银河麒麟桌面操作系统 V10

electron 应用开发者指南

麒麟软件有限公司

2022 年 3 月

版本说明

版本号	编写人	日期	变更内容
V1.0	吴兆惠	2022-03-24	首次创建
V1.1	吴兆惠	2022-04-07	增加 mips 架构下使用 electron-builder 工具打包的方法

目录

1. 目的

本文主要演示 electron-quick-start 从依赖安装到打包成符合银河麒麟桌面操作系统 V10 软件商店上架规范的 deb 包的方法；由于各个架构中依赖包存在差异，所以各个架构单独进行演示编写，请参考对应架构进行打包。

本文中只是介绍了众多打包方式中比较常见的一种，仅供参考使用。

Electron 是一个使用 JavaScript、HTML 和 CSS 构建跨平台桌面应用程序的框架。它通过使用 Node.js 和 Chromium 的渲染引擎完成跨平台的桌面 GUI 应用程序的开发。Electron 兼容 Mac、Windows 和 Linux，可以构建出三个平台的应用程序。

npm 是 JavaScript 世界的包管理工具，并且是 Node.js 平台的默认包管理工具。通过 npm 可以安装、共享、分发代码，管理项目依赖关系。npm 是随同 Nodejs 一起安装的包管理工具，我们经常使用它来下载第三方包到本地。但在使用 npm 过程很多人估计都知道，在国内下载第三方包的速度极其之慢。国内推荐使用淘宝 npm 镜像，它是一个完整 npmjs.org 镜像，可以用此代替官方版本，同步频率目前为 10 分钟一次以保证尽量与官方服务同步。

注：

npm 官方网站：https://www.npmjs.com/

npm 官方源：https://registry.npmmirror.com/binary.html

npm 淘宝源：https://registry.npmmirror.com/binary.html

2. X86_64 架构

2.1. 安装开发基础环境

2.1.1. 安装 npm 和 node(如果已经安装请忽略)

a. V10 系统安装 npm 和 node

V10 版本默认没有预装 npm 和 node，需要从源里安装，但源里的 node(4.2.6)和 npm（3.5.2）版本过低，使用 npm install 安装高版本依赖包的时候可能会存在报错，可以通过 npm 在全局安装 n 模块，然后使用 n 模块来安装高版本的 node，此处以安装 10.19.0 版本为例。具体安装方法如下：

```
$ sudo apt update
$ sudo apt install npm -y
$ sudo npm install -g n
$ sudo n 10.19.0
   installing : node-v10.19.0
            mkdir : /usr/local/n/versions/node/10.19.0
            fetch : https://nodejs.org/dist/v10.19.0/node-v10.19.0-linux-x64.tar.xz
     installed : v10.19.0 (with npm 6.13.4)
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-v10-mips-2107:~$ node -v
v10.19.0
kylin@kylin-v10-mips-2107:~$ npm -v
6.13.4
```

b. V10(SP1)系统安装 npm 和 node

V10(SP1)默认没有预装 npm 和 node，需要从源里安装，源里的 node 版本为 v10.19.0，npm 版本为 v6.14.4。具体安装方法如下：

```
$ sudo apt update
$ sudo apt install npm -y
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-PC:~$ node -v
v10.19.0
kylin@kylin-PC:~$ npm -v
6.14.4
```

如果 V10(SP1)需要其他版本的 node 和 npm，也可以参考 V10 系统中使用 n 模块管理 node 的方式来安装指定版本的 node 和 npm。

2.1.2. 从源中安装 git(如果已经安装请忽略)

```
$ sudo apt install git -y
```

2.1.3. 设置 npm 源（可选）

如果当前网络不能有效的安装 electron，建议将 npm 的镜像源地址修改为国内淘宝源

```
$ npm config set registry https://registry.npm.taobao.org
```

2.2. 项目开发

此处以 electron-quick-start 项目使用 electron-v10.1.5 开发为例：

2.2.1. 下载 electron-quick-start 项目

```
$ git clone https://github.com/electron/electron-quick-start
$ cd electron-quick-start
$ git checkout remotes/origin/10-x-y
```

2.2.2. 安装依赖包

a. 在线安装 electron 及其依赖包

由于 package.json 文件中仅写了 electron 作为开发依赖，所以此处主要是进行 electron 包及其依赖包的安装。具体安装方法如下：

```
$ npm install
> core-js@3.6.5 postinstall /home/kylin/electron-quick-start/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"



> electron@10.1.5 postinstall /home/kylin/electron-quick-start/node_modules/electron
> node install.js

Downloading electron-v10.1.5-linux-x64.zip: [==================================] 100% ETA: 0.0 seconds
added 87 packages from 98 contributors and audited 87 packages in 89.761s

6 packages are looking for funding
  run `npm fund` for details

found 6 vulnerabilities (3 moderate, 3 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

b. 离线安装 electron 及其依赖包

如果在线安装 electron 失败，可以采用离线安装的方式来安装 electron。具体安装方法如下：

① 下载 electron 离线包

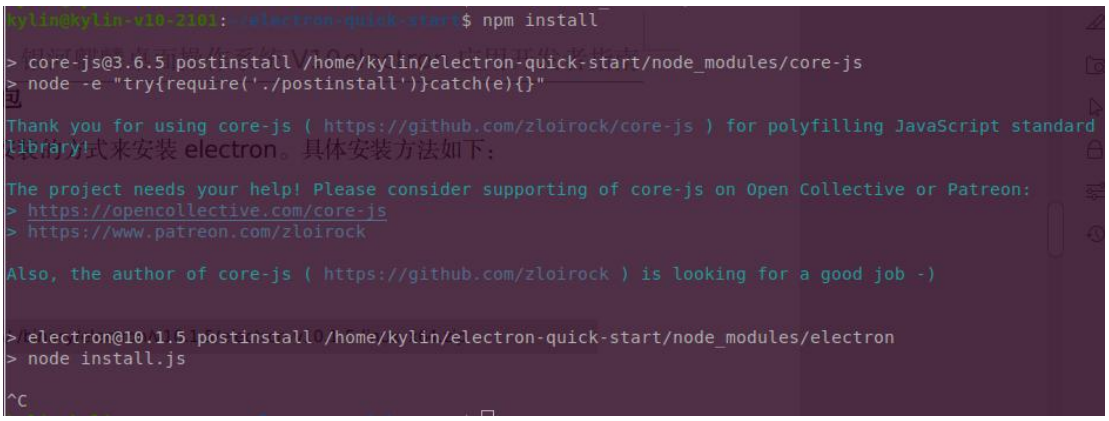
此处以 10.1.5 版本为例：

```
$ cd
$ wget https://registry.npmmirror.com/-/binary/electron/v10.1.5/electron-v10.1.5-linux-x64.zip
$ ls
electron-quick-start                公共的   视频   文档   音乐
electron-v10.1.5-linux-x64.zip  模板     图片   下载   桌面
```

② 安装依赖的模块

```
$ cd electron-quick-start
$ npm install
```

在出现> node install.js 使用 ctrl+c 停止，如下图：



③ 更改 install.js

```
$ vim node_modules/electron/install.js
```

找到下载 electron 的地方，如下

```
// downloads if not cached
downloadArtifact({
  version,
  artifactName: 'electron',
  force: process.env.force_no_cache === 'true',
  cacheRoot: process.env.electron_config_cache,
  platform: process.env.npm_config_platform || process.platform,
  arch: process.env.npm_config_arch || process.arch
}).then(extractFile).catch(err => {
  console.error(err.stack);
  process.exit(1);
});
```

将上述代码注释掉，添加如下代码（根据 extractFile 函数的实现，填写步骤①中下载的 zip 包包名）

```
// downloads if not cached
extractFile('electron-v10.1.5-linux-x64.zip');
//downloadArtifact({
```

```
//    version,
//    artifactName: 'electron',
//    force: process.env.force_no_cache === 'true',
//    cacheRoot: process.env.electron_config_cache,
//    platform: process.env.npm_config_platform || process.platform,
//    arch: process.env.npm_config_arch || process.arch
//}).then(extractFile).catch(err => {
//    console.error(err.stack);
//    process.exit(1);
//});
```

注：

不同版本这个地方的写法不一样，8.*.*之前的版本写法如下:

例如 4.1.3

```
extractFile(0,'electron-v4.1.3-linux-x64.zip');
```

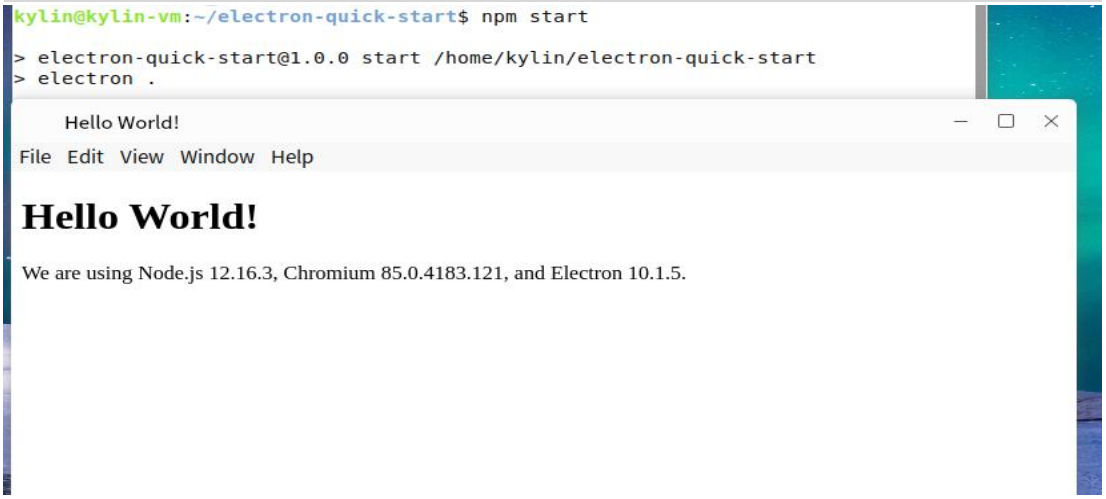
④ 安装 electron

将步骤①中下载的 zip 包拷贝到 electron-quick-start/node_modules/electron 目录下，执行安装命令

```
$ cp ~/electron-v10.1.5-linux-x64.zip node_modules/electron/
$ cd node_modules/electron/
$ node install.js
```

2.3. 启动项目

```
$ cd ~/electron-quick-start
$ npm start
```



如上图所示，即表示项目运行成功。

2.4. 打包

项目调试完成后，需要打包成符合银河麒麟桌面操作系统 V10 软件商店上架规范的 deb 包，X86_64 架构下一般是使用 electron-builder 工具来进行打包。当然，也可以使用 electron-packager 和 electron-installer-debian 工具进行打包，此处以使用 electron-builder 打包为例。

2.4.1. 安装 electron-builder

执行下面的命令来安装 electron-builder，如果后面不加版本号会默认安装当前最新的版本，最新的版本对 node 和 npm 会有一定的要求，安装过程中会有一些告警，可以忽略。

```
$ npm install electron-builder@21.2.0 --save-dev


> ejs@2.7.4 postinstall /home/kylin/electron-quick-start/node_modules/ejs
> node ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)

npm WARN notsup Unsupported engine for fs-extra@10.0.1: wanted: {"node":">=12"} (current: {"node":"10.19.0","npm":"6.13.4"})
npm WARN notsup Not compatible with your version of node/npm: fs-extra@10.0.1

+ electron-builder@21.2.0
added 158 packages from 117 contributors in 14.689s

13 packages are looking for funding
  run `npm fund` for details



┌──────────────────────────────────────────────────────────┐
│                    npm update check failed               │
│              Try running with sudo or get access         │
│              to the local update config store via        │
│ sudo chown -R $USER:$(id -gn $USER) /home/kylin/.config  │
└──────────────────────────────────────────────────────────┘

kylin@kylin-v10-2101:~/electron-quick-start$
```

注：此处使用 electron-builder@21.2.0 版本是由于示例中使用的 node 版本较低，打包时会提示缺少部分模块（Error: Cannot find module 'fs/promises'），处理方法为升级 node 版本或使用降低 electron-builder 的版本，此处采用第二种方法。

2.4.2. 添加打包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron ."
  },
```

添加打包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "builder": "electron-builder"
  },
  "build": {
    "productName": "electron-quick-start",
    "asar": "false",
    "appId": "electron-quick-start",
    "directories": {
      "output": "dist"
    },
    "linux": {
      "icon": "icons",
      "category": "Education",
      "target": {
        "target": "deb",
        "arch": [
          "x64"
        ]
      }
    }
  },
```

注：

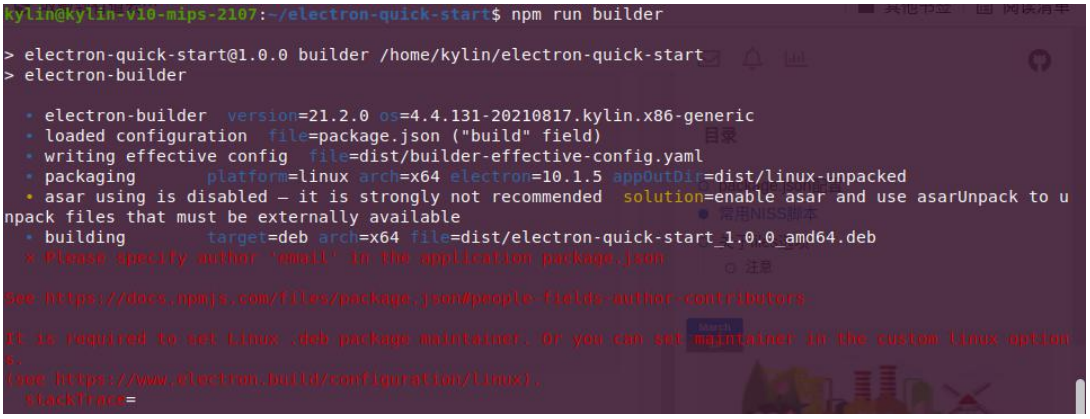
- productName: 项目名,这也是生成的 deb 文件的包名
- appId: 包名
- directories: 目录
  - output: 生成 deb 包的存放目录
- linux: 构建 linux 的选项
  - category: 应用分类

