



银河麒麟桌面操作系统 V10 Electron 应用开发者打包指南

麒麟软件有限公司

生态发展中心

2025 年 12 月 17 日

版本说明

版本号	版本说明	作者	日期	变更内容
V1.0	首次发布	吴兆惠	2022-09-04	首次创建
V1.1	内容变更	吴兆惠	2022-09-07	增加 mips 架构下使用 electron-builder 工具打包的方法
V1.2	模板变更	刘佳鑫	2022-11-30	模板变更
V2.0	内容变更	吴兆惠	2023-01-10	修改龙芯架构 electron 安装方法、新增本地 electron 镜像源安装方法、简化章节、修改 fpm 安装方法等。
V3.0	内容变更	吴兆惠	2023-08-18	改用 nvm 来安装管理 Nodejs、添加 nvm 离线安装 Nodejs 方法、修改龙芯 Mips64el 和 Loongarch64 架构可用的 Nodejs 版本和 Electron 版本、

				V10 系统使用 gem 安装 fpm 报错解决方法、优化目录结构等。
V3.1	内容变更	吴兆惠	2025-10-29	更新 npm 镜像源链接、更新设置 Electron 镜像源方法、更新文档段落描述等。
V4.0	模板变更	吴兆惠	2025-12-17	模板变更

目录

1 概述	1
2 X86_64 架构	2
2.1 安装开发基础环境	2
2.1.1 安装 Nodejs	2
2.1.2 从源中安装 git	3
2.1.3 设置 npm 仓库源(可选)	3
2.1.4 设置 electron 镜像源(可选)	4
2.2 项目开发	4
2.2.1 下载 electron-quick-start 项目	4
2.2.2 修改 electron 版本	4
2.2.3 安装 electron 及依赖包	5
2.3 启动项目	5
2.4 打包	5
2.4.1 安装 electron-builder	5
2.4.2 添加打包命令	6
2.4.3 添加应用图标	8
2.4.4 打 deb 包	9
2.4.5 修改 deb 包	10
2.5 验包	10
2.5.1 安装	10
2.5.2 启动	10
3 ARM64 架构	12
3.1 安装开发基础环境	12
3.1.1 安装 Nodejs	12
3.1.2 从源中安装 git	12
3.1.3 设置 npm 仓库源(可选)	12
3.1.4 设置 electron 镜像源 (可选)	12
3.2 项目开发	12
3.2.1 下载 electron-quick-start 项目	12
3.2.2 修改 electron 版本	12

3.2.3 安装 electron 及依赖包	12
3.3 启动项目	12
3.4 打包	13
3.4.1 安装 electron-builder	13
3.4.2 添加打包命令	13
3.4.3 添加应用图标	13
3.4.4 安装 fpm	13
3.4.5 打 deb 包	13
3.4.6 修改 deb 包	14
3.5 验包	14
3.5.1 安装	14
3.5.2 启动	14
4 Mips64el 架构	15
4.1 安装开发基础环境	15
4.1.1 安装 Nodejs	15
4.1.2 从源中安装 git	17
4.1.3 设置 npm 仓库源(必选)	18
4.1.4 设置 electron 镜像源(必选)	18
4.2 项目开发	18
4.2.1 下载 electron-quick-start 项目	18
4.2.2 修改 electron 版本	19
4.2.3 安装 electron 及依赖包	19
4.3 启动项目	19
4.4 打包	21
4.4.1 安装 electron-builder	21
4.4.2 添加打包命令	21
4.4.3 添加应用图标	21
4.4.4 安装 fpm	21
4.4.5 打 deb 包	22
4.4.6 修改 deb 包	23
4.5 验包	23
4.5.1 安装	23

4.5.2 启动	23
5 LoongArch64 架构	25
5.1 安装开发基础环境	25
5.1.1 安装 Nodejs	25
5.1.2 从源中安装 git	26
5.1.3 设置 npm 仓库源(必选)	26
5.1.4 设置 electron 镜像源(必选)	26
5.2 项目开发	26
5.2.1 下载 electron-quick-start 项目	26
5.2.2 修改 electron 版本	26
5.2.3 安装 electron 及依赖包	27
5.3 启动项目	27
5.4 打包	27
5.4.1 安装 electron-builder	27
5.4.2 添加打包命令	28
5.4.3 添加应用图标	28
5.4.4 安装 fpm	28
5.4.5 打 deb 包	28
5.4.6 修改 deb 包	28
5.5 验包	29
5.5.1 安装	29
5.5.2 启动	29
6 附录	30
6.1 搭建 electron 本地镜像源	30
6.1.1 创建 electron 镜像源目录	30
6.1.2 将 electron 包放入上述目录	30
6.1.3 启动 electron 镜像源	30
6.1.4 设置 electron 镜像源(必选)	31
7 获取帮助	31

1 概述

本指南使用 Electron 官方提供的简单模板项目 `electron-quick-start`，演示在银河麒麟桌面操作系统各个架构开发 Electron 应用时，安装开发基础环境、仓库源设置、Electron 及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。

本文中提到的银河麒麟桌面操作系统 V10 包括 V10-4.4 内核、V10(SP1)-内核 5.4 和 V10(SP1-HWE)-5.10 内核，下文中的 V10 系统指的是老 V10 系统，V10(SP1)系统指的是 V10(SP1)和 V10(SP1-HWE)。

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。

nvm 是 Node.js 的版本管理器，可以简单操作 Nodejs 版本的切换、安装、查看等功能，可以满足一个电脑中安装多个 Nodejs 版本，当我们想使用哪个版本就切换成哪个版本，而 nvm 则是提供切换 Nodejs 版本的工具。

注：

npm 官方网站：<https://www.npmjs.com/>

npm 官方源：<https://registry.npmjs.org/>

npm 淘宝源：<https://registry.npmmirror.com/>

2 X86_64 架构

2.1 安装开发基础环境

2.1.1 安装 Nodejs

麒麟系统的多数版本默认未预装 Node.js，需从系统源中安装。

不同版本源提供的 Node.js 版本存在差异：

- V10 源中，Node.js 版本为 v4.2.6，配套 npm 为 v3.5.2；
- V10 (SP1) 源中，Node.js 版本升级至 v10.19.0，npm 为 v6.14.4。

