\s+(?P<scriptPy>.+?\.\py)"]
    if foundScriptPy:
        curScriptPy = foundScriptPy.group("scriptPy")

        print("curScriptPy=%s" % curScriptPy)
        # TODO: add windows exe support
        macAppP = "[^\\/]+\\.app"
        # macAppP = "[^\\]+\\.app"
        # macAppP = "\\.app"
        # macAppP = "\.app"
        # macAppP = ".app"
        # if re.match(macAppP, curScriptPy, re.I):
        # if re.match(macAppP, curScriptPy):
        # isMacApp = re.match(macAppP, curScriptPy)
        foundMacApp = re.search(macAppP, curScriptPy)
        if foundMacApp:
            removeMacAppP = "/?[^/]+\\.app(.+)?$"
            # appPath = re.sub(removeMacAppP, "", curScriptPy)
            appFolder = re.sub(removeMacAppP, "", curScriptPy)
            print("appFolder=%s" % appFolder)
        else:
            print("not mac app: %s" % curScriptPy)

    return appFolder

appFolder = extractAppFolderFromArgv(sys.argv)
print("appFolder=%s" % appFolder)
if appFolder:
    outputFoler = appFolder
else:
    outputFoler = os.path.dirname(__file__)
print("outputFoler=%s" % outputFoler)

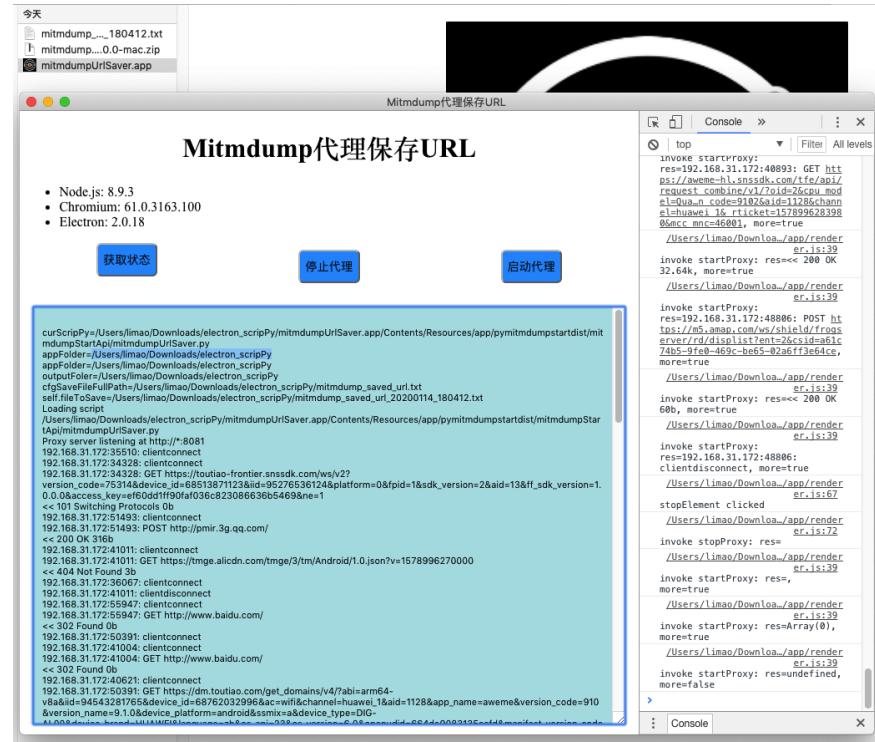
```

打包后双击app运行，即可得到根目录：

```

curScripPy=/Users/limao/Downloads/electron_scripPy/mitmdump
appFolder=/Users/limao/Downloads/electron_scripPy
appFolder=/Users/limao/Downloads/electron_scripPy
outputFoler=/Users/limao/Downloads/electron_scripPy
cfgSaveFilePath=/Users/limao/Downloads/electron_scripPy
self.fileToSave=/Users/limao/Downloads/electron_scripPy/mit
Loading script /Users/limao/Downloads/electron_scripPy/mit
Proxy server listening at http://*:8081

```



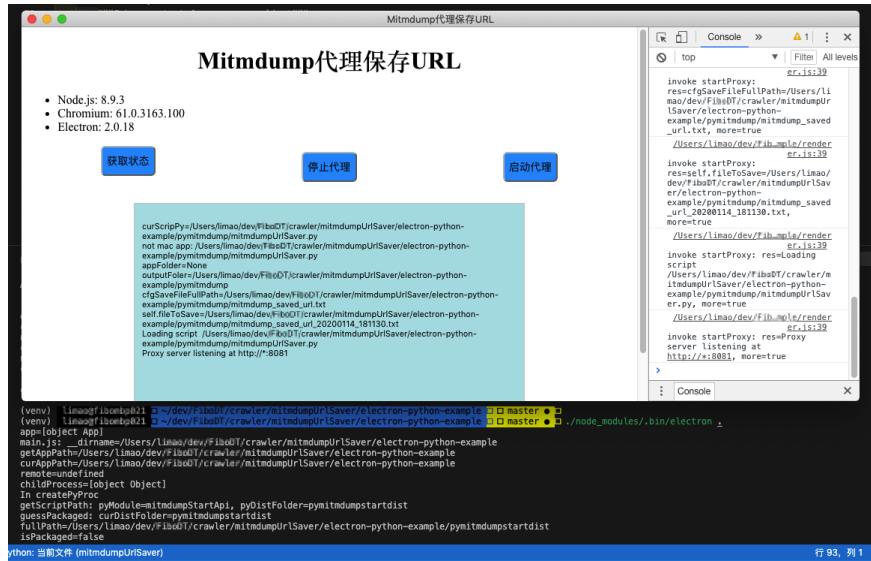
2种情况对应的输出:

(1) debug=调试时

```

curScripPy=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/e
not mac app: /Users/limao/dev/xxx/crawler/mitmdumpUrlSaver,
appFolder=None
outputFoler=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/
cfgSaveFilePath=/Users/limao/dev/xxx/crawler/mitmdumpUrlSav
self.fileToSave=/Users/limao/dev/xxx/crawler/mitmdumpUrlSav
Loading script /Users/limao/dev/xxx/crawler/mitmdumpUrlSav
Proxy server listening at http://*:8081

```



(2) 普通命令行中调用mitmdump时

```