```
#Categories 分类要求：
                安卓 Android；
                网络 Network；
                社交 Messaging；
                影音 Audio、Video；
                开发 Development；
                图像 Graphics；
                游戏 Game；
                办公 Office、Calculator、Spreadsheet、Presentation、WordProcessor、TextEditor；
                教育 Education；
                系统 System、Settings、Security；
```

- target: 目标封装类型，这里要打成 deb 包，所以写成 deb
- icon: 自定义图标路径，如果不指定就用 electron 默认图标
- arch: 架构，这里打的是 amd64 的，写法为 x64

2.4.3. 添加打包者信息

使用 electron-builder 打包时需要使用打包者信息，特别是 name 和 email 字段，不填写打包时会有报错信息：“× Please specify author 'email' in the application package.json”



找到如下代码：

```
    "author": "GitHub",
```

添加打包者信息，修改如下：

```
  "author": {
    "name": "Wu Zhaohui",
    "email": "wuzhaohui@kylinos.cn"
  },
```

2.4.4. 添加应用图标

由于 electron-quick-start 项目中没有自带图标，这里我们需要自己添加一下图标目录及相应尺寸的图标。

```
$ mkdir -p icons
```

添加图标文件

```
$ ls icons
128x128.png  16x16.png  256x256.png  32x32.png  512x512.png  64x64.png
```

另外，某些应用打包后，在系统安装启动后，任务栏图标显示异常，此时可以通过修改 main.js 中的代码来规避此问题。具体操作方法如下：

```
$ vim main.js
```

找到 main.js 中的 createWindow 函数，添加 icon

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
```

```
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

添加 icon，修改后如下：

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    icon: path.join(__dirname, 'icons/512x512.png'),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

注：注意此处图标的写法，如果图标位置不对有可能会造成打包后报：“段错误，核心已转储”，图标需要使用 512x512 以上的 png 格式；

2.4.5. 打 deb 包

使用 electron-builder 打包时会从 github 下载 electron 包及依赖包，如果网络不好的情况，electron 拉取失败就无法打包成功，可以将 1.2.2-b-①中下载的离线包放在~/.cache/electron/目录下，然后再进行打包

```
$ mkdir -p ~/.cache/electron
$ cp ~/electron-v10.1.5-linux-x64.zip ~/.cache/electron/
$ npm run builder
```

打包完成后会有如下提示

```
$ npm run builder
> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder
  • electron-builder  version=21.2.0 os=4.4.131-20210817.kylin.X86_64-generic
  • loaded configuration  file=package.json ("build" field)
  • writing effective config  file=dist/builder-effective-config.yaml
  • packaging  platform=linux arch=x64 electron=10.1.5 appOutDir=dist/linux-unpacked
  • asar using is disabled — it is strongly not recommended  solution=enable asar and use asarUnpack to unpack files that must be externally available
  • building  target=deb arch=x64 file=dist/electron-quick-start_1.0.0_amd64.deb
  • downloading url=https://github.com/electron-userland/electron-builder-binaries/releases/download/fpm-1.9.3-2.3.1-linux-X86_64_64/fpm-1.9.3-2.3.1-linux-X86_64_64.7z size=5.0 MB parts=1
  • downloaded url=https://github.com/electron-userland/electron-builder-binaries/releases/download/fpm-1.9.3-2.3.1-linux-X86_64_64/fpm-1.9.3-2.3.1-linux-X86_64_64.7z duration=29.631s
```

打包后可以看到 dist 目录下生成的 deb 包

```
$ ls dist/
builder-effective-config.yaml    electron-quick-start_1.0.0_amd64.deb    linux-unpacked
```

2.4.6. 修改 deb 包

某些情况下，我们打的包可能会存在一些问题但又不需要重新编译，我们通过下面的命令来对 deb 包进行一定修改。

a. 解包

使用如下命令将打好的 deb 包解包

```
$ fakeroot dpkg-deb -R electron-quick-start_1.0.0_amd64.deb electron-quick-start_1.0.0_amd64
```

按照打包规范对 deb 包进行调试

b. 重新打包

然后，使用如下命令重新打包

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_amd64 .
```

注：

不写打包名称会按照 control 文件自动进行命名打包，会将原包覆盖，可以使用

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_amd64 electron-quick-start_1.0.0_amd64_new.deb
```

命令自定义新包名称来进行打包

2.5. 验包

2.5.1. 安装

在 X86_64 架构机器上，双击或在终端执行 sudo dpkg -i ***.deb 来安装 deb 包

2.5.2. 启动

从开始菜单，找到 electron-quick-start 点击启动，如下图，即表示打包成功

3. ARM64 架构

3.1. 安装开发基础环境

3.1.1. 安装 npm 和 node(如果已经安装请忽略)

a. V10 系统安装 npm 和 node

V10 版本默认没有预装 npm 和 node，需要从源里安装，但源里的 node(4.2.6)和 npm（3.5.2）版本过低，使用 npm install 安装高版本依赖包的时候可能会存在报错，可以通过 npm 在全局安装 n 模块，然后使用 n 模块来安装高版本的 node，此处以安装 10.19.0 版本为例。具体安装方法如下：

```
$ sudo apt update
$ sudo apt install npm -y
$ sudo npm install -g n
$ sudo n 10.19.0
   installing : node-v10.19.0
          mkdir : /usr/local/n/versions/node/10.19.0
          fetch : https://nodejs.org/dist/v10.19.0/node-v10.19.0-linux-arm64.tar.xz
    installed : v10.19.0 (with npm 6.13.4)
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-v10-2101:~$ node -v
v10.19.0
kylin@kylin-v10-2101:~$ npm -v
6.13.4
```

b. V10(SP1)系统安装 npm 和 node

V10(SP1)默认没有预装 npm 和 node，需要从源里安装，源里的 node 版本为 v10.19.0，npm 版本为 v6.14.4。具体安装方法如下：

```
$ sudo apt update
$ sudo apt install npm -y
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-PC:~$ node -v
v10.19.0
kylin@kylin-PC:~$ npm -v
6.14.4
```

如果 V10(SP1)需要其他版本的 node 和 npm，也可以参考 V10 系统中使用 n 模块管理 node 的方式来安装指定版本的 node 和 npm。

3.1.2. 从源中安装 git(如果已经安装请忽略)

```
$ sudo apt install git -y
```

3.1.3. 设置 npm 源（可选）

如果当前网络不能有效的安装 electron，建议将 npm 的镜像源地址修改为国内淘宝源

```
$ npm config set registry https://registry.npm.taobao.org
```

3.2. 项目开发

此处以 electron-quick-start 项目使用 electron-v10.1.5 开发为例：

3.2.1. 下载 electron-quick-start 项目

```
$ git clone https://github.com/electron/electron-quick-start
$ cd electron-quick-start
$ git checkout remotes/origin/10-x-y
```

3.2.2. 安装依赖包

a. 在线安装 electron 及其依赖包

由于 package.json 文件中仅写了 electron 作为开发依赖，所以此处主要是进行 electron 包及其依赖包的安装。具体安装方法如下：

```
$ npm install
> core-js@3.6.5 postinstall /home/kylin/electron-quick-start/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"



> electron@10.1.5 postinstall /home/kylin/electron-quick-start/node_modules/electron
> node install.js

Downloading electron-v10.1.5-linux-arm64.zip: [==================================] 100% ETA: 0.0 seconds
added 87 packages from 98 contributors and audited 87 packages in 89.761s

6 packages are looking for funding
  run `npm fund` for details

found 6 vulnerabilities (3 moderate, 3 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

b. 离线安装 electron 及其依赖包

如果在线安装 electron 失败，可以采用离线安装的方式来安装 electron。具体安装方法如下：

① 下载 electron 离线包

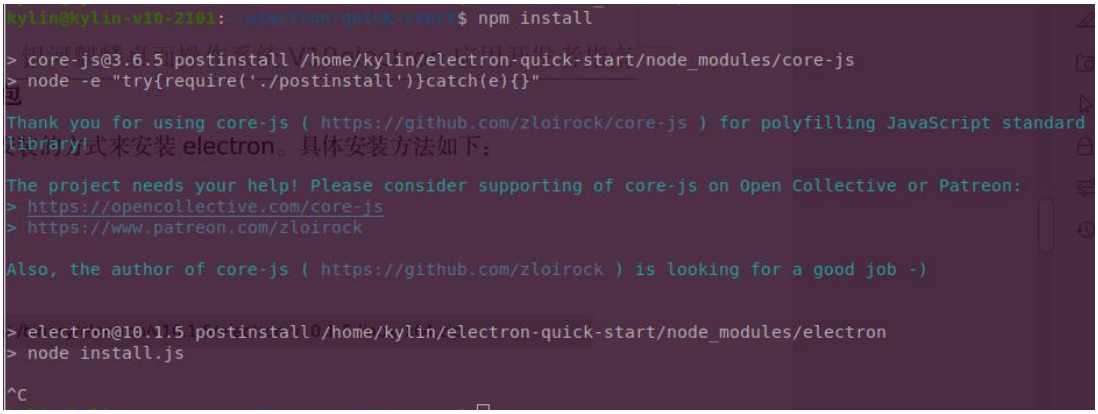
此处以 10.1.5 版本为例：

```
$ cd
$ wget https://registry.npmmirror.com/-/binary/electron/v10.1.5/electron-v10.1.5-linux-arm64.zip
$ ls
electron-quick-start                公共的   视频   文档   音乐
electron-v10.1.5-linux-arm64.zip    模板     图片   下载   桌面
```

② 安装依赖的模块

```
$ cd electron-quick-start
$ npm install
```

在出现> node install.js 使用 ctrl+c 停止，如下图：



③ 更改 install.js

```
$ vim node_modules/electron/install.js
```

找到下载 electron 的地方，如下

```
// downloads if not cached
downloadArtifact({
  version,
  artifactName: 'electron',
  force: process.env.force_no_cache === 'true',
  cacheRoot: process.env.electron_config_cache,
  platform: process.env.npm_config_platform || process.platform,
  arch: process.env.npm_config_arch || process.arch
}).then(extractFile).catch(err => {
  console.error(err.stack);
  process.exit(1);
});
```

将上述代码注释掉，添加如下代码（根据 extractFile 函数的实现，填写步骤①中下载的 zip 包包名）

```
// downloads if not cached
extractFile('electron-v10.1.5-linux-arm64.zip');
//downloadArtifact({
```

```
//    version,
//    artifactName: 'electron',
//    force: process.env.force_no_cache === 'true',
//    cacheRoot: process.env.electron_config_cache,
//    platform: process.env.npm_config_platform || process.platform,
//    arch: process.env.npm_config_arch || process.arch
//}).then(extractFile).catch(err => {
//    console.error(err.stack);
//    process.exit(1);
//});
```

注：

不同版本这个地方的写法不一样，8.*.*之前的版本写法如下:

例如 4.1.3

```
extractFile(0,'electron-v4.1.3-linux-arm64.zip');
```

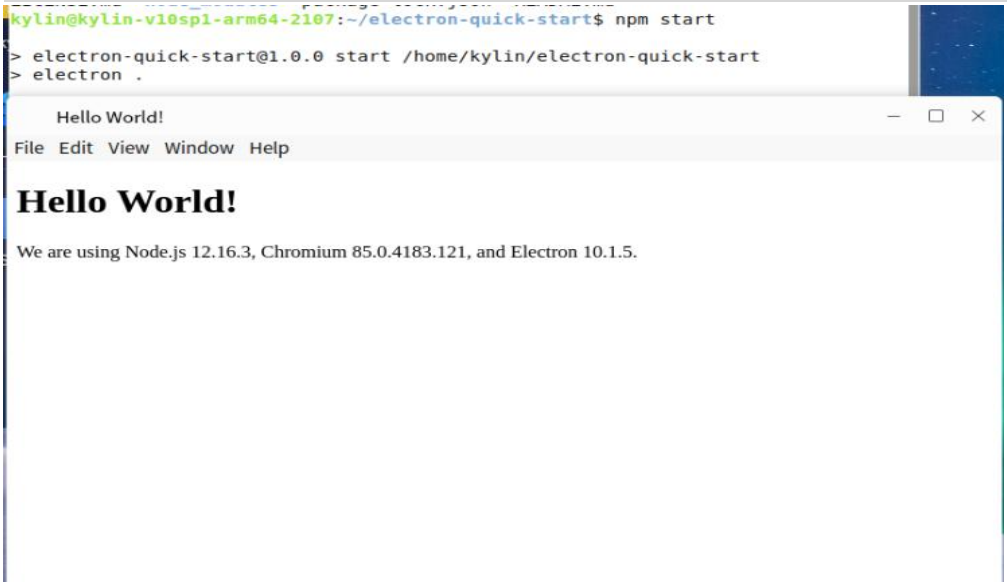
④ 安装 electron

将步骤①中下载的 zip 包拷贝到 electron-quick-start/node_modules/electron 目录下，执行安装命令

```
$ cp ~/electron-v10.1.5-linux-arm64.zip node_modules/electron/
$ cd node_modules/electron/
$ node install.js
```