由于实际开发中常需更高版本的 Node.js（如支持 ES 新特性或适配配合第三方依赖），可通过以下方式安装高版本：

1、二进制包解压：直接下载对应架构的 Node.js 二进制 tar 包，解压后手动配置环境变量；

2、n 模块管理：先从系统源安装基础 npm，再通过 npm 安装 n 模块（Node.js 版本管理工具）；

3、nvm 工具管理：使用 Node Version Manager (nvm) 灵活切换多个 Node.js 版本（推荐，操作更便捷）。

以下以 nvm 安装 Node.js v16.17.1 为例，介绍具体步骤：

a. 安装 nvm

在终端执行如下命令安装 nvm

```
$ sudo apt install nvm
输入密码
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列【新】软件包将被安装：
  nvm
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 122 个软件包未被升级。
需要下载 747 kB 的归档。
解压缩后会消耗 0 B 的额外空间。
获取:1 http://archive2.kylinos.cn/deb/kylin/production/PART-V10-SP1/custom/partner/V10-SP1/default/all amd64
nvm amd64 0.40.0 [747 kB]
已下载 747 kB，耗时 0 秒 (2,733 kB/s)
正在选中未选择的软件包 nvm。
(正在读取数据库 ... 系统当前共安装有 215307 个文件和目录。)
准备解压 .../archives/nvm_0.40.0_amd64.deb ...
```

```
正在解压 nvm (0.40.0) ...
正在设置 nvm (0.40.0) ...
####配置环境变量####
kylin
####环境变量配置完毕####
####在当前终端执行 source ~/.bashrc 或重新开启一个终端使配置生效! #####
```

安装完成后，需要重新打开一个终端，然后使用 `nvm -v` 查看是否安装成功。

```
$ nvm -v
0.40.0
```

如上表示 `nvm` 模块安装成功，下面就可以使用 `nvm` 来安装高版本 Nodejs。

b. 安装 Nodejs

执行 `nvm install 16.17.1` 进行 Nodejs 在线安装。

```
$ nvm install 16.17.1
Downloading and installing node v16.17.1...
Downloading https://nodejs.org/dist/v16.17.1/node-v16.17.1-linux-x64.tar.xz...
#####
##### 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v16.17.1 (npm v8.15.0)
Creating default alias: default -> 16.17.1 (-> v16.17.1)
```

c. 验证 Nodejs 安装的版本

```
$ node -v
v16.17.1
$ npm -v
8.15.0
```

2.1.2 从源中安装 git

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

2.1.3 设置 npm 仓库源(可选)

npm 原始的源(<https://registry.npmjs.org>)在国外服务器上，拉取依赖相对较慢，建议将 npm 的仓库源修改为国内淘宝源，配置方法如下：

npm 的默认仓库源 (<https://registry.npmjs.org>) 部署在国外服务器，国内用户在拉取依赖时可能因网络延迟问题导致速度较慢，建议将仓库源切换为国内淘宝镜像（已更名为「npmirror 镜像站」）以提升下载效率。配置方法如下：

```
# 设置 npm 仓库源为淘宝镜像（当前配置立即生效）
$ npm config set registry https://registry.npmirror.com/
# 验证配置是否成功
$ npm config get registry
```

```
# 若输出 https://registry.npmmirror.com/, 则表示配置生效
```

2.1.4 设置 electron 镜像源(可选)

在项目依赖安装过程中, electron 模块会通过 @electron/get (原 electron-download, 现已整合到该包) 下载 Electron 的预编译二进制文件。若未切换 npm 仓库源, 默认会从 GitHub Releases 页面拉取, 受网络环境影响可能速度较慢。建议将 Electron 镜像源修改为国内淘宝镜像, 配置方法如下:

```
# 临时生效 (当前终端会话)
$ export ELECTRON_MIRROR="https://npmmirror.com/mirrors/electron/"
# 永久生效 (需根据 shell 类型配置, 以 bash 为例)
$ echo 'export ELECTRON_MIRROR="https://npmmirror.com/mirrors/electron/"' >> ~/.bashrc
source ~/.bashrc
```

2.2 项目开发

此处以 electron-quick-start 项目使用 electron-v20.0.3 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
#删除 lock 文件
$ rm -rf package-lock.json
```

2.2.2 修改 electron 版本

编辑 package.json 文件, 将 electron 版本修改为 20.0.3, 如下所示:

```
$ cat package.json
{
  "name": "electron-quick-start",
  "version": "1.0.0",
  "description": "A minimal Electron application",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "repository": "https://github.com/electron/electron-quick-start",
  "keywords": [
    "Electron",
    "quick",
    "start",
    "tutorial",
    "demo"
  ],
}
```

```
"author": "GitHub",
"license": "CC0-1.0",
"devDependencies": {
  "electron": "20.0.3"
}
}
```

2.2.3 安装 electron 及依赖包

执行 `npm install` 命令来安装 electron 及依赖包，安装完成后会有如下提示，表示 electron 及依赖包安装成功。

```
$ npm install
added 92 packages in 8m
```

2.3 启动项目

执行 `npm start` 命令来启动项目。



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，需打包为符合《银河麒麟桌面操作系统 V10-DEB 包打包规范》的 deb 格式安装包。针对 Electron 项目，主流打包方案包括：

- electron-builder（一站式打包工具，支持直接生成 deb 包，推荐）；
- 组合工具链：通过 electron-packager 生成应用包，再配合 electron-installer-debian 转换为 deb 格式。

以下以 electron-builder v21.2.0 为例，演示具体打包流程：

2.4.1 安装 electron-builder

执行以下命令安装 electron-builder：

```

$ 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$

```

需注意：最新版本的 electron-builder 对 Node.js 和 npm 的版本有特定要求（通常需较新版本支持），安装过程中可能会出现版本兼容性告警，若当前环境满足项目运行需求，可忽略此类告警。

2.4.2 添加打包命令

以下是 electron-builder 打包时的核心配置项说明。

需重点关注 必填项 与 麒麟软件商店适配要求：

配置项	说明	关键要求
productName	项目名称，同时作为生成的 deb 包文件名（如 productName_1.0.0_amd64.deb）	需符合麒麟软件商店命名规范，避免特殊字符
appId	应用唯一标识（如 com.company.appname）	建议遵循“反向域名”格式，确保与其他应用不重复
directories.output	deb 包的生成路径（如 ./dist/deb）	建议指定明确目录，便于后续

配置项	说明	关键要求
linux.category	应用分类，需匹配麒麟软件商店的分类标准	<p>查找与上传商店</p> <p>严格参考以下分类：</p> <ul style="list-style-type: none"> - 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
linux.target	目标打包格式	需固定设为 deb（适配麒麟桌面系统）
linux.icon	应用图标路径（如 ./build/icon.png）	建议使用全尺寸的 PNG 图标，不指定则使用 Electron 默认图标
author.name	打包者姓名 / 主体名称（个人或企业）	必填项，不填写会导致打包失败
author.email	打包者联系邮箱	必填项，缺失会触发报错：× Please specify author 'email' in the application package.json

编辑 package.json 文件，添加打包相关的参数，配置示例如下所示：

```
$ cat package.json
{
  "name": "electron-quick-start",
  "version": "1.0.0",
  "description": "A minimal Electron application",
  "main": "main.js",
  "scripts": {
```

