

2021/6/19 - B 站直播间  
前端早早聊大会免费福利专场

早

## 如何用 Babel 做静态分析和代码转换

编译：神说要有光 | 14:00

## 如何将 Web 端打造成桌面端-Electron 实践

Electron：彭道宽@CVTE | 15:00

## 如何在前端埋彩蛋-商业价值和惊喜感

彩蛋：宇航@百度搜索 | 16:00



长按扫码报名，进群领取录播 / 讲稿 / PPT

# 如何将Web端打造成桌面端

## —— Electron 实战

@CVTE 彭道宽

# 自我介绍

## 彭道宽

江湖人称“彭于晏广州分晏”

2019年毕业进入CVTE 担任前端开发工程师

易课堂团队——智慧课堂相关业务



# 01 核心主题

Web 端

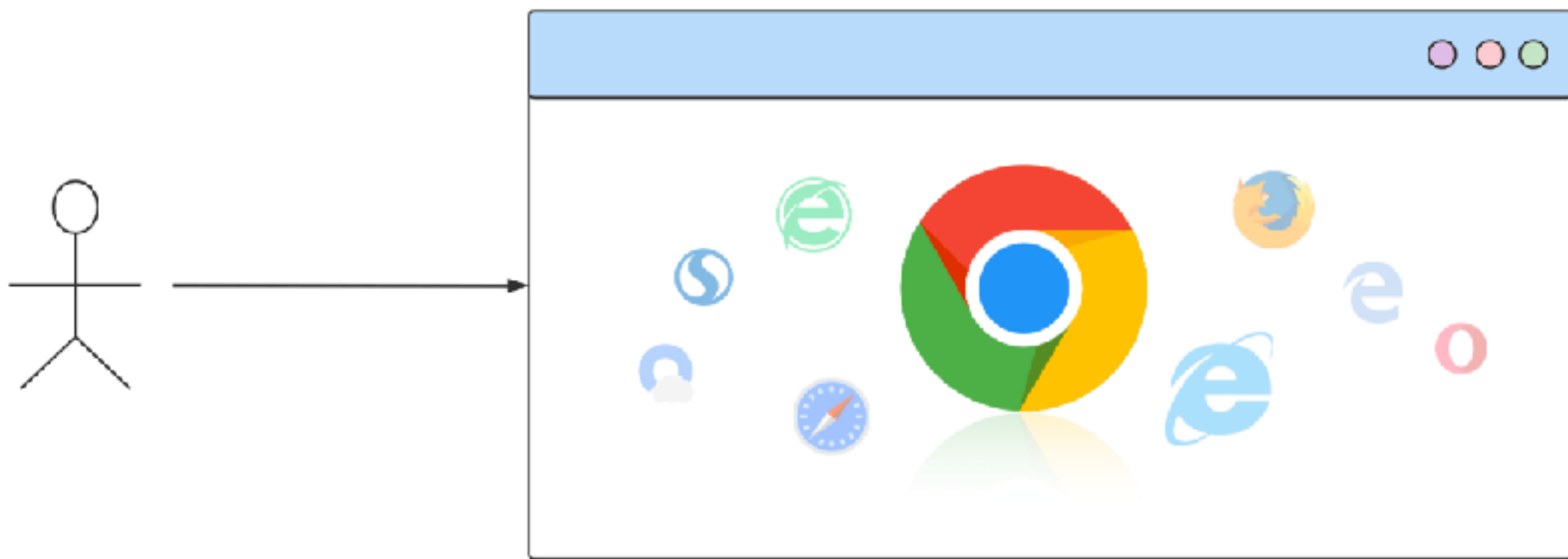


桌面端

切图仔



Web 前端开发工程师



原生应用？



## 02 什么是 Electron ?

# 哪些产品是用 Electron 开发呢？



Atom



VSCode



WordPress



希沃易课堂

还有更多.....