* limao@xxx ~ /dev/xxx/electron-python-example/pymitmdump
__file__=mitmdumpUrlSaver.py
curScripPy=mitmdumpUrlSaver.py
not mac app: mitmdumpUrlSaver.py
appFolder=None
outputFoler=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pymitmdumpUrlSaver.py
cfgSaveFileFullPath=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pymitmdump/mitmdump_saved_url.txt
self.filePathToSave=/Users/limao/dev/xxx/crawler/mitmdumpUrlSaver/electron-python-example/pymitmdump/mitmdump_saved_url_20200114_181130.txt
Loading script mitmdumpUrlSaver.py
Proxy server listening at http://*:8081

```

js

Electron集成js框架

想要把主流的 js 框架集成到Electron中，有如下选择：

- react
 - [electron-react-boilerplate/electron-react-boilerplate: A Foundation for Scalable Cross-Platform Apps](#)
- vue
 - [SimulatedGREG/electron-vue: An Electron & Vue.js quick start boilerplate with vue-cli scaffolding, common Vue plugins, electron-packager/electron-builder, unit/e2e testing, vue-devtools, and webpack](#)

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新： 2020-06-18 10:29:19

Node

Node，全称 Node.js，是一个 js 的服务端框架。

而Electron是基于：

- 后端： Node
- 前端： Chromium

所以Electron开发期间会涉及到一些和Node有关的内容，整理如下。

crifan.com，使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 10:53:25

Electron和Node版本对应关系

开发Electron期间，往往会涉及到，需要特定的版本的Node。

此处整理出Electron和Node版本对应关系：

- 常见版本对应关系
 - Electron 2.0.18 -> Node 8.17.0
 - Electron 3.0.0 -> Node 10.2.0
- 完全列表
 - 详见官网
 - [electron-releases - npm](#)

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 10:28:44

NODE_MODULE_VERSION和node版本对应关系

折腾Electron期间，往往遇到类似于

【未解决】Electron报错：Uncaught Error The module zeromq
zmq.node was compiled against a different Node.js version

的问题，就会涉及到：NODE_MODULE_VERSION 和 node 之间的版本对应关系。

此处总结如下：

	NodeJS版本	说明
NODE_MODULE_VERSION 79	Node.js 13.x	Node.js 13.0.0 ~ Node.js 13.5.0
NODE_MODULE_VERSION 72	Node.js 12.x	Node.js 12.0.0 ~ Node.js 12.14.0
NODE_MODULE_VERSION 75	(据说是) Node.js 12.7.0	
NODE_MODULE_VERSION 67	Node.js 11.x	Node.js 11.0.0 ~ Node.js 11.15.0
NODE_MODULE_VERSION 64	Node.js 10.x	Node.js 10.0.0 ~ Node.js 10.18.0
NODE_MODULE_VERSION 59	Node.js 9.x	Node.js 9.0.0 ~ Node.js 9.11.2
NODE_MODULE_VERSION 57	Node.js 8.x	Node.js 8.0.0 ~ Node.js 8.17.0