```
"start": "electron .",
"builder": "electron-builder -l deb"
},
"build": {
  "productName": "electron-quick-start",
  "asar": "false",
  "appId": "electron-quick-start",
  "directories": {
    "output": "dist"
  },
},
"linux": {
  "icon": "icons",
  "category": "Education",
  "target": "deb"
}
},
"repository": "https://github.com/electron/electron-quick-start",
"keywords": [
  "Electron",
  "quick",
  "start",
  "tutorial",
  "demo"
],
"author": {
  "name": "Wu Zhaohui",
  "email": "wuzhaohui@kylinos.cn"
},
"license": "CC0-1.0",
"devDependencies": {
  "electron": "20.0.3"
}
}
```

2.4.3 添加应用图标

由于 electron-quick-start 项目默认未包含应用图标，需手动添加图标目录及多尺寸图标文件，步骤如下：

```
# 创建图标目录（-p 确保父目录存在）
$ mkdir -p icons
# 将以下尺寸的 PNG 图标放入 icons 目录（建议尺寸完整，适配不同场景显示）
# 示例文件列表：
$ ls icons
16x16.png 32x32.png 64x64.png 128x128.png 256x256.png 512x512.png
```

部分应用打包后，在系统中安装启动时可能出现任务栏图标显示异常（如空白、默认图标），可通过修改 `main.js` 中窗口配置指定图标路径解决：

```
$ vim main.js
```

找到 `createWindow` 函数，在窗口配置中添加 `icon` 字段，指向 `512x512` 尺寸的图标（关键尺寸，确保兼容性）：

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

// 修改后（添加 icon 配置）
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；

2.4.4 打 deb 包

执行 `npm run builder` 命令进行打包，打包完成后会有如下提示，表示打包成功。

```
$ 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=20.0.3 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
```

2.4.5 修改 deb 包

当已打包的 `deb` 包存在配置问题（无需重新编译项目）时，可通过以下步骤直接修改并重新打包：

a. 解包 deb 包

使用如下命令将打好的 `deb` 包解包

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

解包后可在目标目录中修改内容（如 `DEBIAN/control`、资源文件等）。

b. 重新打包 deb 包（需要 fakeroot）

```
# 重新打包时必须用 fakeroot，否则可能导致文件权限错误（安装时可能报“权限不足”）
# 方式 1：默认命名（可能覆盖原包），按解包目录中 DEBIAN/control 的配置命名
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_amd64 .
# 方式 2：指定新包名（推荐，避免覆盖）
$ fakeroot dpkg-deb -b electron-quick-start_1.0.0_amd64 electron-quick-start_1.0.0_amd64_new.deb
```

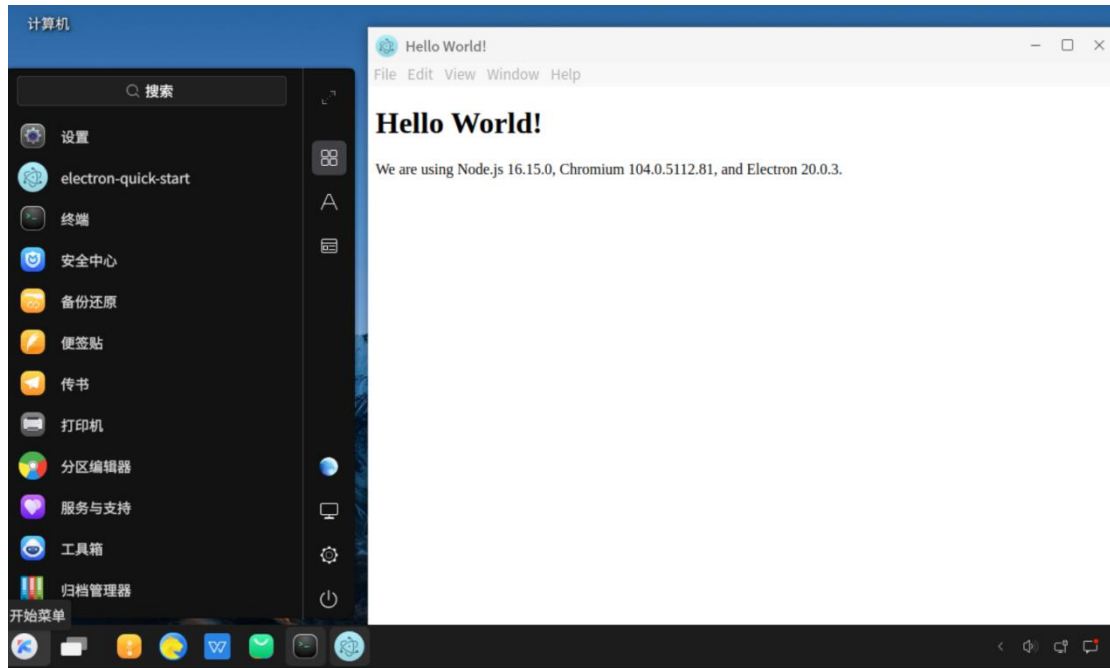
2.5 验包

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 Nodejs

[参考 2.1.1 章节。](#)

3.1.2 从源中安装 git

[参考 2.1.2 章节。](#)

3.1.3 设置 npm 仓库源(可选)

[参考 2.1.3 章节。](#)

3.1.4 设置 electron 镜像源（可选）

[参考 2.1.4 章节。](#)

3.2 项目开发

此处以 electron-quick-start 项目使用 electron-v20.0.3 开发为例：

3.2.1 下载 electron-quick-start 项目

[参考 2.2.1 章节。](#)

3.2.2 修改 electron 版本

[参考 2.2.2 章节。](#)

3.2.3 安装 electron 及依赖包

[参考 2.2.3 章节。](#)

3.3 启动项目

执行 `npm start` 命令来启动项目。



如上图所示，即表示项目运行成功。

3.4 打包

3.4.1 安装 electron-builder

[参考 2.4.1 章节。](#)

3.4.2 添加打包命令

[参考 2.4.2 章节。](#)

3.4.3 添加应用图标

[参考 2.4.3 章节。](#)

3.4.4 安装 fpm

使用 electron-builder 打包时，若涉及非 X86_64 环境，可能会因 github 仓库中 fpm 包仅提供 X86_64 架构支持而报错。此时可通过系统源安装 Ruby 并手动部署 fpm，再强制使用系统级 fpm 避免架构适配问题，具体步骤如下：

```
# 更新系统源
$ sudo apt update
# 安装 Ruby 基础环境
$ sudo apt install ruby -y
# 通过 gem 安装 fpm
$ sudo gem install fpm
# 设置环境变量，强制 electron-builder 使用系统安装的 fpm
$ export USE_SYSTEM_FPM="true"
```

注：

V10 系统默认 Ruby 版本较低，执行 `sudo gem install fpm` 时会触发如下错误