# Electron 介绍



Electron 比我们想象中的还要简单，如果你可以建一个网站，  
你就可以建一个桌面应用程序

# Electron 组成



**Chromium**  
for making  
web pages

+



**Node.js**  
for filesystems  
and networks

+



**Native APIs**  
for three  
systems

=



**ELECTRON**

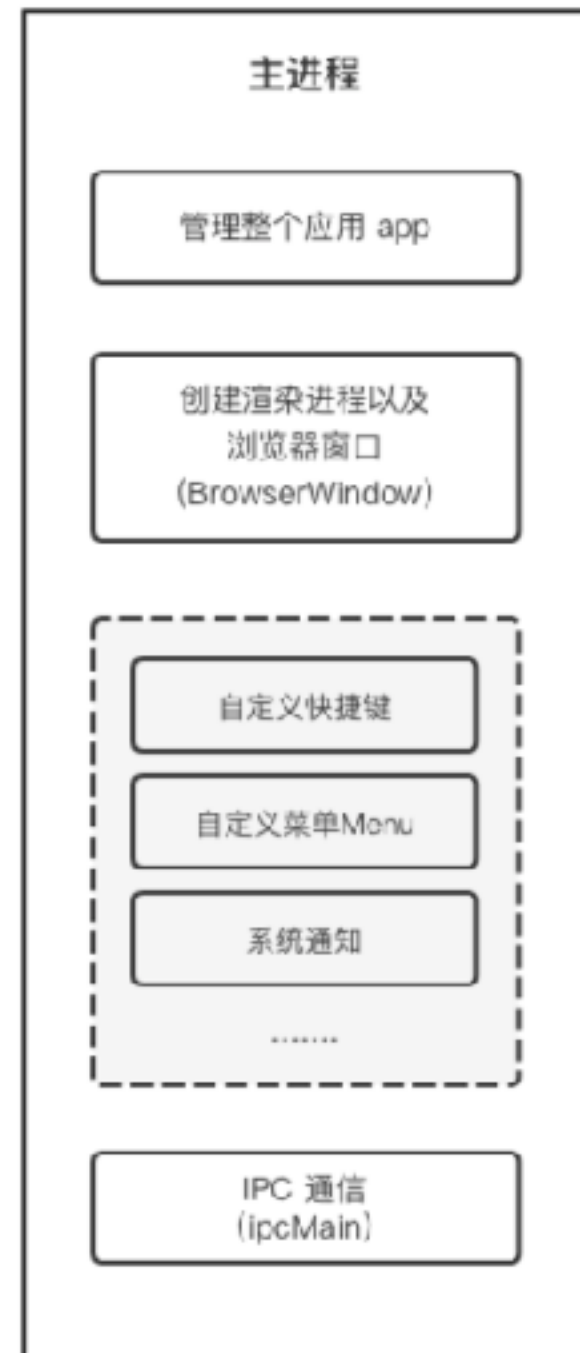
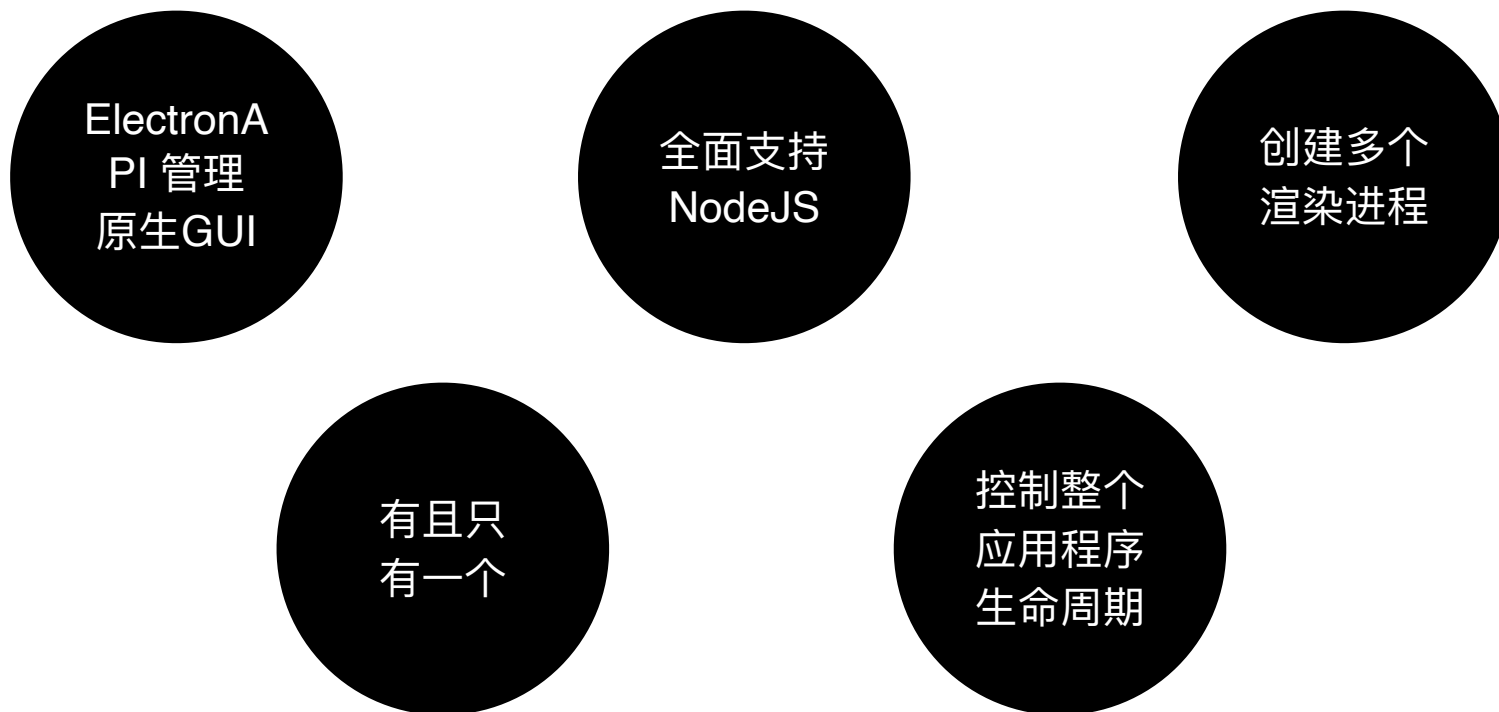
官方宣布，Electron 继承了来自 Chromium 的多进程架构