3.3. 启动项目

```
$ cd ~/electron-quick-start
$ npm start
```



如上图所示，即表示项目运行成功。

3.4. 打包

项目调试完成后，需要打包成符合银河麒麟桌面操作系统 V10 软件商店上架规范的 deb 包，arm64 架构下一般也是使用 electron-builder 工具来进行打包。当然，也可以使用 electron-packager 和 electron-installer-debian 工具进行打包，此处以使用 electron-builder 打包为例。

3.4.1. 安装 electron-builder

执行下面的命令来安装 electron-builder，如果后面不加版本号会默认安装当前最新的版本，最新的版本对 node 和 npm 会有一定的要求，安装过程中会有一些告警，可以忽略。

```
$ npm install electron-builder@21.2.0 --save-dev


> ejs@2.7.4 postinstall /home/kylin/electron-quick-start/node_modules/ejs
> node ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)

npm WARN notsup Unsupported engine for fs-extra@10.0.1: wanted: {"node":">=12"} (current: {"node":"10.19.0","npm":"6.13.4"})
npm WARN notsup Not compatible with your version of node/npm: fs-extra@10.0.1

+ electron-builder@21.2.0
added 158 packages from 117 contributors in 14.689s

13 packages are looking for funding
  run `npm fund` for details
kylin@kylin-v10-2101:~/electron-quick-start$
```

注：此处使用 electron-builder@21.2.0 版本是由于示例中使用的 node 版本较低，打包时会提示缺少部分模块（Error: Cannot find module 'fs/promises'），处理方法为升级 node 版本或使用降低 electron-builder 的版本，此处采用第二种方法。

3.4.2. 添加打包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron ."
  },
```

添加打包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "builder": "electron-builder"
  },
```

```
"build": {
  "productName": "electron-quick-start",
  "asar": "false",
  "appId": "electron-quick-start",
  "directories": {
    "output": "dist"
  },
  "linux": {
    "icon": "icons",
    "category": "Education",
    "target": {
      "target": "deb",
      "arch": [
        "arm64"
      ]
    }
  }
},
```

注：

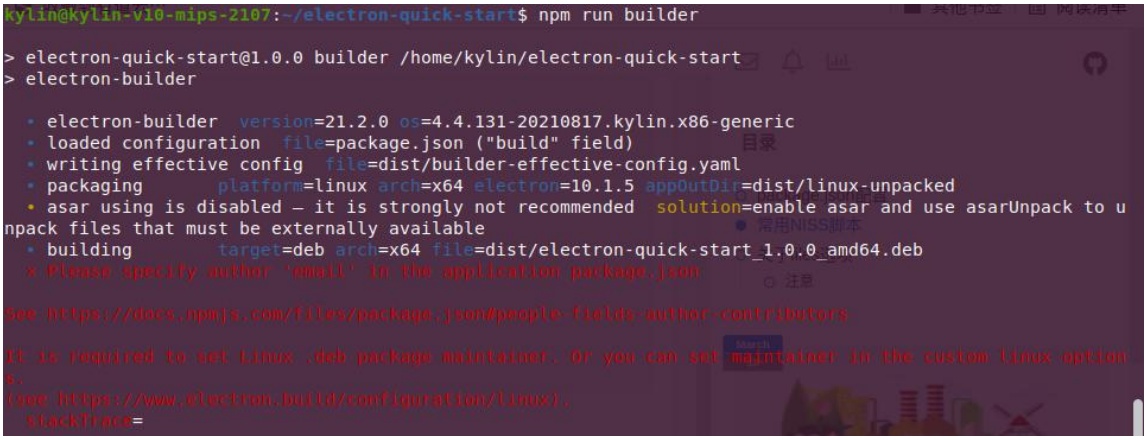
- productName: 项目名,这也是生成的 deb 文件的包名
- appId: 包名
- directories: 目录
  - output: 生成 deb 包的存放目录
- linux: 构建 linux 的选项
  - category: 应用分类

```
#Categories 分类要求：
        安卓 Android；
        网络 Network；
        社交 Messaging；
        影音 Audio、Video；
        开发 Development；
        图像 Graphics；
        游戏 Game；
        办公 Office、Calculator、Spreadsheet、Presentation、WordProcessor、TextEditor；
        教育 Education；
        系统 System、Settings、Security；
```

  - target: 目标封装类型，这里要打成 deb 包，所以写成 deb
  - icon: 自定义图标路径，如果不指定就用 electron 默认图标
  - arch: 架构，这里打的是 arm64 的，写法为 arm64

3.4.3. 添加打包者信息

使用 electron-builder 打包时需要使用打包者信息，特别是 name 和 email 字段，不填写打包时会有报错信息：“× Please specify author 'email' in the application package.json”



找到如下代码：

```
    "author": "GitHub",
```

添加打包者信息，修改如下：

```
  "author": {
    "name": "Wu Zhaohui",
    "email": "wuzhaohui@kylinos.cn"
  },
```

3.4.4. 添加应用图标

由于 electron-quick-start 项目中没有自带图标，这里我们需要自己添加一下图标目录及相应尺寸的图标。

```
$ mkdir -p icons
```

添加图标文件

```
$ ls icons
128x128.png   16x16.png   256x256.png   32x32.png   512x512.png   64x64.png
```

另外，某些应用打包后，在系统安装启动后，任务栏图标显示异常，此时可以通过修改 main.js 中的代码来规避此问题。具体操作方法如下：

```
$ vim main.js
```

找到 main.js 中的 createWindow 函数，添加 icon

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

添加 icon，修改后如下：

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    icon: path.join(__dirname, 'icons/512x512.png'),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

注：注意此处图标的写法，如果图标位置不对有可能会造成打包后报：“段错误，核心已转储”，图标需要使用 512x512 以上的 png 格式；

3.4.5. 安装 fpm

使用 electron-builder 打包时需要用到 fpm 包，但 fpm 包 npm 源中仅有 X86_64 架构的包，没有 arm64 架构的包，打包时会有报错：

```
$ npm run builder

> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder

  • electron-builder  version=21.2.0 os=4.4.131-20210120.kylin.desktop-generic
  • loaded configuration  file=package.json ("build" field)
  • writing effective config  file=dist/builder-effective-config.yaml
  • packaging  platform=linux arch=arm64 electron=10.1.5 appOutDir=dist/linux-arm64-unpacked
  • asar using is disabled — it is strongly not recommended  solution=enable asar and use asarUnpack to unpack files that must be externally available
  • building  target=deb arch=arm64 file=dist/electron-quick-start_1.0.0_arm64.deb
  • default Electron icon is used  reason=application icon is not set
  • downloading url=https://github.com/electron-userland/electron-builder-binaries/releases/download/fpm-1.9.3-2.3.1-linux-X86_64/fpm-1.9.3-2.3.1-linux-X86_64.7z size=4.6 MB parts=1
  • downloaded url=https://github.com/electron-userland/electron-builder-binaries/releases/download/fpm-1.9.3-2.3.1-linux-X86_64/fpm-1.9.3-2.3.1-linux-X86_64.7z duration=2m20.499s
  × cannot execute  cause=exit status 1
                    errorOut=/home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin/ruby:行 6: /home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin.real/ruby: cannot execute binary file: 可执行文件格式错误
    /home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin/ruby:行 6: /home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin.real/ruby: 成功

                    command=/home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/fpm -s dir --force -t deb -d
```

```
libgtk-3-0 -d libnotify4 -d libnss3 -d libxss1 -d libxtst6 -d xdg-utils -d libatspi2.0-0 -d libuuid1 -d libappindicator3-1 -d libsecret-1-0
--deb-compression xz --architecture arm64 --name electron-quick-start --after-install /tmp/t-BtXXvc/0-after-install --after-remove
/tmp/t-BtXXvc/1-after-remove --description '
        A minimal Electron application' --version 1.0.0 --package
/home/kylin/electron-quick-start/dist/electron-quick-start_1.0.0_arm64.deb --maintainer 'Wu Zhaohui <wuzhaohui@kylinos.cn>' --url
'https://github.com/electron/electron-quick-start#readme' --vendor 'Wu Zhaohui <wuzhaohui@kylinos.cn>' --license CC0-1.0
/home/kylin/electron-quick-start/dist/linux-arm64-unpacked/=/opt/electron-quick-start
/home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/16x16.png=/usr/share/icons/hicolor/
16x16/apps/electron-quick-start.png
/home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/32x32.png=/usr/share/icons/hicolor/
32x32/apps/electron-quick-start.png
/home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/48x48.png=/usr/share/icons/hicolor/
48x48/apps/electron-quick-start.png
/home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/64x64.png=/usr/share/icons/hicolor/
64x64/apps/electron-quick-start.png
/home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/128x128.png=/usr/share/icons/hicol
or/128x128/apps/electron-quick-start.png
/home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/256x256.png=/usr/share/icons/hicol
or/256x256/apps/electron-quick-start.png
/tmp/t-BtXXvc/2-electron-quick-start.desktop=/usr/share/applications/electron-quick-start.desktop
                              workingDir=


npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! electron-quick-start@1.0.0 builder: `electron-builder`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the electron-quick-start@1.0.0 builder script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.


npm ERR! A complete log of this run can be found in:
npm ERR!        /home/kylin/.npm/_logs/2022-03-21T08_35_21_121Z-debug.log
kylin@kylin-v10-2101:~/electron-quick-start$
```

我们可以从系统源中安装 ruby，然后使用 gem install fpm 来安装 fpm，然后把 ~/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/fpm 删掉，做一个同名的 fpm 软链接到 /usr/local/bin/fpm。具体操作方法如下：

```
$ sudo apt update
$ sudo apt install ruby
$ sudo gem install fpm
$ cd ~/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/
$ rm -rf fpm
$ ln -s /usr/local/bin/fpm fpm
```

3.4.6. 打 deb 包

使用 electron-builder 打包时会从 github 下载 electron 包及依赖包，如果网络不好的情况，electron 拉取失败就无法打包成功，可以将 2.2.2-b-①中下载的离线包放在~/.cache/electron/目录下，然后再进行打包

```
$ cd ~/electron-quick-start
$ mkdir -p ~/.cache/electron
$ cp ~/electron-v10.1.5-linux-arm64.zip ~/.cache/electron/
$ npm run builder
```

打包完成后会有如下提示

```
$ npm run builder

> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder

  • electron-builder  version=21.2.0 os=4.4.131-20210120.kylin.desktop-generic
  • loaded configuration  file=package.json ("build" field)
  • writing effective config  file=dist/builder-effective-config.yaml
  • packaging  platform=linux arch=arm64 electron=10.1.5 appOutDir=dist/linux-arm64-unpacked
  • asar using is disabled — it is strongly not recommended  solution=enable asar and use asarUnpack to unpack files that must be externally available
  • building  target=deb arch=arm64 file=dist/electron-quick-start_1.0.0_arm64.deb
```

打包后可以看到 dist 目录下生成的 deb 包

```
$ ls dist/
builder-effective-config.yaml  electron-quick-start_1.0.0_arm64.deb  linux-unpacked
```

3.4.7. 修改 deb 包

某些情况下，我们打的包可能会存在一些问题但又不需要重新编译，我们通过下面的命令来对 deb 包进行一定修改。

a. 解包

使用如下命令将打好的 deb 包解包

```
$ fakeroot dpkg-deb -R electron-quick-start_1.0.0_arm64.deb electron-quick-start_1.0.0_arm64
```

按照打包规范对 deb 包进行调试

b. 重新打包

然后，使用如下命令重新打包

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_arm64 .
```

注：

不写打包名称会按照 control 文件自动进行命名打包，会将原包覆盖，可以使用

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_arm64 electron-quick-start_1.0.0_arm64_new.deb
```

命令自定义新包名称来进行打包

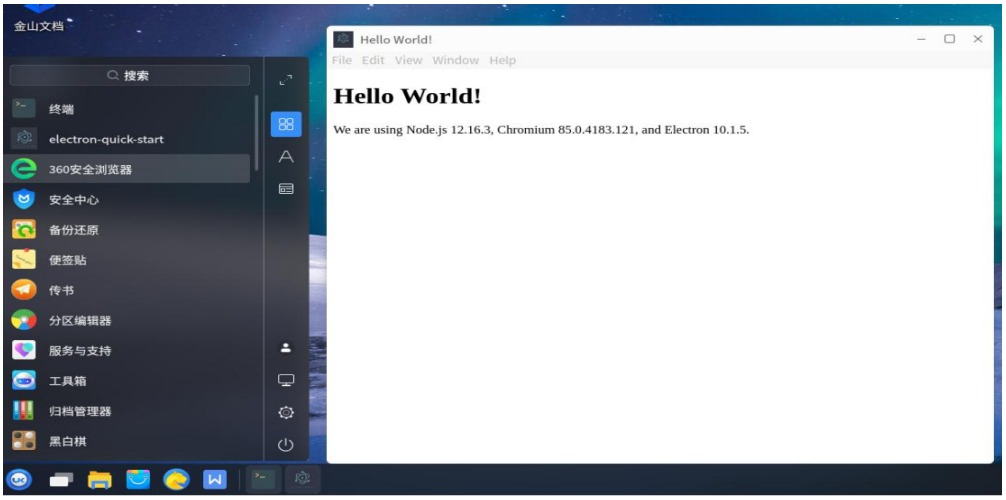
3.5. 验包

3.5.1. 安装

在 AMR64 架构机器上，双击或在终端执行 sudo dpkg -i ***.deb 来安装 deb 包

3.5.2. 启动

从开始菜单，找到 electron-quick-start 点击启动，如下图，即表示打包成功



4. MIPS64 架构

MIPS 架构与其他两个架构（X86_64 和 ARM64）electron 应用打包有些差异，因为 npm 源中没有 MIPS64 架构的 electron 包(或者说在 npmjs 这个网站上龙芯版本的 electron 不是这个包名)。实际包名：loongson-electron，版本有 4.1.3 6.1.7 10.1.0，使用 npm 下载时的命令为:

$ npm install loongson-electron@***

例如：$ npm install loongson-electron@6.1.7

建议安装 6.1.7 以上版本，测试发现 4.1.3 版本在麒麟 v10 系统存在键盘输入界面闪退的情况。

还有一个相关的包，包名：electron-mips，版本都是 10.1.0 以上的版本

如果网络不好的情况下，会存在线装包失败的情况，我们可以通过离线安装的方式来安装 electron，下面就来介绍如何离线安装 electron。

龙芯开源社区提供了 MIPS64 架构上适配的 electron 包，从龙芯开源社区下载 electron 离线包，然后进行离线安装，具体操作方法如下：

4.1. 安装开发基础环境

4.1.1. 安装 npm 和 node(如果已经安装请忽略)

a. V10 系统安装 npm 和 node

V10-2107 默认预装 node（8.9.4）和 npm（5.6.0），V10-2107 之前的版本默认没有预装，需要从源里安装，但源里的 node（4.2.6）和 npm（3.5.2）版本过低，使用 npm install 安装其他模块的时候可能会存在报错，麒麟内部提供 node（12.19.1）和 npm（6.14.8）离线安装版本，有需要可以通过内部获取。具体安装方法如下：

```
$ tar zxvf node-v10-12.19.1.tar.gz
$ cd node-v10-12.19.1
$ sudo apt install ./openssl1.1.1b/*.deb
$ sudo apt install ./python3.7/*.deb
$ sudo apt install ./nodejs_12.19.1_mips64el.deb
$ npm config set python python3.7
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-KaiTianM530Z:~/node-v10-12.19.1$ node -v
v12.19.1
kylin@kylin-KaiTianM530Z:~/node-v10-12.19.1$ npm -v
6.14.8
```

b. V10(SP1)系统安装 npm 和 node

```
$ sudo apt update
$ sudo apt install npm -y
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-PC:~$ node -v
v10.19.0
kylin@kylin-PC:~$ npm -v
```

```
6.14.4
```

4.1.2. 从源中安装 git(如果已经安装请忽略)

```
$ sudo apt install git -y
```

4.1.3. 设置 npm 源（可选）

如果，当前网络不能有效的安装 electron，建议将 npm 的镜像源地址修改为国内源

```
$ npm config set registry https://registry.npm.taobao.org
```

4.2. 项目开发

此处以 electron-quick-start 项目使用 electron-v10.1.0 开发为例：

4.2.1. 下载 electron-quick-start 项目

```
$ git clone https://github.com/electron/electron-quick-start
$ cd electron-quick-start
$ git checkout remotes/origin/10-x-y
```

4.2.2. 离线安装 electron 及其依赖包

建议采用离线安装的方式来安装 electron。具体安装方法如下：

a. 下载 electron 离线包

从龙芯开源社区下载 electron 离线包，将 tar 包解包，重新压缩成 zip 包
此处以 10.1.0 版本为例：

```
$ wget http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-v10.1.0/electron_v10.1.0_kylin_v10.tar.gz
$ tar zxvf electron_v10.1.0_kylin_v10.tar.gz
$ cd electron_v10.1.0_kylin_v10
$ zip -r electron-v10.1.0-linux-mips64el.zip ./*
$ ls
chrome_100_percent.pak          libEGL.so                 LICENSE                       swiftshader
chrome_200_percent.pak          libffmpeg.so              LICENSES.chromium.html        v8_context_snapshot.bin
chrome-sandbox                  libGLESv2.so              locales                       version
electron                        libminigbm.so             resources                     vk_swiftshader_icd.json
electron-v10.1.0-linux-mips64el.zip  libvk_swiftshader.so  resources.pak
icudtl.dat                      libvulkan.so              snapshot_blob.bin
```

注：

请注意 zip 包的包名为固定格式 electron-v*.*.*-linux-arch.zip

使用命令查询当前系统的架构，将 zip 包中的架构名称修改成对应的架构名称，此处为 mips64el

```
$ archdetect
mips64el/loongson-3
```

下载链接：

```
electron-v4.1.3:
http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-v4.1.3/electron_v4.1.3.tar.gz
electron-v6.1.7:
http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-v6.1.7/electron_v6.1.7.tar.gz
electron-v10.1.0:
http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-v10.1.0/electron_v10.1.0_kylin_v10.tar.gz
$ npm install electron@10.1.0
```

b. 安装依赖的模块

```
$ cd electron-quick-start
$ npm install electron@10.1.0 --save-dev
```

在出现> node install.js 使用 ctrl+c 停止，如下图：

```
kylin@kylin-v10sp1-mips-2107:~/electron-quick-start$ npm install electron@10.1.0

> electron@10.1.0 postinstall /home/kylin/electron-quick-start/node_modules/elec
tron
> node install.js

^C
kylin@kylin-v10sp1-mips-2107:~/electron-quick-start$
```

c. 更改 install.js

```
$ vim node_modules/electron/install.js
```

找到下载 electron 的地方，如下

```
// downloads if not cached
downloadArtifact({
  version,
  artifactName: 'electron',
  force: process.env.force_no_cache === 'true',
  cacheRoot: process.env.electron_config_cache,
  platform: process.env.npm_config_platform || process.platform,
  arch: process.env.npm_config_arch || process.arch
}).then(extractFile).catch(err => {
  console.error(err.stack);
  process.exit(1);
});
```

将上述代码注释掉，添加如下代码（根据 extractFile 函数的实现，填写步骤 a 中制作好的 zip 包包名）

```
// downloads if not cached
extractFile('electron-v10.1.0-linux-mips64el.zip');
//downloadArtifact({
```

```
//    version,
//    artifactName: 'electron',
//    force: process.env.force_no_cache === 'true',
//    cacheRoot: process.env.electron_config_cache,
//    platform: process.env.npm_config_platform || process.platform,
//    arch: process.env.npm_config_arch || process.arch
//}).then(extractFile).catch(err => {
//    console.error(err.stack);
//    process.exit(1);
//});
```

注：

不同版本这个地方的写法不一样，4.1.3 和 6.1.7 版本修改后如下:

4.1.3

```
extractFile(0,'electron-v4.1.3-linux-mips64el.zip');
```

6.1.7

```
extractFile(0,'electron-v6.1.7-linux-mips64el.zip');
```

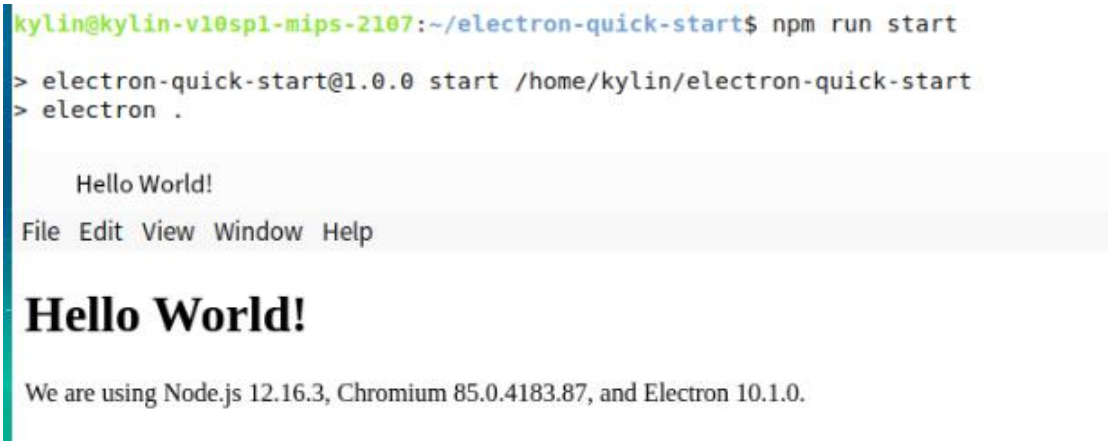
d. 安装 electron

将步骤 a 中重新打包的 zip 包放到 electron-quick-start/node_modules/electron 目录下，执行安装命令

```
$ cp ~/electron_v10.1.0_kylin_v10/electron-v10.1.0-linux-mips64el.zip node_modules/electron/
$ cd node_modules/electron/
$ node install.js
```

4.3. 启动项目

```
$ cd ~/electron-quick-start
$ npm start
```



如上图所示，即表示项目运行成功。

注：

V10 系统上面执行 npm strart 的时候可能会有报错，报错如下：

```
$ npm start
```

```
> electron-quick-start@1.0.0 start /home/kylin/electron-quick-start
> electron .

[22583:0406/173856.497761:FATAL:setuid_sandbox_host.cc(158)] The SUID sandbox helper binary was found, but is not configured
correctly. Rather than run without sandboxing I'm aborting now. You need to make sure that
/home/kylin/electron-quick-start/node_modules/electron/dist/chrome-sandbox is owned by root and has mode 4755.
/home/kylin/electron-quick-start/node_modules/electron/dist/electron exited with signal SIGTRAP
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! electron-quick-start@1.0.0 start: `electron .`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the electron-quick-start@1.0.0 start script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /home/kylin/.npm/_logs/2022-04-06T09_38_56_664Z-debug.log
```

根据报错提示可以看出是 chrome-sandbox 权限问题，需要进行权限修改

```
$ sudo chown root:root /home/kylin/electron-quick-start/node_modules/electron/dist/chrome-sandbox
$ sudo chmod 4755 /home/kylin/electron-quick-start/node_modules/electron/dist/chrome-sandbox
```

4.4. 打包（electron-builder 方式）

项目调试完成后，需要打包成符合银河麒麟桌面操作系统 V10 软件商店上架规范的 deb 包，X86_64 和 ARM64 架构下一般是使用 electron-builder 工具来进行打包，但 electron-builder 不支持 MIPS64 架构打包，需要进行一些调整才能正常使用，具体操作如下：

4.4.1. 安装 electron-builder

安装 electron-builder 的时候会读取 package-lock.json 中 electron 的版本，但文件中使用的是 10.1.5 版本，此处使用的是 10.1.0 版本，所以需要做一些修改，不然执行 npm install electron-builder 的时候会重新覆盖安装 electron@10.1.5，导致安装失败。具体修改如下：

a. 修改 electron 版本为当前使用版本

```
$ cd ~/electron-quick-start
$ vim package.json
```

找到如下代码

```
"devDependencies": {
    "electron": "^10.1.5"
  }
```

修改为：

```
"devDependencies": {
    "electron": "^10.1.0"
  }
```

此处还需要删除 package-lock.json,不然安装 electron-packager 的时候还会在线安装 10.1.5 的版本

```
$ rm -rf package-lock.json
```

b. 在线安装 electron-builder

执行下面的命令来安装 electron-builder，如果后面不加版本号会默认安装当前最新的版本，最新的版本对 node 和 npm 会有一定的要求，安装过程中会有一些告警，可以忽略。

```
$ npm install electron-builder@21.2.0 --save-dev

> ejs@2.7.4 postinstall /home/kylin/electron-quick-start/node_modules/ejs
> node ./postinstall.js

Thank you for installing EJS: built with the Jake JavaScript build tool (https://jakejs.com/)

npm notice created a lockfile as package-lock.json. You should commit this file.
+ electron-builder@21.2.0
added 159 packages from 121 contributors in 22.901s

12 packages are looking for funding
  run `npm fund` for details
```

注：

此处使用 electron-builder@21.2.0 版本是由于示例中使用的 node 版本较低，打包时会提示缺少部分模块（Error: Cannot find module 'fs/promises'），处理方法为升级 node 版本或使用降低 electron-builder 的版本，此处采用第二种方法。

另外，由于 electron-builder 不支持 MIPS64 架构打包，所以需要修改源码后重新编译 electron-builder，具体参考附录 6.1。需要注意的是，重新编译的版本和项目中安装的 electron-builder 版本需要保持一致，否则打包的时候会有报错。

c. 替换 electron-builder 中的 builder-util

拷贝附录 6.1 中编译成功的文件（packages/builder-util/out/*）覆盖目标 electron 工程里 node_modules/builder-util/out/*目录下，操作方法如下：

```
$ cp ~/electron-builder/electron-builder-21.2.0/packages/builder-util/out/* ~/electron-quick-start/node_modules/builder-util/out/ -r
```

如果不想编译，也可以直接下载已经编译好的 builder-util/out/文件，放到目标 electron 工程里 node_modules/builder-util/out/*目录下，前提条件是安装的也是 electron-builder@21.2.0 版本

下载链接：

https://gitee.com/wuzhaohui891999/electron-builder-mips64el/raw/master/builder-util.zip

```
$ wget https://gitee.com/wuzhaohui891999/electron-builder-mips64el/raw/master/builder-util.zip
$ unzip builder-util.zip
$ mv out/* ~/electron-quick-start/node_modules/builder-util/out/
```