```
ERROR: Error installing fpm:
  rexml requires Ruby version >= 2.5.0.
```

下载 V10 系统适配的高版本 Ruby 安装包

链接：<https://wwpp.lanzoum.com/b03es12ba> 密码:lusn

解压后进入目录，使用 `sudo dpkg -i *.deb` 安装，安装完成后，重新执行 `sudo gem install fpm` 及后续操作。

3.4.5 打 deb 包

执行 `npm run builder` 命令进行打包，打包完成后会有如下提示，表示打包成功。

```
$ 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=20.0.3 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
```

3.4.6 修改 deb 包

[参考 2.4.5 章节](#)。

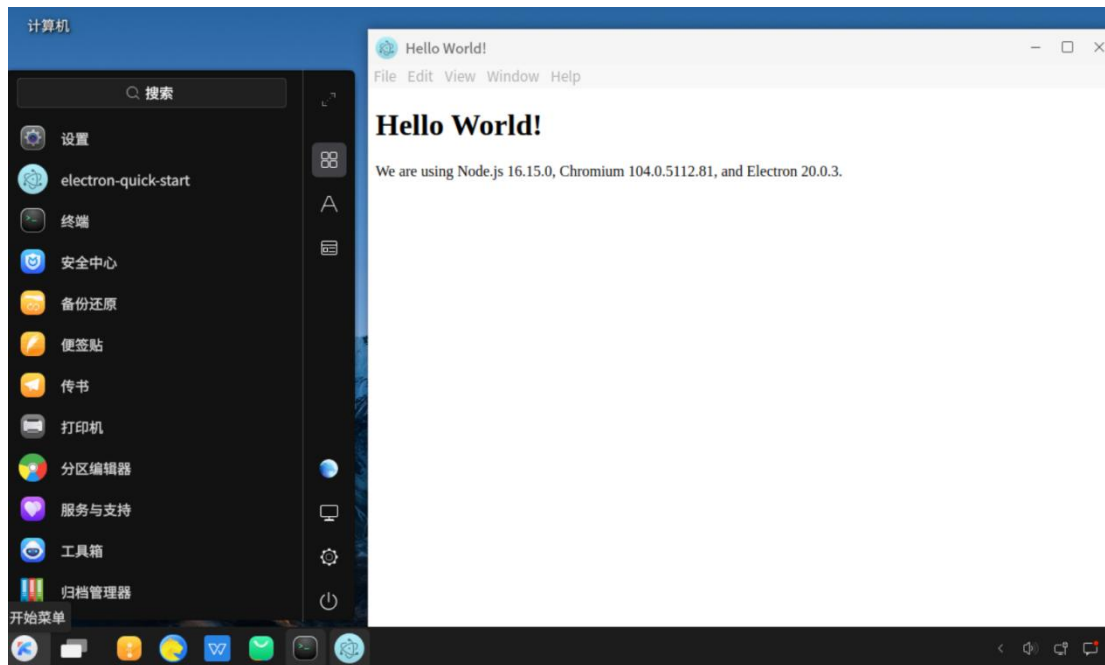
3.5 验包

3.5.1 安装

在 ARM64 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 Mips64el 架构

Mips64el 架构下的 Electron 应用打包，与 X86_64、ARM64 架构存在明显差异，核心问题集中在 Node.js 安装限制和 npm 仓库源适配两点，具体差异如下：

1、Node.js 无法通过常规工具安装

Node.js 官方未提供 Mips64el 架构的预编译二进制 tar 包，导致无法像 X86_64/ARM64 那样，通过 n 模块（Node 版本管理工具）或 nvm（Node Version Manager）在线下载并安装 Node.js，需依赖其他方式部署 Node.js 环境。

2、需切换龙芯移植的 npm 仓库源

npm 官方源中，Electron 及相关依赖包（如 electron-builder、@electron/get）未提供 Mips64el 架构版本，直接使用默认源会因“架构不匹配”导致安装失败。需将 npm 仓库源切换为龙芯移植后的专用源，才能获取适配 Mips64el 架构的依赖包，确保打包流程正常推进。

4.1 安装开发基础环境

4.1.1 安装 Nodejs

Mips64el 架构下，V10 与 V10 (SP1) 系统的 Node.js 预装情况、默认版本及高版本安装方式存在差异，且因 Node.js 官方未提供该架构的二进制包，需通过离线方式部署高版本，具体说明如下：

系统版本	预装情况	系统源提供版本（安装命令： sudo apt install nodejs npm）	已适配高版本 Node.js
V10-2107	默认预装 Node.js/npm	预装版本：Node.js 8.9.4、npm 5.6.0	v12.19.1、v16.17.1、 v18.13.0
V10（2107前）	无预装	Node.js 4.2.6、npm 3.5.2	
V10(SP1)	无预装	Node.js 10.19.0、npm 6.14.4	v12.16.1、v12.16.3、 v16.17.1、v18.13.0

高版本 Node.js 下载链接

V10 系统（含 2107 及之前版本）

链接：<https://wwpp.lanzoum.com/b03es15tg>（提取码：9u67）

V10 (SP1) 系统

链接: <https://wwpp.lanzoum.com/b03es18sd> (提取码: 8cat)

由于 Node.js 官方未提供 Mips64el 架构的二进制 tar 包, 无法通过 nvm 在线安装 (执行 nvm install 会自动下载源码编译, 但因编译参数不匹配导致失败), 因此需采用“离线安装”方式部署高版本 Node.js。

下面以使用 nvm 离线安装 Node.js v16.17.1 为例, 详细介绍具体操作步骤。

a. 安装 nvm

参考 [2.1.1-a 章节](#)。

b. 安装 openssl(仅限于 V10)

在 V10 系统 Mips64el 架构上, 由于编译高版本的 nodejs 时需要使用高版本的 openssl, 所以 nodejs 运行环境也需要高版本的 openssl 环境, 需要手动下载高版本的 openssl 离线包, 解压后进行离线安装。

在 V10 系统 Mips64el 架构中编译或运行高版本 Node.js 时, 需依赖高版本 OpenSSL (低版本 OpenSSL 可能导致编译失败或运行时依赖报错)。由于系统默认 OpenSSL 版本通常不满足需求, 需手动下载高版本 OpenSSL 离线包并完成安装。

高版本 openssl 下载链接:

<https://wwpp.lanzoum.com/b03es1i5a> 密码:5ywl