NODE_MODULE_VERSION 51	Node.js 7.x	Node.js 7.0.0 ~ Node.js 7.10.1
NODE_MODULE_VERSION 48	Node.js 6.x	Node.js 6.0.0 ~ Node.js 6.17.1
NODE_MODULE_VERSION 47	Node.js 5.x	Node.js 5.0.0 ~ Node.js 5.12.0
NODE_MODULE_VERSION 46	Node.js 4.x	Node.js 4.0.0 ~ Node.js 4.9.1

更多细节详见：

- [以往的版本 | Node.js](#)
- [Node v12.7.0 \(Current\) | Node.js](#)

附录

下面列出相关参考资料。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新: 2020-05-29 16:34:53

Electron文档和教程

此处整理Electron的相关资料和教程：

Electron官网有非常详尽的教程，需要查找某些技术细节时，推荐优先参考官网教程：

- 官网文档主入口
 - 文档 | Electron
 - 截图预览



■ 指南：开始使用 Electron

配置开发环境

- macOS 开发环境配置
- Windows 开发环境配置
- Linux 开发环境配置

选择一个编辑器

- 创建你的第一个应用
- 安装 Electron
- 开发一个简单的 Electron
- 启动你的应用
- 模板和命令行界面
- electron-forge
- electron-builder
- electron-react-boilerplate
- 其它工具和模板

应用架构

- 主进程和渲染器进程
- 使用 Electron 的 API
- 使用 Node.js 的 API
- 使用原生 Node.js 模块
- 性能策略

为你的应用添加功能

- 通知
- 最近的文件
- 应用程序进程
- 自定义 Dock 菜单
- 自定义 Windows 任务栏
- 自定义 Linux 桌面动作
- 键盘快捷键
- 网络在线侦测
- 针对 macOS 系统 BrowserWindows 的展示文件
- 原生文件拖放 API
- 黑屏溢出
- 支持 macOS 深色模式
- 语言
- 辅助功能
- Spectron
- Devtron
- 启用辅助功能

测试和调试

- 测试主进程
- Déboguer le Main Process avec Visual Studio Code
- 使用 Selenium 和 WebDriver
- 使用自动化持续集成系统 (CI) 进行测试 (Travis, Jenkins)
- 开发者工具扩展
- 使用自定义驱动程序进行自动化测试
- 分发
- 支持平台
- 代码签名
- Mac App Store
- Windows Store
- Snapcraft

安全

- 报告安全问题
- Chromium 安全问题和升级
- Electron 安全警告
- 安全性检查列表
- 更新
- 部署更新服务器

■ 详细信息

安装 Electron

- 代理
- 自定义图像和缓存
- 故障排查

Electron发布 & 开发者反馈

- 版本规则
- 发布时间线
- 应用反馈项目

用asar打包App源代码

- 生成asar档案
- 使用asar档案文件
- 局限性
- 添加未打包的文件到asar档案

源码(Widewin)CDN

使用Pepper Flash插件

术语表

简介

进程对象

支持的命令行开关

环境变量

Chrome Extensions Support

重要的API变更

自定义DOM元素

CustomScheme 对象

DesktopCapturerSource 对象

Display对象

事件对象扩展通用事件

扩展信息对象

FileFilter 对象

FilePathHeaders 对象

GPU特性和状态对象

GPU特性和状态对象

InputEvent Object 输入事件

IOCounters 对象

IpcMainEvent Object extends Event

IpcRendererEvent对象继承Event

JumpListCategory 对象

JumpListItem 对象

KeyboardEvent 对象继承 Event

KeyboardInputEvent 对象继承 InputEvent

MemoryInfo 对象

MemoryUsageDetails 对象

MimeTypeBuffer 对象

MouseInputEvent 对象扩展自 InputEvent

NotificationAction 对象

Updating an Appveyor Azure Image