注：

此修改是为了修改打包过程中下列报错：

```
$ npm run builder

> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder

  • electron-builder  version=21.2.0 os=4.4.131-20210817.kylin.desktop-generic
  • loaded configuration  file=package.json ("build" field)
  × Invalid configuration object. electron-builder 21.2.0 has been initialised using a configuration object that does not match the API schema.
 - configuration.linux.target.arch[0] should be one of these:
    "arm64" | "armv7l" | "ia32" | "x64"
      How to fix:
      1. Open https://www.electron.build/configuration/linux
      2. Search the option name on the page (or type in into Search to find across the docs).
        * Not found? The option was deprecated or not exists (check spelling).
        * Found? Check that the option in the appropriate place. e.g. "title" only in the "dmg", not in the root.
  stackTrace=
```

d. 替换 app-builder

app-builder 需要手动编译，如果不想编译可以从下面的链接获取

app-builder 下载链接:

https://gitee.com/wuzhaohui891999/electron-builder-mips64el/raw/master/app-builder

手动编译方法见附录 6.2。

```
$ mkdir -p ~/electron-quick-start/node_modules/app-builder-bin/linux/mips64el
$ cd ~/electron-quick-start/node_modules/app-builder-bin/linux/mips64el
$ wget https://gitee.com/wuzhaohui891999/electron-builder-mips64el/raw/master/app-builder
$ chmod +x app-builder
```

注：

此修改是为了修改打包过程中的如下报错：

```
$ npm run builder


> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder


  • electron-builder  version=21.2.0 os=4.4.131-20210817.kylin.desktop-generic
  • loaded configuration  file=package.json ("build" field)
  • writing effective config  file=dist/builder-effective-config.yaml
  × spawn /home/kylin/electron-quick-start/node_modules/app-builder-bin/linux/mips64el/app-builder ENOENT  stackTrace=
```

4.4.2. 添加打包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron ."
  },
```

添加打包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "builder": "electron-builder"
  },
  "build": {
    "productName": "electron-quick-start",
    "asar": "false",
    "appId": "electron-quick-start",
    "directories": {
      "output": "dist"
    },
    "linux": {
      "icon": "icons",
      "category": "Education",
      "target": "deb"
    }
  },
```

注：

- productName: 项目名,这也是生成的 deb 文件的包名

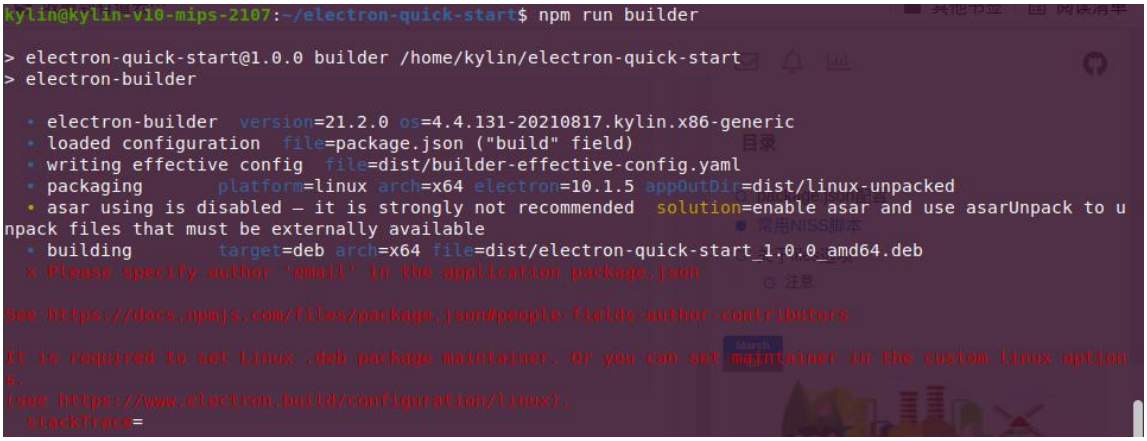
- appId: 包名
- directories: 目录
  - output: 生成 deb 包的存放目录
- linux: 构建 linux 的选项
  - category: 应用分类

```
#Categories 分类要求：
        安卓 Android；
        网络 Network；
        社交 Messaging；
        影音 Audio、Video；
        开发 Development；
        图像 Graphics；
        游戏 Game；
        办公 Office、Calculator、Spreadsheet、Presentation、WordProcessor、TextEditor；
        教育 Education；
        系统 System、Settings、Security；
```

  - target: 目标封装类型，这里要打成 deb 包，所以写成 deb
  - icon: 自定义图标路径，如果不指定就用 electron 默认图标
  - arch: 架构，这里打的是 arm64 的，写法为 arm64

4.4.3. 添加打包者信息

使用 electron-builder 打包时需要使用打包者信息，特别是 name 和 email 字段，不填写打包时会有报错信息：“× Please specify author 'email' in the application package.json”



找到如下代码：

```
    "author": "GitHub",
```

添加打包者信息，修改如下：

```
  "author": {
    "name": "Wu Zhaohui",
    "email": "wuzhaohui@kylinos.cn"
  },
```

4.4.4. 添加应用图标

由于 electron-quick-start 项目中没有自带图标，这里我们需要自己添加一下图标目录及相应尺寸的图标。

```
$ mkdir -p icons
```

添加图标文件

```
$ ls icons
128x128.png   16x16.png   256x256.png   32x32.png   512x512.png   64x64.png
```

另外，某些应用打包后，在系统安装启动后，任务栏图标显示异常，此时可以通过修改 main.js 中的代码来规避此问题。具体操作方法如下：

```
$ vim main.js
```

找到 main.js 中的 createWindow 函数，添加 icon

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

添加 icon，修改后如下：

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    icon: path.join(__dirname, 'icons/512x512.png'),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

注：注意此处图标的写法，如果图标位置不对有可能会造成打包后报：“段错误，核心已转储”，图标需要使用 512x512 以上的 png 格式；

4.4.5. 安装 fpm

使用 electron-builder 打包时需要用到 fpm 包，但 fpm 包 npm 源中仅有 X86_64 架构的包，没有 MIPS64 架构的包，打包时会有报错：

```
$ npm run builder


> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder
```

```
• electron-builder  version=21.2.0 os=4.4.131-20210120.kylin.desktop-generic
• loaded configuration  file=package.json ("build" field)
• writing effective config  file=dist/builder-effective-config.yaml
• packaging  platform=linux arch=arm64 electron=10.1.5 appOutDir=dist/linux-arm64-unpacked
• asar using is disabled — it is strongly not recommended  solution=enable asar and use asarUnpack to unpack files that must be externally available
• building  target=deb arch=arm64 file=dist/electron-quick-start_1.0.0_arm64.deb
• default Electron icon is used  reason=application icon is not set
• downloading url=https://github.com/electron-userland/electron-builder-binaries/releases/download/fpm-1.9.3-2.3.1-linux-X86_64/fpm-1.9.3-2.3.1-linux-X86_64.7z size=4.6 MB parts=1
• downloaded url=https://github.com/electron-userland/electron-builder-binaries/releases/download/fpm-1.9.3-2.3.1-linux-X86_64/fpm-1.9.3-2.3.1-linux-X86_64.7z duration=2m20.499s
× cannot execute  cause=exit status 1
                errorOut=/home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin/ruby:行 6: /home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin.real/ruby: cannot execute binary file: 可执行文件格式错误
    /home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin/ruby:行 6: /home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/lib/ruby/bin.real/ruby: 成功

                command=/home/kylin/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/fpm -s dir --force -t deb -d libgtk-3-0 -d libnotify4 -d libnss3 -d libxss1 -d libxtst6 -d xdg-utils -d libatspi2.0-0 -d libuuid1 -d libappindicator3-1 -d libsecret-1-0 --deb-compression xz --architecture arm64 --name electron-quick-start --after-install /tmp/t-BtXXvc/0-after-install --after-remove /tmp/t-BtXXvc/1-after-remove --description '
    A minimal Electron application' --version 1.0.0 --package /home/kylin/electron-quick-start/dist/electron-quick-start_1.0.0_arm64.deb --maintainer 'Wu Zhaohui <wuzhaohui@kylinos.cn>' --url 'https://github.com/electron/electron-quick-start#readme' --vendor 'Wu Zhaohui <wuzhaohui@kylinos.cn>' --license CC0-1.0 /home/kylin/electron-quick-start/dist/linux-arm64-unpacked/=/opt/electron-quick-start /home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/16x16.png=/usr/share/icons/hicolor/16x16/apps/electron-quick-start.png /home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/32x32.png=/usr/share/icons/hicolor/32x32/apps/electron-quick-start.png /home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/48x48.png=/usr/share/icons/hicolor/48x48/apps/electron-quick-start.png /home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/64x64.png=/usr/share/icons/hicolor/64x64/apps/electron-quick-start.png /home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/128x128.png=/usr/share/icons/hicolor/128x128/apps/electron-quick-start.png /home/kylin/electron-quick-start/node_modules/app-builder-lib/templates/icons/electron-linux/256x256.png=/usr/share/icons/hicolor/256x256/apps/electron-quick-start.png /tmp/t-BtXXvc/2-electron-quick-start.desktop=/usr/share/applications/electron-quick-start.desktop
                workingDir=
```

```
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! electron-quick-start@1.0.0 builder: `electron-builder`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the electron-quick-start@1.0.0 builder script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /home/kylin/.npm/_logs/2022-03-21T08_35_21_121Z-debug.log
kylin@kylin-v10-2101:~/electron-quick-start$
```

我们可以从系统源中安装 ruby，然后使用 gem install fpm 来安装 fpm，然后把 ~/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/fpm 删掉，做一个同名的 fpm 软链接到 /usr/local/bin/fpm。如果没有可以直接创建此目录，然后直接做软链接。具体操作方法如下：

```
$ sudo apt update
$ sudo apt install ruby
$ sudo gem install fpm
$ mkdir -p ~/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/
$ cd ~/.cache/electron-builder/fpm/fpm-1.9.3-2.3.1-linux-X86_64/
$ rm -rf fpm
$ ln -s /usr/local/bin/fpm fpm
```

4.4.6. 打 deb 包

使用 electron-builder 打包时会从 github 下载 electron 包及依赖包，如果网络不好的情况，electron 拉取失败就无法打包成功，可以将 4.2.2-b-①中下载的离线包放在~/.cache/electron/目录下，然后再进行打包

```
$ cd ~/electron-quick-start
$ mkdir -p ~/.cache/electron
$ cp ~/electron_v10.1.0_kylin_v10/electron-v10.1.0-linux-mips64el.zip ~/.cache/electron
$ npm run builder
```

打包完成后会有如下提示

```
$ npm run builder

> electron-quick-start@1.0.0 builder /home/kylin/electron-quick-start
> electron-builder

  • electron-builder  version=21.2.0 os=4.4.131-20210120.kylin.desktop-generic
  • loaded configuration  file=package.json ("build" field)
  • writing effective config  file=dist/builder-effective-config.yaml
  • packaging  platform=linux arch=mips64el electron=10.1.5 appOutDir=dist/linux-mips64el-unpacked
  • asar using is disabled — it is strongly not recommended  solution=enable asar and use asarUnpack to unpack files that must be
externally available
  • building  target=deb arch=mips64el file=dist/electron-quick-start_1.0.0_mips64el.deb
```

打包后可以看到 dist 目录下生成的 deb 包

```
$ ls dist/
builder-effective-config.yaml    electron-quick-start_1.0.0_mips64el.deb    linux-unpacked
```

注：

V10 系统上如果打好的包无法运行，并且从命令行运行报错如下：

```
$ ./electron-quick-start
[15793:0407/171417.990981:FATAL:setuid_sandbox_host.cc(158)] The SUID sandbox helper binary was found, but is not configured correctly. Rather than run without sandboxing I'm aborting now. You need to make sure that /home/kylin/electron-quick-start/dist/linux-mips64el-unpacked/chrome-sandbox is owned by root and has mode 4755.
追踪与中断点陷阱 (核心已转储)
```

根据报错提示可以看出是 chrome-sandbox 权限问题，需要参考 4.5.5 解包后进行权限修改，然后重新打包

```
$ sudo chown root:root ~/electron-quick-start_1.0.0_mips64el/opt/electron-quick-start/chrome-sandbox
$ sudo chmod 4755 ~/electron-quick-start_1.0.0_mips64el/opt/electron-quick-start/chrome-sandbox
```

4.5. 打包(electron-packager+electron-installer-debian 方式)

MIPS64 架构下使用 electron-builder 工具打包需要调整的内容比较多，如果觉得麻烦，可以使用 electron-packager + electron-installer-debian 工具替换 electron-builder 来进行打包，下面使用 electron-packager + electron-installer-debian 工具进行打包演示。

4.5.1. 安装 electron-packager

安装 electron-packager 的时候会读取 package-lock.json 中 electron 的版本，但文件中使用的是 10.1.5 版本，此处使用的是 10.1.0 版本，所以需要做一些修改，不然执行 npm install electron-packager 的时候会重新覆盖安装 electron。

a. 修改 electron 版本为当前使用版本

```
$ cd ~/electron-quick-start
$ vim package.json
```

找到如下代码

```
"devDependencies": {
    "electron": "^10.1.5"
  }
```

修改为：

```
"devDependencies": {
    "electron": "^10.1.0"
  }
```

此处还需要删除 package-lock.json,不然安装 electron-packager 的时候还会在线安装 10.1.5 的版本

```
$ rm -rf package-lock.json
```

b. 在线安装 electron-packager

```
$ npm install electron-packager --save-dev
```

c. 修改 electron-packager 源码以支持 MIPS 架构的高版本 electron

```
$ vim node_modules/electron-packager/src/targets.js
```