```
$ tar xvf openssl_1.1.1n-0+deb10u5kylin1_mips64el_v10.tar.gz
openssl_1.1.1n-0+deb10u5kylin1_mips64el_v10/libssl1.1_1.1.1n-0+deb10u5kylin1_mips64el.deb
openssl_1.1.1n-0+deb10u5kylin1_mips64el_v10/libssl-dev_1.1.1n-0+deb10u5kylin1_mips64el.deb
openssl_1.1.1n-0+deb10u5kylin1_mips64el_v10/libssl-doc_1.1.1n-0+deb10u5kylin1_all.deb
openssl_1.1.1n-0+deb10u5kylin1_mips64el_v10/openssl_1.1.1n-0+deb10u5kylin1_mips64el.deb
$ sudo dpkg -i openssl_1.1.1n-0+deb10u5kylin1_mips64el_v10/*.deb
[sudo] kylin 的密码:
正在选中未选择的软件包 libssl1.1:mips64el。
(正在读取数据库 ... 系统当前共安装有 243962 个文件和目录。)
正准备解包 .../libssl1.1_1.1.1n-0+deb10u5kylin1_mips64el.deb ...
正在解包 libssl1.1:mips64el (1.1.1n-0+deb10u5kylin1) ...
正准备解包 .../libssl-dev_1.1.1n-0+deb10u5kylin1_mips64el.deb ...
正在将 libssl-dev:mips64el (1.1.1n-0+deb10u5kylin1) 解包到 (1.0.2g-1kord4.15) 上 ...
正在选中未选择的软件包 libssl-doc。
正准备解包 .../libssl-doc_1.1.1n-0+deb10u5kylin1_all.deb ...
正在解包 libssl-doc (1.1.1n-0+deb10u5kylin1) ...
正准备解包 .../openssl_1.1.1n-0+deb10u5kylin1_mips64el.deb ...
正在将 openssl (1.1.1n-0+deb10u5kylin1) 解包到 (1.0.2g-1kord4.15) 上 ...
正在设置 libssl1.1:mips64el (1.1.1n-0+deb10u5kylin1) ...
正在设置 libssl-dev:mips64el (1.1.1n-0+deb10u5kylin1) ...
```

```
正在设置 libssl-doc (1.1.1n-0+deb10u5kylin1) ...
正在设置 openssl (1.1.1n-0+deb10u5kylin1) ...
正在安装新版本配置文件 /etc/ssl/openssl.cnf ...
正在处理用于 libc-bin (2.23-0kord11k20.8) 的触发器 ...
正在处理用于 man-db (2.7.5-1kord) 的触发器 ...
```

c. 安装 Nodejs

通过 `nvm` 离线安装高版本 Node.js，需按“下载包→建目录→解压→激活”四步操作，具体流程如下：

1、下载 Node.js 二进制离线包

从适配 Mips64el 架构的渠道（如前文提及的专用链接），下载对应版本的 Node.js 二进制 tar 包 `node-v16.17.1-linux-mips64el.tar.gz`。

2、创建 nvm 版本安装目录

在 `nvm` 的版本管理目录下，手动创建对应版本的安装文件夹，默认路径为 `~/.nvm/versions/node/vX.X.X`（将 `vX.X.X` 替换为实际下载的 Node.js 版本，如 `v16.17.1`）。

```
$ mkdir -p ~/.nvm/versions/node/v16.17.1
```

3、解压 tar 包到目标目录

执行解压命令，将下载的 tar 包直接解压到上一步创建的目录中，确保包内文件（如 `bin`、`lib` 文件夹）直接位于 `vX.X.X` 目录下。

```
$ tar xvf node-v16.17.1-linux-mips64el.tar.gz -C ~/.nvm/versions/node/v16.17.1 --strip-components 1
```

3、使用 nvm use 命令激活版本

执行 `nvm use X.X.X` 命令，即可将当前终端会话的 Node.js 版本切换为离线安装的版本，执行 `node -v` 可验证是否生效。若需长期生效，可执行 `nvm alias default vX.X.X` 将其设为默认版本。

```
$ nvm use 16.17.1
```

```
Now using node v16.17.1 (npm v8.15.0)
```

d. 验证 Nodejs 安装的版本

```
$ node -v
```

```
v16.17.1
```

```
$ npm -v
```

```
8.15.0
```

4.1.2 从源中安装 git

[参考 2.1.2 章节](#)。

4.1.3 设置 npm 仓库源(必选)

在龙芯架构环境中，需将 npm 仓库源切换为龙芯专用源，以获取适配该架构的依赖包（避免官方源无适配版本的问题），具体配置方法如下：

```
# 设置 npm 仓库源为龙芯专用源
$ npm config set registry https://registry.loongnix.cn:4873
# 验证配置是否生效（可选）
$ npm config get registry
# 若输出 https://registry.loongnix.cn:4873，说明配置成功
```

注：

龙芯 npm 仓库的详细使用指南可参考官方文档：

<https://docs.loongnix.cn/nodejs/Doc/list/03.龙芯 npm 的安装和仓库配置使用.html>

4.1.4 设置 electron 镜像源(必选)

龙芯开源生态社区当前主要维护 Loongarch64 架构资源，Mips64el 架构资源已基本停止网站层面的更新。针对 Mips64el 架构的 Electron 需求，需按版本情况选择获取方式。龙芯源中仍保留 Mips64el 架构的部分 Electron 旧版本，无需额外申请，直接通过镜像源即可安装，支持版本包括 v4.1.3、v6.1.7、v10.1.0。若需 Mips64el 架构的更高版本 Electron（上述版本之外），需通过线下提需求的方式获取，社区暂不提供线上直接下载渠道。

对于需要 Electron v4.1.3、v6.1.7、v10.1.0 版本的，可以直接配置龙芯存放 electron 的镜像源，配置方法如下：

```
# 临时生效（当前终端会话）
$ export ELECTRON_MIRROR="http://ftp.loongnix.cn/os/loongnix/1.0/electron/releases/mips/"
# 永久生效（需根据 shell 类型配置，以 bash 为例）
$ echo 'export ELECTRON_MIRROR="http://ftp.loongnix.cn/os/loongnix/1.0/electron/releases/mips/"' >> ~/.bashrc
source ~/.bashrc
```

若通过线下渠道获取了高版本 Electron 包，需通过本地搭建 Electron 镜像源实现安装。具体搭建步骤可参考 [附录 6.1 章节](#)，确保本地源能被 npm 正确识别调用。

4.2 项目开发

此处以 electron-quick-start 项目使用 electron-v20.0.3 开发为例：

4.2.1 下载 electron-quick-start 项目

[参考 2.2.1 章节](#)。

4.2.2 修改 electron 版本

编辑 package.json 文件，将 electron 版本修改为 20.0.3。

```
$ cat package.json
{
  "name": "electron-quick-start",
  "version": "1.0.0",
  "description": "A minimal Electron application",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "repository": "https://github.com/electron/electron-quick-start",
  "keywords": [
    "Electron",
    "quick",
    "start",
    "tutorial",
    "demo"
  ],
  "author": "GitHub",
  "license": "CC0-1.0",
  "devDependencies": {
    "electron": "20.0.3"
  }
}
```

4.2.3 安装 electron 及依赖包

执行以下命令安装 Electron 及项目依赖包，其中 `electron_use_remote_checksums=1` 用于启用远程校验和验证，确保安装包完整性。