Build Instructions

构建步骤(Linux)

构建步骤(macOS)

构建步骤(Windows)

构建系统概述

Chromium 开发

在 C++ 代码中使用 clang-format 工具

编码风格

在 Windows 中调试

在 macOS 中调试

使用 XCode 调试

Technical Differences Between Electron and NW.js

Goma

Electron中的问题

Patches in Electron

合并请求

对 Electron 进行开发

在调试器中设置 Symbol 服务器

源代码目录结构

测试

升级 Node

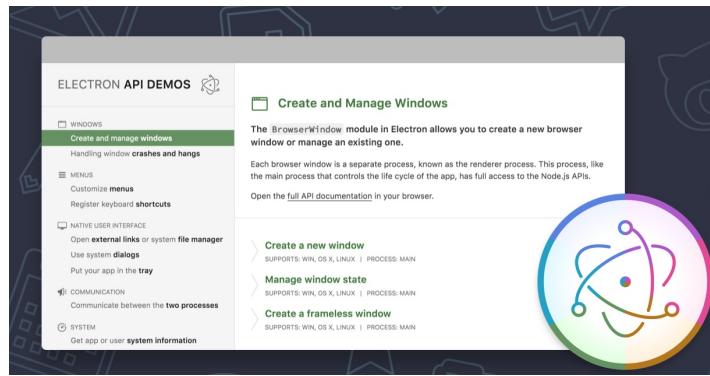
V8 开发

■ API 参考

API 结构

■ 高级

- 所有文档 单页面 版本
 - 所有的 Electron 的文档 | Electron
- 快速了解Electron所有API及效果
 - [electron/electron-api-demos: Explore the Electron APIs](#)



- 快速上手模板
 - [electron/electron-quick-start: Clone to try a simple Electron app](#)

Python打包

PyInstaller

如果涉及到Electron支持Python，则Electron打包期间也会涉及到Python的打包。

最常用的Python打包工具是：[PyInstaller](#)

其打包常用参数可以参考：

[What to bundle, where to search](#)

和运行时的路径有关的资料：

[Run-time Information — PyInstaller 3.6 documentation](#)

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 15:01:29

参考资料

- 【已解决】npm的electron的最新版本是多少
- 【已解决】electron-python中去打包Electron成独立的可执行程序
- 【已解决】electron打包npm run dist报错：signing Error Command failed codesign sign
- 【已解决】electron-builder打包报错：make-dir.js 86 catch SyntaxError Unexpected token {
- 【已解决】给Electron-Python打包出win的exe
- 【已解决】windows中用PyInstaller打包mitmdump的Python脚本为exe
- 【已解决】Windows10中重建NodeJS开发环境以运行和打包Electron-Python的项目为exe
- 【已解决】Windows中如何获取到Electron-builder打包后exe双击运行时的目录
- 【已解决】Electron的打包工具选择：electron-packager vs electron-builder vs electron-rebuild
- 【未解决】electron-builder打包Mac期间报错：skipped macOS application code signing reason cannot find valid Developer ID Application identity
- 【已解决】windows中electron-builder打包出免安装的可移动的单个exe
- 【已解决】js中用正则处理替换Electron打包后app的路径
- 【已解决】mac中用Image2icon制作要打包的app的图标
- 【无需解决】Mac中electron-rebuild失败：zeromq vendor napi.h error unknown type name napi_callback_scope
- 【已解决】Electron报错：Uncaught Error The module zeromq zmq.node was compiled against a different Node.js version
- 【已解决】electron-python-example运行后输入计算内容但是不显示结果
- 【已解决】mac中重新安装以降低node和npm版本
- 【已解决】electron和node版本对应关系
- 【未解决】Mac中electron打包后process.env是undefined获取不到有效值
- 【已解决】mac中npm安装electron速度太慢
- 【已解决】electron中js启动python调用mitmdump报错：mitmdump No such script
- 【已解决】electron-python中去打包Electron成独立的可执行程序