找到如下代码

```
  linux: {
    arm64: '>= 1.8.0',
    mips64el: '^10.1.0'
  },
```

修改 mips64 架构版本信息

```
  linux: {
    arm64: '>= 1.8.0',
    mips64el: '^10.1.0'
  },
```

4.5.2. 添加应用图标

由于 electron-quick-start 项目中没有自带图标，这里我们需要自己添加一下图标目录及相应尺寸的图标。

```
$ mkdir -p icons
```

添加图标文件

```
$ ls icons
128x128.png   16x16.png   256x256.png   32x32.png   512x512.png   64x64.png
```

另外，某些应用打包后，在系统安装启动后，任务栏图标显示异常，此时可以通过修改 main.js 中的代码来规避此问题。具体操作方法如下：

```
$ vim main.js
```

找到 main.js 中的 createWindow 函数，添加 icon

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

添加 icon，修改后如下：

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    icon: path.join(__dirname, 'icons/512x512.png'),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

注：注意此处图标的写法，如果图标位置不对有可能会造成打包后报：“段错误，核心已转储”，图标需要使用 512x512 以上的 png 格式；

4.5.3. 打 unpack 包

a. 添加打包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron ."
```

```
  },
```

添加打包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "packager": "electron-packager . --overwrite --platform=linux --arch=mips64el --out=dist/    --app-version=10.1.0
--electron-zip-dir=/home/kylin/electron_v10.1.0_kylin_v10/"
  },
```

注：

--out=dist/指的是生成 unpack 包目录

--app-version=10.1.0 指的是使用的 electron 版本号

--electron-zip-dir=/home/kylin/指的是 electron zip 离线包放置目录

b. 打 unpack 包

将 electron-quick-start 打包

```
$ npm run packager
```

打包完成后会有如下提示：

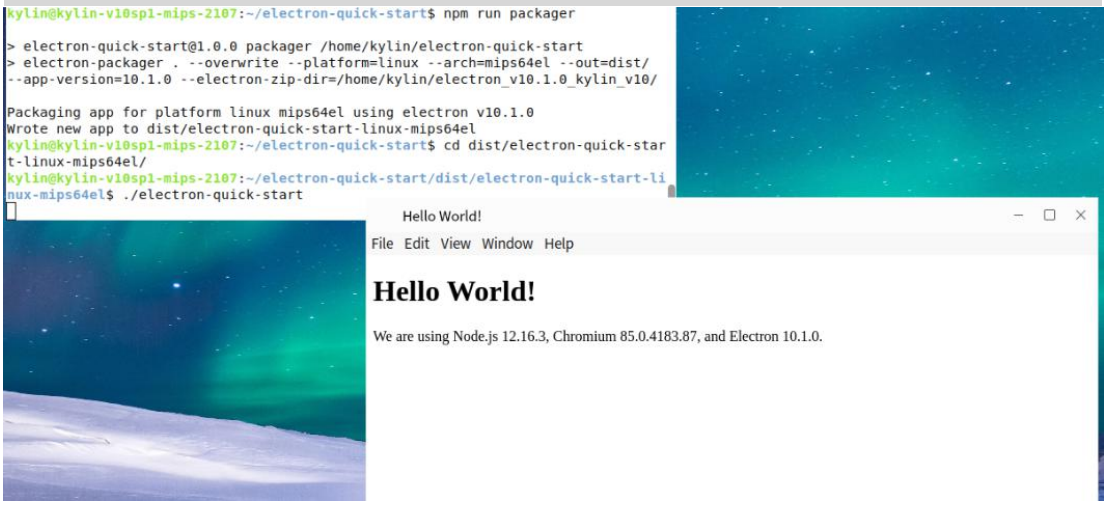
```
kylin@kylin-v10sp1-mips-2107:~/electron-quick-start$ npm run packager


> electron-quick-start@1.0.0 packager /home/kylin/electron-quick-start
> electron-packager . --overwrite --platform=linux --arch=mips64el --out=dist/    --app-version=10.1.0
--electron-zip-dir=/home/kylin/electron_v10.1.0_kylin_v10/


Packaging app for platform linux mips64el using electron v10.1.0
Wrote new app to dist/electron-quick-start-linux-mips64el
```

打包完成，可以看到当前目录下生成了一个 dist/electron-quick-start-linux-mips64el 目录，进入会后执行 electron-quick-start 二级制程序即可出现 hello，world 页面。

```
$ cd dist/electron-quick-start-linux-mips64el
$ ./electron-quick-start
```



4.5.4. 打 deb 包

上面使用 electron-packager 只是打成了 unpack 包（二进制包），要想打成符合麒麟软件上架要求的 deb 包，还需要使用 electron-installer-debian 打成 deb 包，然后参考开发者指南或打包规范进一步修改打成 deb 包

a. 安装 electron-installer-debian

```
$ cd ~/electron-quick-start
$ npm install electron-installer-debian --save-dev
```

b. 添加打 deb 包脚本

```
$vim config.json
```

添加如下代码

```
{
  "src": "dist/electron-quick-start-linux-mips64el/",
  "dest": "dist/deb/",
  "arch": "mips64el",
  "icon": {
    "16x16": "icons/16x16.png",
    "32x32": "icons/32x32.png",
    "64x64": "icons/64x64.png",
    "128x128": "icons/128x128.png",
    "256x256": "icons/256x256.png",
    "512x512": "icons/512x512.png"
  },
  "categories": [
    "Utility"
  ],
  "lintianOverrides": [
    "changelog-file-missing-in-native-package"
  ]
}
```

注：

src 指的是 unpack 包的目录

dest 指的是生成 deb 包的目录

c. 添加打 deb 包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron .",
    "packager": "electron-packager . --overwrite --platform=linux --arch=mips64el --out=dist/   --app-version=10.1.0
--electron-zip-dir=/home/kylin/electron_v10.1.0_kylin_v10/"
  },
```

添加打 deb 包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "packager": "electron-packager . --overwrite --platform=linux --arch=mips64el --out=dist/    --app-version=10.1.0
--electron-zip-dir=/home/kylin/electron_v10.1.0_kylin_v10/",
    "deb": "electron-installer-debian --config config.json"
  },
```

注:

--config config.json 指的是使用 config.json 脚本中的参数来进行打包

如果不想使用脚本也可以直接使用参数打包，例如：

```
"deb": "electron-installer-debian --src dist/electron-quick-start-linux-mips64el --dest dist/deb/ --icons icons --arch mips64el",
```

d. 打 deb 包

```
$ npm run deb
```

打包完成后会有如下提示

```
kylin@kylin-v10sp1-mips-2107:~/electron-quick-start$ npm run deb

> electron-quick-start@1.0.0 deb /home/kylin/electron-quick-start
> electron-installer-debian --src dist/electron-quick-start-linux-mips64el --dest dist/deb/ --arch mips64el

Creating package (this may take a while)
Successfully created package at dist/deb/
```

4.5.5. 修改 deb 包

a. 解包

使用如下命令将打好的 deb 包解包

```
$ fakeroot dpkg-deb -R electron-quick-start_1.0.0_mips64el.deb electron-quick-start_1.0.0_mips64el
```

按照打包规范对 deb 包进行调试

b. 重新打包

然后，使用如下命令重新打包

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_mips64el .
```

注：

不写打包名称会按照 control 文件自动进行命名打包，会将原包覆盖，可以使用

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_mips64el electron-quick-start_1.0.0_mips64el_new.deb
```

命令自定义新包名称来进行打包

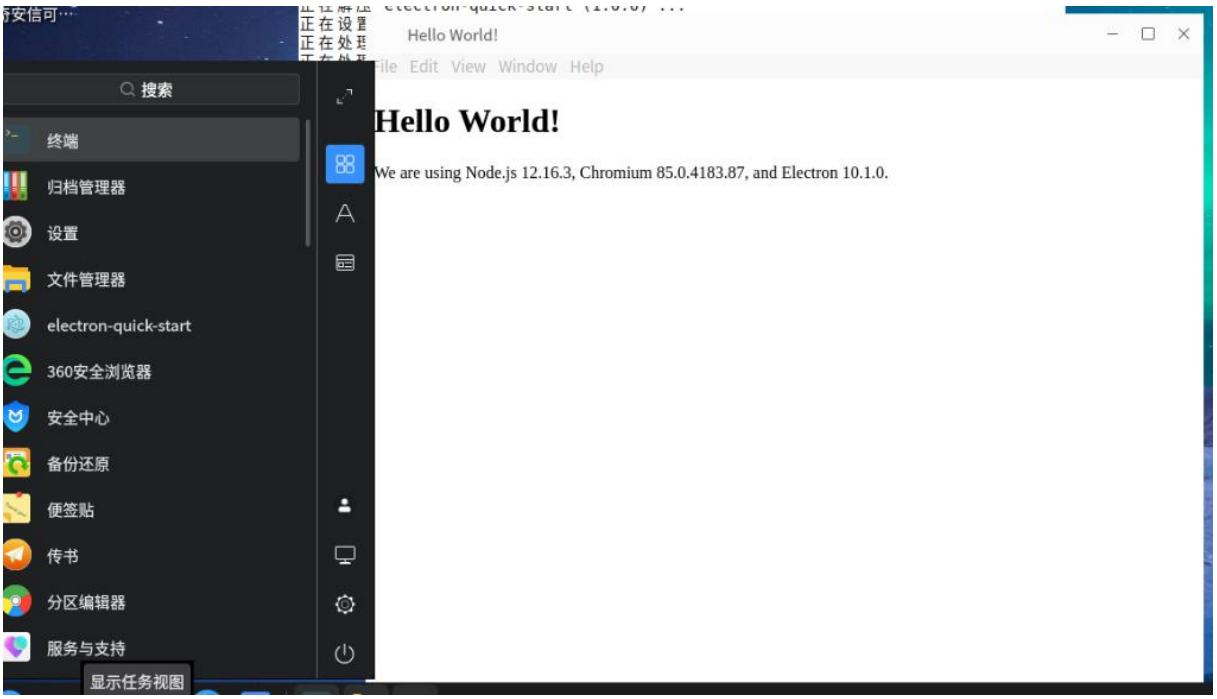
4.6. 验包

4.6.1. 安装

在 MIPS64 架构机器上，双击或在终端执行 sudo dpkg -i ***.deb 来安装 deb 包

4.6.2. 启动

从开始菜单，找到 electron-quick-start 点击启动，如下图，即表示打包成功



5. LoongArch64 架构

LoongArch64 架构和 MIPS64 架构一样，npm 源中没有对应架构的 electron 包，所以在银河麒麟桌面操作系统 V10(SP1)LoongArch64 架构上无法通过 npm install 命令在线安装 electron；

龙芯开源社区提供了 LoongArch64 架构上适配的 electron 包，在 LoongArch64 架构上安装 electron 需要从龙芯开源社区下载 electron 离线包，然后进行离线安装，具体操作方法如下：

5.1. 安装开发基础环境

5.1.1. 安装 npm 和 node（如果已经安装请忽略）

系统默认没有预装 npm 和 node，需要从源里安装，源里的 node 版本为 v12.19.0，npm 版本为 v6.14.4。具体安装方法如下：

```
$ sudo apt update
$ sudo apt install npm -y
```

安装完成后使用下列命令查看 node 和 npm 是否安装成功

```
kylin@kylin-PC:~$ node -v
v12.19.0
kylin@kylin-PC:~$ npm -v
6.14.4
```

5.1.2. 从源中安装 git（如果已经安装请忽略）

```
$ sudo apt install git -y
```

5.1.3. 设置 npm 源（可选）

如果，当前网络不能有效的安装 electron，建议将 npm 的镜像源地址修改为国内源

```
$ npm config set registry https://registry.npm.taobao.org
```

5.2. 项目开发

此处以 electron-quick-start 项目使用 electron-v12.0.9 开发为例：

5.2.1. 下载 electron-quick-start 项目

```
$ git clone https://github.com/electron/electron-quick-start
```

```
$ cd electron-quick-start
$ git checkout remotes/origin/12-x-y
```

5.2.2. 离线安装 electron 及其依赖包

a. 下载 electron 离线包

从龙芯开源社区下载 electron 离线包，并重命名

```
$ wget http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-LoongArch/v12.0.9/electron-v12.0.9-linux-loong64.zip
$ mv electron-v12.0.9-linux-loong64.zip electron-v12.0.9-linux-loongarch64.zip
```

注：

使用命令查询当前系统的架构，将 zip 包中的架构名称修改成对应的架构名称，此处为 loongarch64

```
$ archdetect
loongarch64/generic
```

