electron-react学习手册

# 环境安装过程

1. 配置node
2. 安装create-react-app

npm install -g create-react-app

1. 创建项目

create-react-app react-demo

1. 展开项目

cd react-demo

npm run eject

1. 安装"@babel/plugin-proposal-decorators"

npm install @babel/plugin-proposal-decorators

1. 安装mobx

npm install mobx mobx-react

1. 修改package.json，红色部分为添加

"babel": {

"plugins": [

[

"@babel/plugin-proposal-decorators",

{

"legacy": true

}

]

],

"presets": [

"react-app"

]

}

}

1. npm i --安装依赖
2. 将package.json中dependes都移到devDependencies中，这个是把create-react-app得配置移到dev中。
3. 再根据需要安装依赖的库 npm install [electron@6.1.7](mailto:electron@6.1.7)
4. 项目目录下编写main.js

const { app, BrowserWindow } = require('electron')

// 保持对window对象的全局引用，如果不这么做的话，当JavaScript对象被

// 垃圾回收的时候，window对象将会自动的关闭

let win

function createWindow () {

// 创建浏览器窗口。

win = new BrowserWindow({

width: 800,

height: 600,

webPreferences: {

nodeIntegration: true,

webviewTag:true //使用webview

}

})

// 加载index.html文件

win.loadFile('./build/index.html')

// 打开开发者工具

win.webContents.openDevTools()

// 当 window 被关闭，这个事件会被触发。

win.on('closed', () => {

// 取消引用 window 对象，如果你的应用支持多窗口的话，

// 通常会把多个 window 对象存放在一个数组里面，

// 与此同时，你应该删除相应的元素。

win = null

})

}

// Electron 会在初始化后并准备

// 创建浏览器窗口时，调用这个函数。

// 部分 API 在 ready 事件触发后才能使用。

app.on('ready', createWindow)

// 当全部窗口关闭时退出。

app.on('window-all-closed', () => {

// 在 macOS 上，除非用户用 Cmd + Q 确定地退出，

// 否则绝大部分应用及其菜单栏会保持激活。

if (process.platform !== 'darwin') {

app.quit()

}

})

app.on('activate', () => {

// 在macOS上，当单击dock图标并且没有其他窗口打开时，

// 通常在应用程序中重新创建一个窗口。

if (win === null) {

createWindow()

}

})

// 在这个文件中，你可以续写应用剩下主进程代码。

// 也可以拆分成几个文件，然后用 require 导入。

1. 修改package.json文件 添加：

"homepage": ".",

"main":"main.js",

“script”: {

Electron-start: “electron .”,

}

## Electron下载慢的问题解决方案

安装Electron时会判断缓存中是否有要下载的zip包，如果没有的话就去下载这个zip包。但是往往下载这个zip包时非常慢，总是下载不成功。下面给出我的解决办法：

首先要知道缓存位置在哪，根据 [**electron-download**](https://github.com/electron-userland/electron-download)的描述,缓存的位置取决于操作系统，默认值为：  
Linux：$XDG\_CACHE\_HOME或~/.cache/electron/  
苹果系统：~/Library/Caches/electron/  
Windows：$LOCALAPPDATA/electron/Cache或~/AppData/Local/electron/Cache/

然后，[下载](https://npm.taobao.org/mirrors/electron)需要的zip版本的包,放到上面提到的缓存地址中。

最后，在命令窗口中执行npm install electron -g,执行成功

使用npm install [electron@6.1.3](mailto:electron@6.1.3) -g 安装特定版本的electron

问题描述

在运行官网示例的过程中，一个安装包（electron-v1.8.6-darwin-x64.zip）下载特别慢，基本下载不下来。更改npm的安装源到taobao的镜像，也没有效果。

解决方案

所以，可以先在这里下载安装包：

https://npm.taobao.org/mirrors/electron/

然后，将安装包（.zip文件）拷贝到以下路径（Mac）：

# electron

## Electron工作原理

Eletron appliactions consist of two types of processes: the main process and zero or more renderer processes. Each process plays a different role in the appliaction. The electron runtime includes different modules to assist you in building your application. Certain modules, such as the ability to read and write from the system’s clipboard, are available in both types of processes. Others, such as the ability to access an operating system’s APIs, are limited to the main process.