- 【记录】把Electron打包的app放到别的mac中测试
- 【已解决】打包后Electron路径中包含中文会无法启动mitmdump代理

- 【已解决】 Mac中electron-python用electron-builder打包后app运行失败： Error Lost remote after 10000ms
- 【已解决】 Mac中electron-builder打包的app运行Contents/MacOS下二进制没问题但是双击app运行却报错
- 【已解决】 Mac中electron-builder打包后app运行报错：
PermissionError Errno 13 Permission denied txt
- 【已解决】 mitmdump被PyInstaller和electron-builder打包后功能不正常无法抓报到有效url链接地址
- 【已解决】 mac中如何获取electron的app的根目录
- 【未解决】 Electron打包双击app启动时python用正则处理从sys的argv中解析出app文件所在根目录路径
- 【已解决】 electron中如何复制textarea中的内容
- 【已解决】 Windows中写bat脚本方便清理内容和编译运行PyInstaller 打包和Electron打包
- 【已解决】 windows中一个bat脚本文件如何包含另一个bat脚本文件且即时报错也继续运行
- 【已解决】 electron-python-example运行出错： Uncaught
ReferenceError process is not defined
- 【未解决】 mac中给Electron打包Windows的exe
- 【未解决】 mac中用wine运行electron-builder打包的win的exe
- 【已解决】 mac中wine中安装Windows的python 3
- 【已解决】 Mac中运行跑通electron-python-example的示例
- 【已解决】 Electron的打包工具选择： electron-packager vs electron-builder vs electron-rebuild
- 【已解决】 mac中安装node和npm
- 【已解决】 mac中npm install报错： make
Release/obj.target/zmq/binding.o Error 1
- 【已解决】 mac中electron-python用PyInstaller去打包可执行文件
- 【已解决】 Electron-python中优化被调用python代码的print为输出日志到log文件
- 【未解决】 js中child_process启动Python子进程无法获取到stderr的出错详细信息
- 【已解决】 electron-python中zerorpc的stream持续输出无返回导致后续函数stopProxy无法调用
- 【已解决】 Mac中把Electron的Python代码换成启动mitmdump的 Python脚本
- 【已解决】 electron通过js启动python但是实际上python代码print没输出后台mitmdump服务也没运行
- 【已解决】 把electron的js端通过zerorpc调用python代码的逻辑优化为支持返回连续输出结果
- 【已解决】 把electron的python中优化逻辑： 增加检测和关闭 mitmdump
- 【未解决】 Mac中用Electron打包集成Python代码为GUI独立程序
- 【未解决】 electron-python中去打包Electron成独立的可执行程序

- 【已解决】electron-builder打包配置中main.js的asar的asarUnpack不起作用
- 【已解决】electron的python的stream输出报错:
zerorpc.exceptions.LostRemote Lost remote after 10s heartbeat
- 【已解决】Electron-python中优化被调用python代码的print为输出日志到log文件
- 【已解决】Mac中打包后的electron中如何让main.js中的console.log输出到Electron的devTool中的console的log中
- 【已解决】】Mac中electron-builder打包app运行报错: cant open file app/Contents/Resources app.asar Errno 20 Not a directory
- 【已解决】Python中如何获取Mac中electron-builder打包出的app通过双击启动时app文件所在的文件夹路径
- 【已解决】electron中实现html的文本区域界面用于显示js中获取到的字符串
- 【已解决】mac打包Electron出app运行报错: Error loading Python lib /private/var/folders .Python dlopen image not found