```
$ electron_use_remote_checksums=1 npm install

added 91 packages in 3m

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

4.3 启动项目

执行 `npm start` 命令来启动项目。

```
$ 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 start` 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 .

[981:0817/155620.718418:FATAL:setuid_sandbox_host.cc(157)] 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.
#0 0x00aab150393c base::debug::CollectStackTrace()
#1 0x00aab155cacc base::debug::StackTrace::StackTrace()
#2 0x00aab1445b14 logging::LogMessage::~~LogMessage()
#3 0x00aab3002aa4 sandbox::SetuidSandboxHost::PrependWrapper()
#4 0x00aab06f89d4 content::ZygoteHostImpl::LaunchZygote()
#5 0x00aaad1bdffc content::(anonymous namespace)::LaunchZygoteHelper()
#6 0x00aaad1be070 base::internal::Invoker<>::RunOnce()
#7 0x00aaae4b1da0 content::ZygoteCommunication::Init()
#8 0x00aaae4b24f8 content::CreateGenericZygote()
#9 0x00aaad1bce4c content::ContentMainRunnerImpl::Initialize()
#10 0x00aaad1ba184 content::RunContentProcess()
#11 0x00aaad1ba398 content::ContentMain()
#12 0x00aaacd0a534 main
#13 0x00fff1c04aa4 __libc_start_main

Crash keys:
  "platform" = "linux"
  "process_type" = "browser"

/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/2023-08-17T07_56_21_674Z-debug.log
```

根据报错提示可以看出是 `chrome-sandbox` 权限问题，需要修改权限

```
$ 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

项目调试完成后，需打包为符合《银河麒麟桌面操作系统 V10-DEB 包打包规范》的 `deb` 格式安装包。X86_64 和 ARM64 架构一般是使用 `electron-builder` 工具来进行打包，但 `npm` 官方源中的 `electron-builder` 不支持 Mips64el 架构打包，需要使用龙芯 `npm` 仓库中已经适配的 `electron-builder` 版本。

龙芯 `npm` 仓库适配列表见：[https://docs.loongnix.cn/nodejs/Doc/list/04.npm 仓库适配列表.html](https://docs.loongnix.cn/nodejs/Doc/list/04.npm%20仓库适配列表.html)。

此处以使用 `electron-builder-v22.14.7` 版本打包为例。

4.4.1 安装 electron-builder

执行下面的命令来安装 `electron-builder`，安装成功后，会有如下提示信息。

```
$ npm install electron-builder@22.14.7 --save-dev
```

```
added 189 packages in 2m
```

```
31 packages are looking for funding
```

```
run `npm fund` for details
```

4.4.2 添加打包命令

[参考 2.4.2 章节](#)。

4.4.3 添加应用图标

[参考 2.4.3 章节](#)。

4.4.4 安装 fpm

[参考 3.4.4 章节](#)。

4.4.5 打 deb 包

使用 electron-builder 打包时会从 electron_mirror 镜像源中拉取 electron 包，Mips64el 架构拉取时的链接有一些问题，我们可以将本地已经存在的 electron 包拷贝到 electron 缓存目录，从而在打包时不需要再次拉取 electron 包，具体操作如下：

```
$ cp ~/.cache/electron/*/electron-v20.0.3-linux-mips64el.zip ~/.cache/electron/
```

注：上面命令中的“*”是一个很长的字符串，需要根据实际值进行填写

执行 `npm run builder` 命令进行打包，打包时间会有些长，等待完成即可。

打包完成后会有如下提示，表示打包成功。

```
$ npm run builder

> electron-quick-start@1.0.0 builder
> electron-builder

• electron-builder version=22.14.7 os=5.4.18-53-generic
• loaded configuration file=package.json ("build" field)
• writing effective config file=dist/builder-effective-config.yaml
• packaging platform=linux arch=mips64el electron=20.0.3 appOutDir=dist/linux-mips64el-unpacked
• asar usage is disabled — this is strongly not recommended solution=enable asar and use asarUnpack to
unpack files that must be externally available
• asar usage is disabled — this 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-debug.yml          electron-quick-start_1.0.0_mips64el.deb
builder-effective-config.yaml  linux-mips64el-unpacked
```

注：

V10 系统打包过程中可能会出现卡死无法完成打包的情况，使用 `Ctrl+c` 取消会有如下报错：

```
• cancelled by SIGINT
/home/kylin/electron-quick-start/node_modules/app-builder-bin/linux/mips64el/app-builder exited with code
ERR_ELECTRON_BUILDER_CANNOT_EXECUTE failedTask=build stackTrace=Error:
/home/kylin/electron-quick-start/node_modules/app-builder-bin/linux/mips64el/app-builder exited with code
ERR_ELECTRON_BUILDER_CANNOT_EXECUTE
  at ChildProcess.<anonymous> (/home/kylin/electron-quick-start/node_modules/builder-util/src/util.ts:250:14)
  at Object.onceWrapper (node:events:628:26)
  at ChildProcess.emit (node:events:513:28)
  at maybeClose (node:internal/child_process:1093:16)
  at Socket.<anonymous> (node:internal/child_process:451:11)
```

```
at Socket.emit (node:events:513:28)
at Pipe.<anonymous> (node:net:757:14)
```

出现如上问题，需要替换一下 `app-builder` 文件

文件下载地址链接：

<https://wwpp.lanzoum.com/iXQOA16t1xfg> 密码:c1xa

下载解压后，将 `~/electron-quick-start/node_modules/app-builder-bin/linux/mips64el/` 目录下的 `app-builder` 文件备份，然后将解压目录下的 `app-builder` 文件拷贝到此目录下，再次进行打包即可。

另外，V10 系统上如果打好的包如果无法运行，并且从终端启动报错如下：