下载链接：

```
electron-v8.5.5:
http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-LoongArch/v8.5.5/electron-v8.5.5-linux-loong64.zip
electron-v12.0.9:
http://ftp.loongnix.cn/os/loongnix/1.0/electron/electron-LoongArch/v12.0.9/electron-v12.0.9-linux-loong64.zip
```

b. 安装依赖的模块

```
$ cd electron-quick-start
$ npm install electron@12.0.9
```

在出现> node install.js 使用 ctrl+c 停止，如下图：

```
kylin@kylin-PC:~/electron-quick-start$ npm install electron@12.0.9

> electron@12.0.9 postinstall /home/kylin/electron-quick-start/node_modules/elec
tron
> node install.js

^C
kylin@kylin-PC:~/electron-quick-start$
```

c. 更改 install.js

```
$ vim node_modules/electron/install.js
```

找到下载 electron 的地方，如下

```
// downloads if not cached
downloadArtifact({
  version,
  artifactName: 'electron',
  force: process.env.force_no_cache === 'true',
  cacheRoot: process.env.electron_config_cache,
  platform: process.env.npm_config_platform || process.platform,
  arch: process.env.npm_config_arch || process.arch
}).then(extractFile).catch(err => {
```

```
  console.error(err.stack);
  process.exit(1);
});
```

将上述代码注释掉，添加如下代码（根据 extractFile 函数的实现，填写步骤 a 中制作好的 zip 包包名）

```
// downloads if not cached
extractFile('electron-v12.0.9-linux-loongarch64.zip');
//downloadArtifact({
//   version,
//   artifactName: 'electron',
//   force: process.env.force_no_cache === 'true',
//   cacheRoot: process.env.electron_config_cache,
//   platform: process.env.npm_config_platform || process.platform,
//   arch: process.env.npm_config_arch || process.arch
//}).then(extractFile).catch(err => {
//   console.error(err.stack);
//   process.exit(1);
//});
```

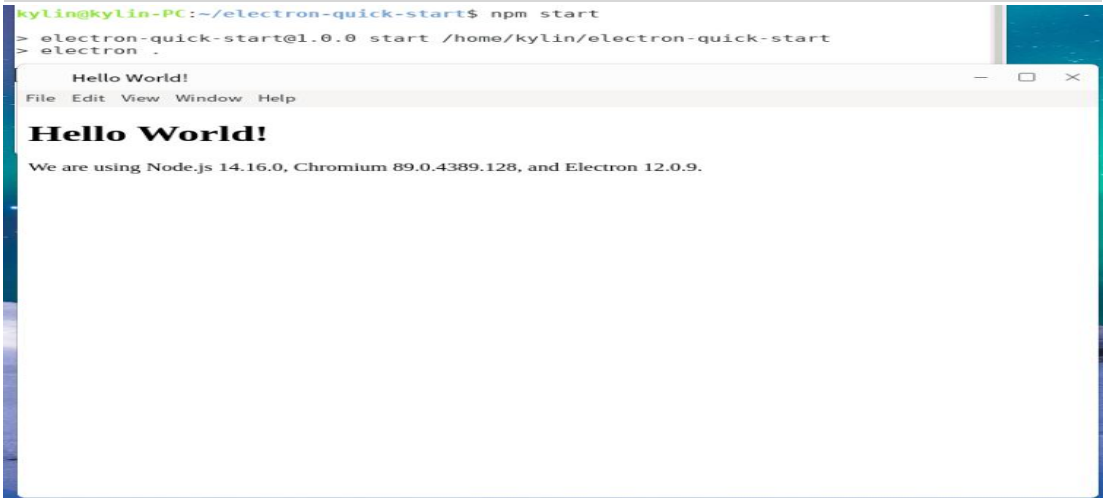
d. 安装 electron

将步骤 a 中重命名的 zip 包放到 electron-quick-start/node_modules/electron 目录下，执行安装命令

```
$ cp ~/electron-v12.0.9-linux-loongarch64.zip node_modules/electron/
$ cd node_modules/electron/
$ node install.js
```

5.3. 启动项目

```
$ cd ~/electron-quick-start
$ npm start
```



5.4. 打包

项目调试完成后，需要打包成符合银河麒麟桌面操作系统 V10 软件商店上架规范的 deb 包，建议使用 electron-packager + electron-installer-debian 工具进行打包，此处以使用 electron-packager 和 electron-installer-debian 打包为例。

5.4.1. 安装 electron-packager

安装 electron-packager 的时候会读取 package-lock.json 中 electron 的版本，但文件中使用的是 10.1.5 版本，此处使用的是 10.1.0 版本，所以需要做一些修改，不然执行 npm install electron-packager 的时候会重新覆盖安装 electron。

a. 修改 package.json 中 electron 版本为当前使用版本

```
$ cd ~/electron-quick-start
$ vim package.json
```

找到如下代码

```
"devDependencies": {
    "electron": "^10.1.5"
  }
```

修改为：

```
"devDependencies": {
    "electron": "^10.1.0"
  }
```

此处还需要删除 package-lock.json,不然安装 electron-packager 的时候还会在线安装 10.1.5 的版本

```
$ rm -rf package-lock.json
```

b. 在线安装 electron-packager

```
$ npm install electron-packager --save-dev
```

c. 修改 electron-packager 源码以支持 loongarch 架构

```
$ vim node_modules/electron-packager/src/targets.js
```

找到如下代码

```
const officialArchs = ['ia32', 'x64', 'armv7l', 'arm64', 'mips64el']
const officialPlatforms = ['darwin', 'linux', 'mas', 'win32']
const officialPlatformArchCombos = {
  darwin: ['x64', 'arm64'],
  linux: ['ia32', 'x64', 'armv7l', 'arm64', 'mips64el'],
  mas: ['x64', 'arm64'],
  win32: ['ia32', 'x64', 'arm64']
}

const buildVersions = {
```

```
  darwin: {
    arm64: '>= 11.0.0-beta.1'
  },
  linux: {
    arm64: '>= 1.8.0',
    mips64el: '^1.8.2-beta.5'
  },
```

添加 loongarch64 架构信息

```
const officialArchs = ['ia32', 'x64', 'armv7l', 'arm64', 'mips64el', 'loongarch64']
const officialPlatforms = ['darwin', 'linux', 'mas', 'win32']
const officialPlatformArchCombos = {
  darwin: ['x64', 'arm64'],
  linux: ['ia32', 'x64', 'armv7l', 'arm64', 'mips64el', 'loongarch64'],
  mas: ['x64', 'arm64'],
  win32: ['ia32', 'x64', 'arm64']
}

const buildVersions = {
  darwin: {
    arm64: '>= 11.0.0-beta.1'
  },
  linux: {
    arm64: '>= 1.8.0',
    mips64el: '^1.8.2-beta.5',
    loongarch64: '^12.0.9'
```

5.4.2. 添加应用图标

由于 electron-quick-start 项目中没有自带图标，这里我们需要自己添加一下图标目录及相应尺寸的图标。

```
$ mkdir -p icons
```

添加图标文件

```
$ ls icons
128x128.png   16x16.png   256x256.png   32x32.png   512x512.png   64x64.png
```

另外，某些应用打包后，在系统安装启动后，任务栏图标显示异常，此时可以通过修改 main.js 中的代码来规避此问题。具体操作方法如下：

```
$ vim main.js
```

找到 main.js 中的 createWindow 函数，添加 icon

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
```

```
        preload: path.join(__dirname, 'preload.js')
    }
  })
```

添加 icon，修改后如下：

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    icon: path.join(__dirname, 'icons/512x512.png'),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
```

注：注意此处图标的写法，如果图标位置不对有可能会造成打包后报：“段错误，核心已转储”，图标需要使用 512x512 以上的 png 格式；

5.4.3. 打 unpack 包

a. 添加打包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron ."
  },
```

添加打包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "packager": "electron-packager . --overwrite --platform=linux --arch=loongarch64 --out=dist/    --app-version=12.0.9
--electron-zip-dir=/home/kylin/"
  },
```

注：

--out=dist/指的是生成 unpack 包目录

--app-version=12.0.9 指的是使用的 electron 版本号

--electron-zip-dir=/home/kylin/指的是 electron zip 离线包放置目录

b. 打 unpack 包

将 electron-quick-start 打包

```
$ cd ~/electron-quick-start
$ npm run packager
```

打包完成后会有如下提示：

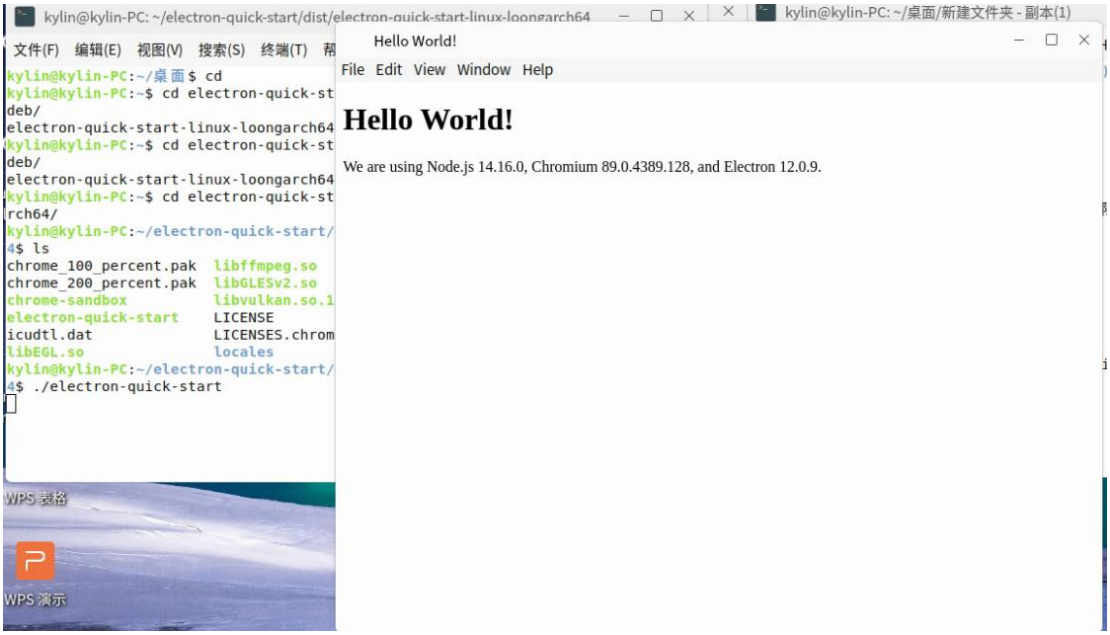
```
kylin@kylin-PC:~/electron-quick-start$ npm run packager
```

```
> electron-quick-start@1.0.0 packager /home/kylin/electron-quick-start
> electron-packager . --overwrite --platform=linux --arch=loongarch64 --out=dist/    --app-version=12.0.9
--electron-zip-dir=/home/kylin/

Packaging app for platform linux loongarch64 using electron v12.0.9
Wrote new app to dist/electron-quick-start-linux-loongarch64
```

打包完成，可以看到当前目录下生成了一个 dist/electron-quick-start-linux-loongarch64 目录，进入会后执行 electron-quick-start 二进制程序即可出现 hello，world 页面。

```
$ cd dist/electron-quick-start-linux-mips64el
$ ./electron-quick-start
```



5.4.4. 打 deb 包

上面使用 electron-packager 只是打成了 unpack 包（二进制包），要想打成符合麒麟软件上架要求的 deb 包，还需要使用 electron-installer-debian 打成 deb 包，然后参考开发者指南或打包规范进一步修改打成 deb 包

a. 安装 electron-installer-debian

```
$ cd ~/electron-quick-start
$ npm install electron-installer-debian --save-dev
```

b. 添加打 deb 包脚本

```
$vim config.json
```

添加如下代码

```
{
  "src": "dist/electron-quick-start-linux-loongarch64/",
  "dest": "dist/deb/",
  "arch": "loongarch64",
```

```
  "icon": {
    "16x16": "icons/16x16.png",
    "32x32": "icons/32x32.png",
    "64x64": "icons/64x64.png",
    "128x128": "icons/128x128.png",
    "256x256": "icons/256x256.png",
    "512x512": "icons/512x512.png"
  },
  "categories": [
    "Utility"
  ],
  "lintianOverrides": [
    "changelog-file-missing-in-native-package"
  ]
}
```

注：

src 指的是 unpack 包的目录

dest 指的是生成 deb 包的目录

c. 添加打 deb 包命令

```
$ vim package.json
```

找到如下代码

```
"scripts": {
    "start": "electron .",
    "packager": "electron-packager . --overwrite --platform=linux --arch=loongarch64 --out=dist/   --app-version=12.0.9
--electron-zip-dir=/home/kylin/"
  },
```

添加打 deb 包命令，添加后如下

```
"scripts": {
    "start": "electron .",
    "packager": "electron-packager . --overwrite --platform=linux --arch=loongarch64 --out=dist/   --app-version=12.0.9
--electron-zip-dir=/home/kylin/",
    "deb": "electron-installer-debian --config config.json"
  },
```

注:

--config config.json 指的是使用 config.json 脚本中的参数来进行打包

如果不想使用脚本也可以直接使用参数打包，例如：

```
"deb": "electron-installer-debian --src dist/electron-quick-start-linux-loongarch64 --dest dist/deb/ --arch loongarch64"
```

d. 打 deb 包

```
$ npm run deb
```

打包完成后会有如下提示

```
kylin@kylin-PC:~/electron-quick-start$ npm run deb
```

```
> electron-quick-start@1.0.0 deb /home/kylin/electron-quick-start
> electron-installer-debian --src dist/electron-quick-start-linux-loongarch64 --dest dist/deb/ --arch loongarch64


Creating package (this may take a while)
Successfully created package at dist/deb/
```

5.4.5. 修改 deb 包

使用如下命令将打好的 deb 包解包

```
$ fakeroot dpkg-deb -R electron-quick-start_1.0.0_loongarch64.deb electron-quick-start_1.0.0_loongarch64
```