- 【已解决】electron-builder打包时如何设置参数把把额外的文件夹添加和打包进来
- 【已解决】mac中PyInstaller打包electron-python的单个主文件报错: OSError Python library not found libpython3.8m.dylib
- 【未解决】mac中PyInstaller打包python报错: TypeError an integer is required got type bytes
- 【已解决】Electron中main.js即js输出log日志文件到当前文件夹
- 【已解决】windows中启动Electron-python报错: Uncaught Error A dynamic link library DLL initialization routine failed zerorpc zeromq zmq.node
- 【无需解决】win10中调试运行本地electron期间如何清除缓存让最新代码生效
- 【已解决】Mac中electron-builder打包后app运行报错:
PermissionError Errno 13 Permission denied txt
- 【已解决】mac中如何获取electron的app的根目录
- 【已解决】windows中打包Electron的mitmdumpUrlSaver为exe但运行没有在当前目录下生成txt文件
- 【已解决】electron通过js启动python但是实际上python代码print没输出后台mitmdump服务也没运行
- 【已解决】mac中Python的subprocess.Popen启动mitmdump报错:
No such script
- 【已解决】electron-builder打包时如何设置参数把把额外的文件夹添加和打包进来
- 【已解决】electron-builder打包期间实现拷贝整个目录和其中二进制文件到打包后的特定目录
- 【已解决】打包后Electron路径中包含中文会无法启动mitmdump代理

- 【已解决】Electron打包双击app启动时python用正则处理从sys的argv中解析出app文件所在根目录路径
- 【已解决】PyInstaller打包electron-python后运行报错：路径问题和subprocess.Popen依赖调用python文件没被打包进来的问题
- 【已解决】mac中PyInstaller打包后的二进制文件在electron-builder打包后app中有些无法通过child_process的execFile运行
- 【已解决】NODE_MODULE_VERSION和node版本号对应关系
- 【已解决】node中是否有根据当前系统类型不同而不同的路径分隔符
- 【已解决】js中child_process的spawn启动python脚本无效实际上没运行
-
- [开发环境 | Electron](#)
- [小公司小项目开发跨平台的桌面应用用什么编程语言比较好？ - 知乎](#)
- [在 Visual Studio 2019 - .NET 框架中创建您的第一个 WPF 应用 | Microsoft Docs](#)
- [C# WPF应用程序基础开发入门 | 小奋斗](#)
- [This free brand-new digital design app wants to take on Adobe and Sketch - News - Digital Arts](#)
- [\[译\] 零基础 macOS 应用开发（一） - 个人文章 - SegmentFault 思否](#)
- [Mac开发——基础篇（swift桌面App） swift刘成利的博客-CSDN博客](#)
- [Xcode - Apple Developer](#)
- [macOS - Apple Developer](#)
- [Electron 和当下其他的桌面开发方法相比如何？ - 知乎](#)
- [打造你的第一个 Electron 应用 | Electron](#)
- [Electron 应用架构 | Electron](#)
- [electron/electron-api-demos: Explore the Electron APIs](#)
- [Electron - 维基百科，自由的百科全书](#)
- [python - How to configure ZeroRPC and timeouts - Stack Overflow](#)
- [Path | Node.js v13.6.0 Documentation](#)
-

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved,
powered by Gitbook最后更新：2020-06-18 17:18:34