```
$ ./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
```

4.4.6 修改 deb 包

[参考 2.4.5 章节](#)。

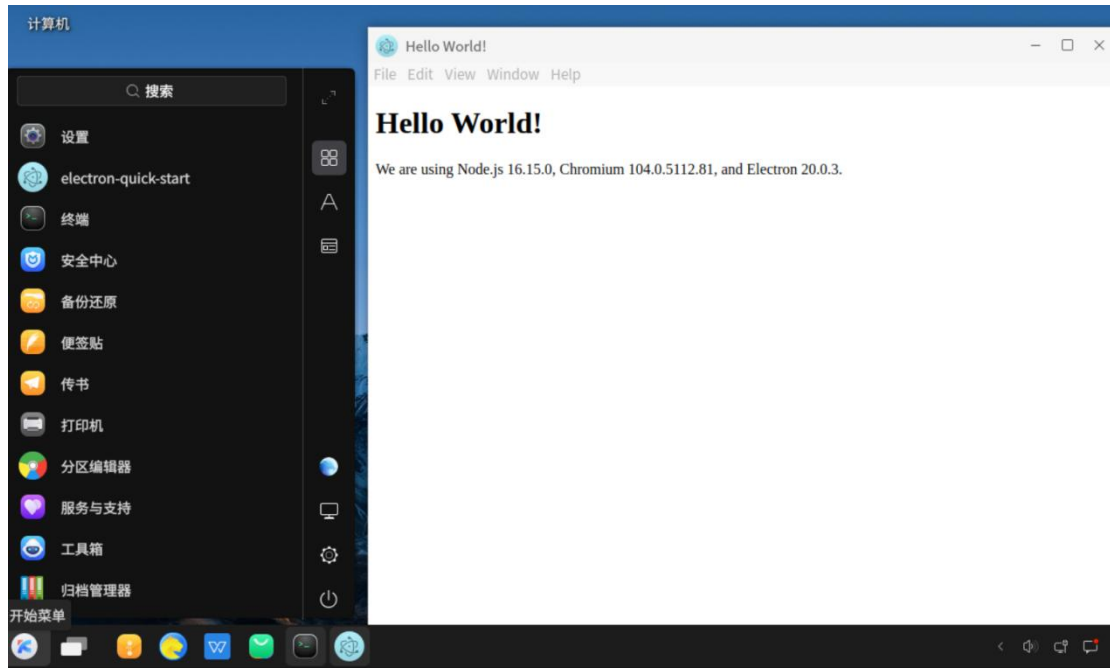
4.5 验包

4.5.1 安装

在 Mips64el 架构机器上，双击或在终端执行 `sudo dpkg -i ***.deb` 来安装 deb 包。

4.5.2 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



5 LoongArch64 架构

LoongArch64 架构与 Mips64el 架构类似，均存在官方资源缺失问题：Node.js 官方未提供该架构的二进制 tar 包，npm 官方源也无适配的 Electron 及相关依赖包，因此需从龙芯开源社区获取适配资源。目前龙芯已完成多个版本的适配，具体适配版本可以从[龙芯开源社区-Node.js 历史版本](#)和[龙芯生态社区-Electron 历史版本](#)中查看。

龙芯开源社区的适配版本会随技术迭代更新，建议在获取资源前访问社区官网或文档中心，确认目标版本的适配状态及最新下载链接，避免使用过时或未完全适配的版本导致运行异常。

5.1 安装开发基础环境

5.1.1 安装 Nodejs

龙芯开源社区已为 LoongArch64 架构适配了多个稳定版本的 Node.js，可直接通过 nvm 工具安装，安装时可以使用 `nvm ls-remote node` 命令获取龙芯社区已适配的 Node.js 版本。

```
$ nvm ls-remote node
  v10.24.1 (LTS: Dubnium)
  v10.24.2 (Latest LTS: Dubnium)
  v12.19.1 (LTS: Erbium)
 v12.22.12 (Latest LTS: Erbium)
  v14.16.1 (LTS: Fermium)
  v14.21.3 (Latest LTS: Fermium)
   v16.3.0
   v16.5.0
  v16.17.1 (LTS: Gallium)
  v16.20.1 (LTS: Gallium)
  v16.20.2 (Latest LTS: Gallium)
  v18.13.0 (LTS: Hydrogen)
  v18.18.1 (LTS: Hydrogen)
  v18.19.1 (LTS: Hydrogen)
  v18.20.2 (LTS: Hydrogen)
  v18.20.3 (LTS: Hydrogen)
  v18.20.8 (Latest LTS: Hydrogen)
   v20.8.0
  v20.11.1 (LTS: Iron)
  v20.13.0 (LTS: Iron)
  v20.13.1 (LTS: Iron)
  v20.19.2 (Latest LTS: Iron)
```

```
v21.7.3
v22.1.0
-> v22.16.0 (Latest LTS: Jod)
```

5.1.2 从源中安装 git

[参考 2.1.2 章节。](#)

5.1.3 设置 npm 仓库源(必选)

[参考 4.1.3 章节。](#)

5.1.4 设置 electron 镜像源(必选)

配置 npm 使用龙芯提供的 electron 镜像源，配置方法如下：

```
# 临时生效（当前终端会话）
$ export ELECTRON_MIRROR="http://ftp.loongnix.cn/os/loongnix/1.0/electron/releases/mips/"
# 永久生效（需根据 shell 类型配置，以 bash 为例）
$ echo 'export ELECTRON_MIRROR="http://ftp.loongnix.cn/os/loongnix/1.0/electron/releases/mips/"' >> ~/.bashrc
source ~/.bashrc
```

5.2 项目开发

此处以 electron-quick-start 项目使用 electron-v20.0.3 开发为例：

5.2.1 下载 electron-quick-start 项目

[参考 2.2.1 章节。](#)

5.2.2 修改 electron 版本

编辑 package.json 文件，将 electron 版本修改为 20.0.3。

```
$ cat package.json
{
  "name": "electron-quick-start",
  "version": "1.0.0",
  "description": "A minimal Electron application",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "repository": "https://github.com/electron/electron-quick-start",
  "keywords": [
    "Electron",
    "quick",
    "start",
```

```
"tutorial",
  "demo"
],
"author": "GitHub",
"license": "CC0-1.0",
"devDependencies": {
  "electron": "20.0.3"
}
}
```

5.2.3 安装 electron 及依赖包

执行以下命令安装 Electron 及项目依赖包，其中 `electron_use_remote_checksums=1` 用于启用远程校验和验证，确保安装包完整性。

安装完成后会有如下提示，表示 electron 及依赖包安装成功。

```
$ electron_use_remote_checksums=1 npm install

added 13 packages, removed 191 packages, and changed 2 packages in 2m

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

5.3 启动项目

执行 `npm start` 命令来启动项目。



如上图所示，即表示项目运行成功。

5.4 打包

项目调试完成后，需打包为符合《银河麒麟桌面操作系统 V10-DEB 包打包规范》的 `deb` 格式安装包。此处以使用 `electron-builder-v22.14.7` 版本打包为例。

5.4.1 安装 electron-builder

执行下面的命令来安装 `electron-builder`，安装成功后，会有如下提示信息。

```
$ npm install electron-builder@22.14.7 --save-dev
```

```
added 190 packages in 2m
```

```
31 packages are looking for funding  
run `npm fund` for detail
```

5.4.2 添加打包命令

[参考 2.4.2 章节。](#)

5.4.3 添加应用图标

[参考 2.4.3 章节。](#)

5.4.4 安装 fpm

[参考 3.4.4 章节。](#)