按照打包规范对 deb 包进行调试

然后，使用如下命令重新打包

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_loongarch64 .
```

注：

使用 dpkg -b 打包时不写打包名称会按照 control 文件自动进行命名打包，会将原包覆盖，可以使用

```
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_loongarch64 electron-quick-start_1.0.0_loongarch64_new.deb
```

命令自定义新包名称来进行打包

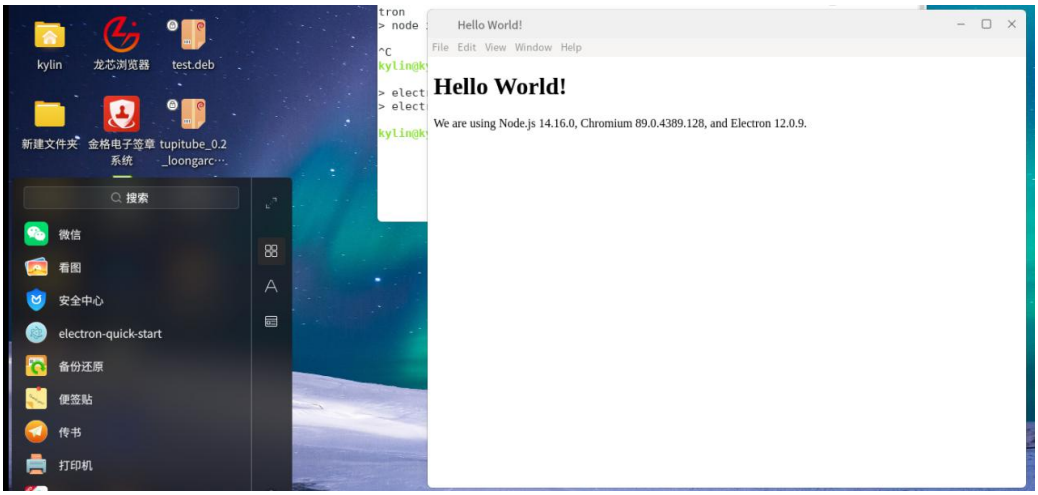
5.5. 验包

5.5.1. 安装

在 loongarch64 架构机器上，双击或在终端执行 sudo dpkg -i ***.deb 来安装 deb 包

5.5.2. 启动

从开始菜单，找到 electron-quick-start 点击启动，如下图，即表示打包成功



如上图所示，即表示打包成功。

6. 附录

6.1. MIPS64 架构下编译 electron-builder

6.1.1. 下载 electron-builder 源码

此处我们以 electron-builder@v21.2.0 版本为例
源码路径：https://github.com/electron-userland/electron-builder/archive/v21.2.0.zip

```
$ wget https://github.com/electron-userland/electron-builder/archive/refs/tags/v22.1.0.zip
$ ls
v21.2.0.zip
$ unzip v21.2.0.zip
$ ls
electron-builder-21.2.0    v21.2.0.zip
$ cd electron-builder-21.2.0
$ ls
appveyor.yml        CHANGELOG.md        docker      mkdocs.yml       package.json    README.md    test        yarn.lock
babel.config.js    CONTRIBUTING.md    LICENSE    netlify.toml    packages          scripts       typings
```

6.1.2. 修改 electron-builder 源码

a. **修改文件路径 packages/app-builder-lib/src**

① **vim packages/app-builder-lib/src/linuxPackager.ts**

```
115 export function toAppImageOrSnapArch(arch: Arch): string {
116    switch (arch) {
117       case Arch.x64:
118          return "X86_64_64"
119       case Arch.ia32:
120          return "i386"
121       case Arch.armv7l:
122          return "arm"
123       case Arch.arm64:
124          return "arm_aarch64"
125       case Arch.mips64el:
126           return "mips64el"
```

② **vim packages/app-builder-lib/src/targets/MsiTarget.ts**

```
76      // noinspection SpellCheckingInspection
```

```
77        const candleArgs = [
78            "-arch", arch === Arch.ia32 ? "X86_64" : (arch === Arch.armv7l ? "arm" : (arch === Arch.mips64       el ? "mips64el" :
"X64")),
79            `-dappDir=${vm.toVmFile(appOutDir)}`,
80        ].concat(this.getCommonWixArgs())
```

b. 修改文件路径 packages/electron-builder/src

① vim packages/electron-builder/src/builder.ts

```
17 export interface CliOptions extends PackagerOptions, PublishOptions {
18    x64?: boolean
19    ia32?: boolean
20    armv7l?: boolean
21    arm64?: boolean
22    mips64el?: boolean
23
24    dir?: boolean
25 }
```

```
35    function processTargets(platform: Platform, types: Array<string>) {
36        function commonArch(currentIfNotSpecified: boolean): Array<Arch> {
37            if (platform === Platform.MAC) {
38                return args.x64 || currentIfNotSpecified ? [Arch.x64] : []
39            }
40
41            const result = Array<Arch>()
42            if (args.x64) {
43                result.push(Arch.x64)
44            }
45            if (args.armv7l) {
46                result.push(Arch.armv7l)
47            }
48            if (args.arm64) {
49                result.push(Arch.arm64)
50            }
51            if (args.mips64el) {
52                result.push(Arch.mips64el)
53            }
54            if (args.ia32) {
55                result.push(Arch.ia32)
56            }
```

```
127    delete result.ia32
128    delete result.x64
129    delete result.armv7l
```

```
delete result.arm64
delete result.mips64el
-------------------------------------------------------------
    .option("arm64", {
      group: buildGroup,
      description: "Build for arm64",
      type: "boolean",
    })
    .option("mips64el", {
      group: buildGroup,
      description: "Build for mips64el",
      type: "boolean",
    })
```

② **vim packages/electron-builder/src/cli/install-app-deps.ts**

```
    .option("arch", {
      choices: getArchCliNames().concat("all"),
      default: process.arch === "arm" ? "armv7l" : (process.arch === "mips64el" ? "mips64el" : pr      ocess.arch),
      description: "The target arch",
    })
}
```

c. 修改文件路径 **packages/builder-util/src**

① **vim packages/builder-util/src/arch.ts**

```
export enum Arch {
  ia32, x64, armv7l, arm64, mips64el
}

export type ArchType = "x64" | "ia32" | "armv7l" | "arm64" | "mips64el"

export function toLinuxArchString(arch: Arch, targetName: string): string {
  switch (arch) {
    case Arch.x64:
      return "amd64"
    case Arch.ia32:
      return targetName === "pacman" ? "i686" : "i386"
    case Arch.armv7l:
      return targetName === "snap" || targetName === "deb" ? "armhf" : "armv7l"
    case Arch.arm64:
      return "arm64"
    case Arch.mips64el:
      return "mips64el"

    default:
```

```
21            throw new Error(`Unsupported arch ${arch}`)
22      }
23 }
-----------------------------------------------------
25 export function getArchCliNames(): Array<string> {
26      return [Arch[Arch.ia32], Arch[Arch.x64], Arch[Arch.armv7l], Arch[Arch.arm64], Arch[Arch.mips64e        l]]
27 }
-----------------------------------------------------
33 export function archFromString(name: string): Arch {
34      switch (name) {
35          case "x64":
36              return Arch.x64
37          case "ia32":
38              return Arch.ia32
39          case "arm64":
40              return Arch.arm64
41          case "armv7l":
42              return Arch.armv7l
43          case "mips64el":
44              return Arch.mips64el
45
46          default:
47              throw new Error(`Unsupported arch ${name}`)
48      }
49 }
```

6.1.3. 重新编译 electron-builder

a. 安装依赖

```
$ npm install
$ npm install compile-run
$ npm install ts-babel
$ npm install yarn
$ npm install dts-gen
$ ./node_modules/.bin/yarn
yarn install v1.22.18
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear this warning, remove package-lock.json.
[1/5] Resolving packages...
[2/5] Fetching packages...
[3/5] Linking dependencies...
[4/5] Building fresh packages...
[5/5] Cleaning modules...
```

```
success Saved lockfile.
Done in 29.87s.
```

注：

如果执行./node_modules/.bin/yarn 报错

```
$ ./node_modules/.bin/yarn
yarn install v1.22.18
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear this warning, remove package-lock.json.
[1/5] Resolving packages...
[2/5] Fetching packages...
error An unexpected error occurred: "https://registry.npmmirror.com/typescript/-/typescript-3.9.10.tgz: unexpected end of file".
info If you think this is a bug, please open a bug report with the information provided in "/home/kylin/electron-builder/electron-builder-21.2.0/yarn-error.log".
info Visit https://yarnpkg.com/en/docs/cli/install for documentation about this command.
```

可以使用参考如下方法解决：

```
#关闭代理
npm config set proxy null
#设置淘宝镜像
npm config set registry https://registry.npm.taobao.org
#测试一下
npm info underscore
```

b. **编译**

```
$ npm run compile

> @ compile /home/kylin/electron-builder/electron-builder-21.2.0
> ts-babel "packages/*" test

Building builder-util-runtime
Building builder-util, electron-updater
Building electron-publish
Building app-builder-lib
Building dmg-builder, electron-builder-squirrel-windows
Building electron-builder
Building test
```

如上图所示，即表示编译完成，此时我们就得到了支持 mips64el 架构的 electron-builder 包。在 MIPS 架构使用 electron-builder 打包时我们会用到编译成功中的某些文件。

6.2. MIPS64 架构下编译 app-builder

6.2.1. 安装配置 go 环境

此处我们使用的是 go1.17.7 版本，低版本的 go 不支持设置-w 参数，不能设置代理，执行 go get 的时候会报错，

下载链接：https://pan.baidu.com/s/1LUFXQltqeEXf2c7awIm4gA?pwd=mm24

```
#解压
$ tar xvf go-linux-mips64le-bootstrap.tbz
#将解压后的文件拷贝到/usr/local 目录
$ sudo mv go-linux-mips64le-bootstrap /usr/local/go1.17.7
#添加环境变量
$ vim ~/.bashrc
export GOROOT=/usr/local/go1.17.7
export PATH=$GOROOT/bin:$PATH
$ source ~/.bashrc
$ go version
go version go1.17.7 linux/mips64le
```

设置 go 代理

```
$ go env -w GOPROXY=https://goproxy.cn,direct
```

安装 go-bindata

```
$ sudo apt install go-bindata
```

6.2.2. 下载 app-builder 源码

```
$ git clone https://github.com/develar/app-builder
```

6.2.3. 编译 app-builder

```
$ cd app-builder
$ go get
go: downloading github.com/alecthomas/kingpin v2.2.6+incompatible
go: downloading github.com/segmentio/ksuid v1.0.4
go: downloading github.com/develar/errors v0.9.0
go: downloading github.com/develar/go-pkcs12 v0.0.0-20181115143544-54baa4f32c6a
go: downloading github.com/json-iterator/go v1.1.12
go: downloading go.uber.org/zap v1.21.0
go: downloading github.com/develar/go-fs-util v0.0.0-20190620175131-69a2d4542206
go: downloading github.com/oxtoacart/bpool v0.0.0-20190530202638-03653db5a59c
go: downloading github.com/aclements/go-rabin v0.0.0-20170911142644-d0b643ea1a4c
go: downloading github.com/minio/blake2b-simd v0.0.0-20160723061019-3f5f724cb5b1
go: downloading github.com/phayes/permbits v0.0.0-20190612203442-39d7c581d2ee
go: downloading github.com/dustin/go-humanize v1.0.0
go: downloading github.com/mitchellh/go-homedir v1.1.0
go: downloading github.com/mattn/go-colorable v0.1.12
go: downloading github.com/mattn/go-isatty v0.0.14
```

```
go: downloading github.com/biessek/golang-ico v0.0.0-20180326222316-d348d9ea4670
go: downloading github.com/disintegration/imaging v1.6.2
go: downloading github.com/pkg/xattr v0.4.6
go: downloading github.com/pkg/errors v0.9.1
go: downloading howett.net/plist v1.0.0
go: downloading github.com/aws/aws-sdk-go v1.43.14
go: downloading github.com/mcuadros/go-version v0.0.0-20190830083331-035f6764e8d2
go: downloading gopkg.in/alessio/shellescape.v1 v1.0.0-20170105083845-52074bc9df61
go: downloading github.com/zieckey/goini v0.0.0-20180118150432-0da17d361d26
go: downloading github.com/alecthomas/template v0.0.0-20190718012654-fb15b899a751
go: downloading github.com/alecthomas/units v0.0.0-20211218093645-b94a6e3cc137
go: downloading github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd
go: downloading github.com/modern-go/reflect2 v1.0.2
go: downloading go.uber.org/multierr v1.8.0
go: downloading go.uber.org/atomic v1.9.0
go: downloading golang.org/x/sys v0.0.0-20220307203707-22a9840ba4d7
go: downloading github.com/jsummers/gobmp v0.0.0-20151104160322-e2ba15ffa76e
go: downloading golang.org/x/image v0.0.0-20220302094943-723b81ca9867
go: downloading github.com/jmespath/go-jmespath v0.4.0
$ make
```

make 完成后会在 dist/linux_amd64 目录下生成 app-builder,这里需要注意 app-builder 的构建脚本把构建产物放在了 dist/linux_amd64 目录下，但是实际上新构建出来的文件并不是 amd64 架构的，这个文件最终需要覆盖到目标 electron 工程里 node_modules/app-builder-bin/linux/mips64el/app-builder。