* The main process

The main process has a few important responsibilites. It can respond to application lifecycle events such as starting up, quitting, preparing to quit, going to the background, coming to the foreground, and more. The main process is also responsible for communicating to native operating system APIs. If you want to display a dialog box to open or save a file, you do it from the main process.

* Renderer processes

The main process can create and destroy renderer processes using Electron’s BrowserWindow module. Renderer processes can load web pages to display a GUI. Each process takes advantage of Chromiums’multiprocess architecture and runs on its own thread.

## Electron主要API

### app

应用程序的本身，提供控制应用程序的生命周期，提供事件。

#### 方法

##### App.getAppPath()

返回String类型-当前应用程序所在目录

##### App.getPath(name)

Home – 用户的home文件夹目录

appData – 当前用户的 应用数据文件夹

* + %APPDATA% Windows中
  + $XDG\_CONFIG\_HOME or ~/.config Linux中
  + ~/Library/Application Support macOS中

userData存储你应用程序设置文件的文件夹，默认是appData文件夹附加应用的名称

cache 缓存

temp 临时目录

exe 当前执行的文件

module libchromiumcontent库

desktop 当前用户的桌面目录

documents “我的文档”目录

downloads “下载”目录

music 音乐目录

pictures 用户pictures目录

videos 用户videos目录

log 你的app的log目录

pepperFlashSystemPlugin Pepper Flash插件的全路径

返回String 以name参数指定的文件夹或文件路径。当失败时抛出Error。

If app.getPath(‘logs’) is called without called app.setAppLogsPath() being called first, a default log directory will be created equivalent to calling app.setAppLogsPath() without a path parameter.

### BrowserWindow

创建和控制浏览器窗口

选项

* webPreferences 网页功能的设置
  + nodeIntegration – 是否集成Node,默认为false
  + nodeIntegrationInWorker – 是否再Web工作器中启用了Node集成，默认为false
  + webSecurity – 当设置为false，它将禁用同源策略，如果此选项不是由开发者设置的，还会把allowRunningInsecureContent设置为true.默认为true.
  + allowRunningInsecureContent – 允许一个https页面运行http url里的资源，包括JavaScript,CSS或plugins，默认为false。
  + preload 在页面运行其他脚本之前预先加载指定的脚本。无论页面是否集成Node，此脚本都可以访问所有Node API脚本路径的文件的绝对路径。当node integration关闭时，预加载的脚本将从全局范围重新引入node的全局引用标准。

### ipcMain

从主进程到渲染进程的异步通信，是EventEmitter类的一个实例。当在主进程使用时，它处理从渲染进程（网页）发送出来的异步和同步信息。从渲染进程发送的消息将被发送到该模块。

### WebContents

渲染和控制 BrowserWindow 实例的内容。webContents 是 EventEmitter 的实例， 负责渲染和控制网页, 是 BrowserWindow 对象的一个属性。

### ipcRenderer

从渲染器进程到主进程的异步通信。

进程: Renderer

ipcRenderer 是一个 EventEmitter 的实例。 你可以使用它提供的一些方法从渲染进程 (web 页面) 发送同步或异步的消息到主进程。 也可以接收主进程回复的消息。