5.4.5 打 deb 包

执行 `npm run builder` 命令进行打包，打包时间会有些长，等待完成即可。

打包完成后会有如下提示，表示打包成功。

```
$ npm run builder  
  
> electron-quick-start@1.0.0 builder  
> electron-builder  
  
• electron-builder version=22.14.7 os=5.4.18-55-generic  
• loaded configuration file=package.json ("build" field)  
• writing effective config file=dist/builder-effective-config.yaml  
• packaging platform=linux arch=loong64 electron=20.0.3 appOutDir=dist/linux-loong64-unpacked  
• asar usage is disabled — this is strongly not recommended solution=enable asar and use asarUnpack to  
unpack files that must be externally available  
• asar usage is disabled — this is strongly not recommended solution=enable asar and use asarUnpack to  
unpack files that must be externally available  
• building target=deb arch=loong64 file=dist/electron-quick-start_1.0.0_loong64.deb
```

打包后可以看到 `dist` 目录下生成的 `deb` 包。

```
$ ls dist/  
builder-debug.yml          electron-quick-start_1.0.0_loong64.deb    linux-loong64-unpacked  
builder-effective-config.yaml
```

5.4.6 修改 deb 包

[参考 2.4.5 章节。](#)

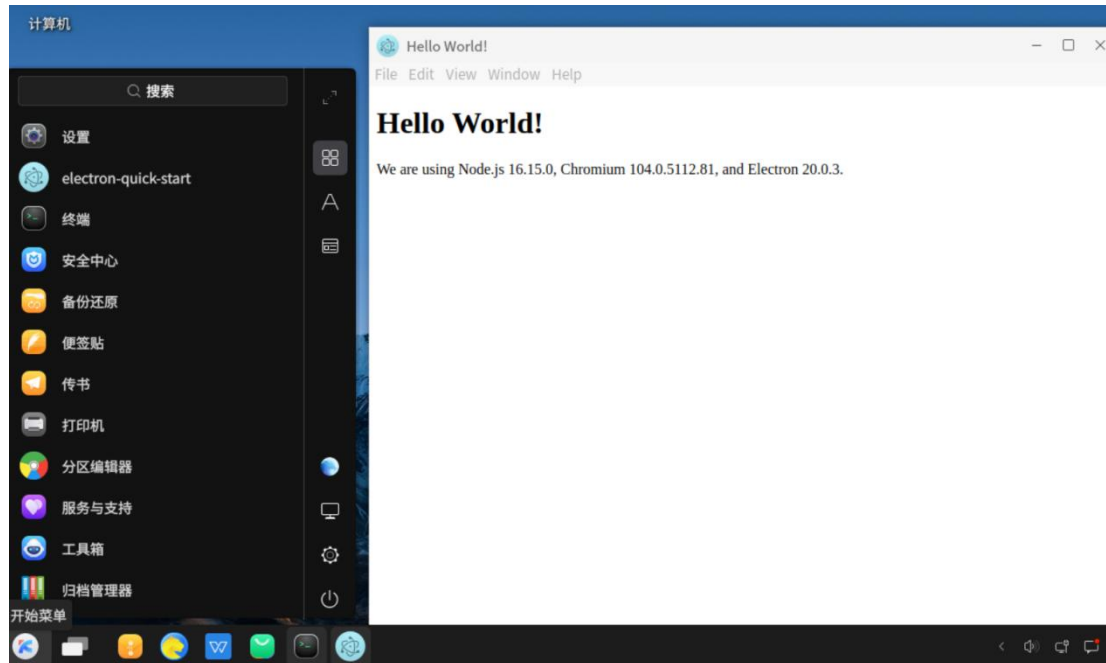
5.5 验包

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 搭建 electron 本地镜像源

下面介绍一种在本机快速搭建一个 electron 镜像源的方法，供打包使用。

6.1.1 创建 electron 镜像源目录

新开启一个终端，执行如下命令。

```
$ cd ~
$ mkdir -p http/electron
$ cd http/electron
```

6.1.2 将 electron 包放入上述目录

下载 electron_v20.0.3-bin.tar.gz，然后进行解压

```
$ tar xvf electron_v20.0.3-bin.tar.gz
$ tree -L 3
.
├── electron_v20.0.3-bin.tar.gz
└── v20.0.3
    ├── chromedriver-v20.0.3-linux-loong64.zip
    ├── chromedriver-v20.0.3-linux-mips64el.zip
    ├── electron-api.json
    ├── electron.d.ts
    ├── electron-v20.0.3-linux-loong64.zip
    ├── electron-v20.0.3-linux-mips64el.zip
    ├── ffmpeg-v20.0.3-linux-loong64.zip
    ├── ffmpeg-v20.0.3-linux-mips64el.zip
    ├── hunspell_dictionaries.zip
    ├── mksnapshot-v20.0.3-linux-loong64.zip
    ├── mksnapshot-v20.0.3-linux-mips64el.zip
    └── SHASUMS256.txt

1 directory, 13 files
```

#创建不带 v 的版本目录，解决打包时 Response 404(Not Found)问题。

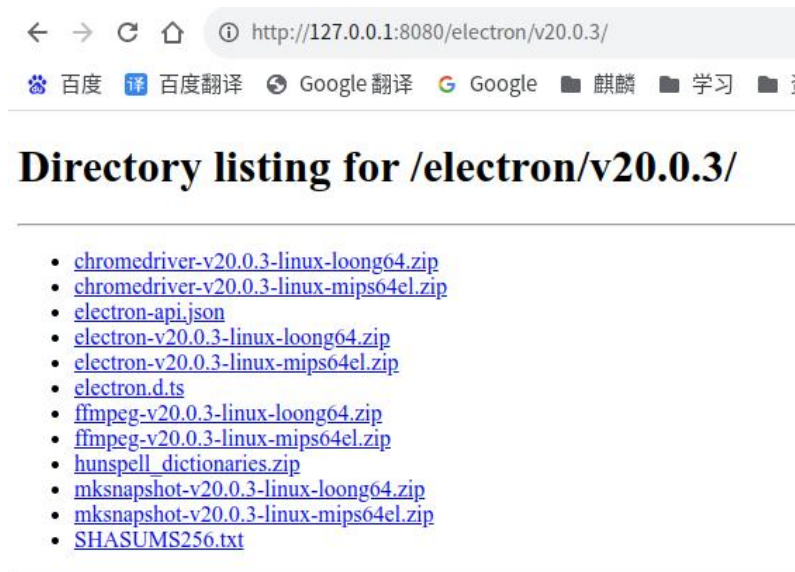
```
$ cp -r v20.0.3 20.0.3
```

6.1.3 启动 electron 镜像源

在 http 目录，使用如下命令启动 electron 镜像源。

```
$ cd ..
$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080) ...
```

然后在浏览器输入 `http://127.0.0.1:8080/`验证 electron 镜像源是否搭建成功。浏览器显示如下，表示 electron 镜像源搭建成功：



6.1.4 设置 electron 镜像源(必选)

配置 npm 使用本地搭建的 electron 镜像源，配置方法如下：

```
$ npm config set electron_mirror http://127.0.0.1:8080/electron/
```

7 获取帮助

如有需要帮助，可联系我们：

咨询热线

400-089-1870