### remote

在渲染进程中使用主进程模块。在渲染进程中，它代表主进程。使用remote模块，可以调用main进程对象的方法。

### webFrame

在渲染进程中使用，表示当前BrowserWindow的top frame。子框架可以通过属性和方法获得

# 组件的显示和隐藏

## 原理

html5的元素显示属性，在react中可以使用

display属性：

block

块级元素的默认值，元素会被显示为块级元素，该元素前后会带有换行符

inline

内联元素的默认值。元素会被显示为内联元素，该元素前后没有换行符

inline-block

行内块元素，元素既具有内联元素的特性，也具有块元素的特性

none

设置元素不会被显示

## react中使用

在react中使用style={{'display':'block'}}设置。

# 第一个electron应用

使用nodejs 12.13.0, electron6.1.4

const electron = require('electron');

const {app, BrowserWindow} = require('electron');

function createWindow(){

let win = new BrowserWindow({

width:800,

height: 600,

webPreferences: {

nodeIntegration: true

}

})

win.loadFile('index.html');

win.webContents.openDevTools()

win.on('closed',()=> {

win = null;

})

}

app.on('ready',createWindow);

app.on('window-all-closed', () => {

// 在 macOS 上，除非用户用 Cmd + Q 确定地退出，

// 否则绝大部分应用及其菜单栏会保持激活。

if (process.platform !== 'darwin') {

app.quit()

}

})

app.on('activate', () => {

// 在macOS上，当单击dock图标并且没有其他窗口打开时，

// 通常在应用程序中重新创建一个窗口。

if (win === null) {

createWindow()

}

})

# React向后跨域申请数据

## 客户端

create-react-app使用whatwg-fetch这个组件，

import React from 'react';

import {fetch } from 'whatwg-fetch';

export default class extends React.Component{

componentDidMount() {

fetch("http://localhost:8081/",{credentials:'omit'}).then(function(**response**){

return response.json();

}).then(function(json){

console.log('parsed json',json)

console.log(json.user);

}).catch(function(ex){

console.log('parsing failed',ex)

})

}

render() {

return (

<div>

<p>Hello world</p>

</div>

);

}

}

## 服务器端使用express

var express = require('express');

var app = express();

app.get('/', function (req, res) {

**res.header("Access-Control-Allow-Origin", "\*"); //关键这一句**

res.send({user:'Hello World'});

})

var server = app.listen(8081, function () {

var host = server.address().address

var port = server.address().port

console.log("应用实例，访问地址为 http://%s:%s", host, port)

})

# Express后端渲染react --

## 初始化项目express-react

Mkdir express-react

Cd express-react init -y

## 安装依赖包

### 全局安装包

npm install -g nodemon

### 项目安装包

npm install --save-dev @babel/cli @babel/core @babel/node @babel/plugin-proposal-class-properties @babel/plugin-transform-runtime @babel/polyfill @babel/preset-env

@babel/cli — Allows us to compile from command line

@babel/core — The core runtime

@babel/node — Allow us to use babel instead of node.

@babel/plugin-proposal-class-properties — Transpile classes

@babel/plugin-transform-runtime —To avoid duplication of files babel require.

@babel/polyfill — Needed when Promise is used

@babel/preset-env — Allows the use of latest javascript without needed to manage the transform

npm install express

npm install compression

npm install react react-dom handlebars

npm install –save-dev @babel/preset-react

npm install –save-dev webpack webpack-cli babel-loader

## 创建目录

Mkdir public

Mkdir src

Cd src

Mkdir routes

Mkdir components

## 代码文件

### Server.js

./src/server.js

import express from "express";

import compression from "compression";

import ssr from "./routes/ssr";

const app = express();

app.use(compression());

app.use(express.static("public"));

app.use("/firstssr", ssr);

const port = process.env.PORT || 3030;

app.listen(port, function listenHandler() {

console.info(`Running on ${port}...`);

});

### Ssr.js

./src/routes/ssr.js

import express from "express";

import App from "../components/app";

import React from "react";

import { renderToString } from "react-dom/server";

import hbs from "handlebars";

const router = express.Router();

router.get("/", async (req, res) => {

const theHtml = `

<html>

<head><title>My First SSR</title></head>

<body>

<h1>My First Server Side Render</h1>

<div id="reactele">{{{reactele}}}</div>

<script src="/app.js" charset="utf-8"></script>

<script src="/vendor.js" charset="utf-8"></script>

</body>

</html>

`;

const hbsTemplate = hbs.compile(theHtml);

const reactComp = renderToString(<App />);

const htmlToSend = hbsTemplate({ reactele: reactComp });

res.send(htmlToSend);

});

export default router;

### Index.js

./src/components/index.js

import React from "react";

import { hydrate } from "react-dom";

import App from "./app";

hydrate(<App />, document.getElementById("reactele"));

### App.jsx

./src/components/app.jsx

import React from "react";

class App extends React.Component {

constructor() {

super();

this.handleButtonClick = this.handleButtonClick.bind(this);

this.handleTextChange = this.handleTextChange.bind(this);

this.handleReset = this.handleReset.bind(this);

this.state = {

name: "",

msg: ""

};

}

//Handlers

handleButtonClick = e => {

const nameLen = this.state.name.length;

if (nameLen > 0) {

this.setState({

msg: `You name has ${nameLen} characters including space`

});

}

};

handleTextChange = e => {

this.setState({ name: e.target.value });

};

handleReset = () => {

this.setState({ name: "", msg: "" });

};

//End Handlers

render() {

let msg;

if (this.state.msg !== "") {

msg = <p>{this.state.msg}</p>;

} else {

msg = "";

}

return (

//do something here where there is a button that will replace the text

<div>

<label>Your name </label>

<input

type="text"

id="txtName"

name="txtName"

value={this.state.name}

onChange={this.handleTextChange}

/>

<button id="btnSubmit" onClick={this.handleButtonClick}>

Calculate Name Length

</button>

<button id="btnReset" onClick={this.handleReset}>

Reset All

</button>

<hr />

{msg}

</div>

);

}

}

export default App;

## 配置

### 文件.babelrc

./.babelrc

{

"presets": [["@babel/preset-env", { "targets": { "node": "current" } }],

"@babel/preset-react"],

"plugins": ["@babel/plugin-proposal-class-properties"]

}

### Webpack.config.js

./webpack.config.js

const path = require("path");

const config = {

entry: {

vendor: ["@babel/polyfill", "react"],

app: ["./src/components/index.js"]

},

output: {

path: path.resolve(\_\_dirname, "public"),

filename: "[name].js"

},

module: {

rules: [

{

test: /\.(js|jsx)$/,

use: {

loader: "babel-loader",

options: {

presets: ["@babel/preset-env", "@babel/preset-react"]

}

},

exclude: /node\_modules/

}

]

},

resolve: {

extensions: [".js", ".jsx", ".json", ".wasm", ".mjs", "\*"]

}

};

module.exports = config;

### Package.json中添加执行脚本

“"scripts": {

"dev": "nodemon --exec babel-node src/server.js",

"webpack": "webpack -wd"

},

# React-electron-webview-html通信

## React-electron-webview代码

import React from 'react';

import logo from './logo.svg';

import './App.css';

class App extends React.Component {

constructor(props){

super(props);

}

componentDidMount(){

let that = this

const webview = document.querySelector('webview');

webview.addEventListener('dom-ready', (e) => {

webview.openDevTools();

webview.executeJavaScript("const {ipcRenderer} = require('electron')\n" +

"function onClick(){" +

"console.log('in web');" +

"}"+

" ipcRenderer.on('ping', () => {\n" +

" ipcRenderer.sendToHost('pong')\n" +

"})" ,false,function(){

webview.send('ping',"ping");

});

})

webview.addEventListener('ipc-message', (event) => {

console.log(event.channel)

// Prints "pong"

})

}

render(){

return (

<div className="App">

<webview src={"http://localhost:8080/webview\_t"} nodeintegration="true"/>

</div>

);

}

}

export default App;

## Html代码

<html>

<head>

<title>webview\_t</title>

</head>

<body>

<h1>in web view</h1>

<button onclick="onClick()">ontest</button> **//再次调用注入的函数**

</body>

</html>