在 Electron 中，我们只需要关心两大进程：**主进程与渲染进程**

# 主进程

每一个 Electron 应用都有一个单一的主进程，作为应用程序的入口点。**主进程就像是一个桥梁，连接着渲染进程和操作系统。**

主进程特点：



# 渲染进程

在主进程中调用 `BrowserWindow` 时，会生成一个独立的渲染进程，恰如其名，渲染进程是负责渲染 Web 网页内容。

渲染进程的入口点是一个 **HTML 文件**。

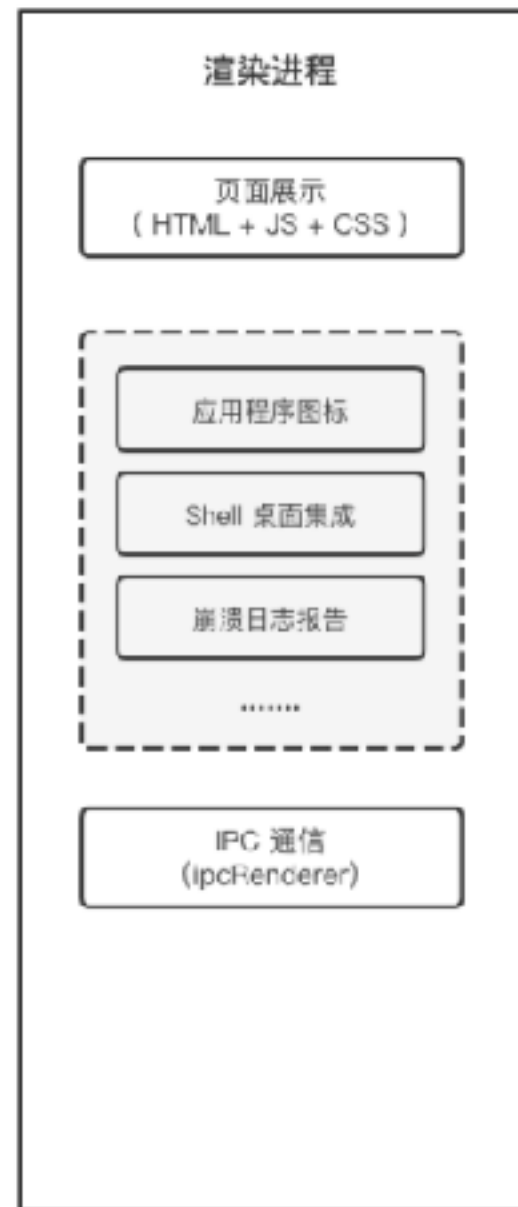
渲染进程特点：

可使用  
部分的  
Electron  
API

全面支持  
NodeJS

存在多个  
渲染进程

可以访问  
DOM API



# 主进程与渲染进程对 Electron 模块的访问范围

## Electron 常用 API

### 主进程

- app 控制应用程序的生命周期
- BrowserWindow 创建和控制浏览器窗口
- dialog 对话框
- globalShortcut 定制各种快捷键
- Menu 应用菜单，可自定义
- powerMonitor 电源管理，监视电源状态的变化
- protocol 自定义协议
- systemPreferences 系统界面
- Tray 系统托盘
- ipcMain 与渲染进程通信

### 渲染进程

- webFrame 自定义渲染当前网页
- contextBridge
- remote 远端调用主进程的模块
- desktopCapturer

在 v12.x 版本中将其移除

### 通用模块

- clipboard 剪切板
- crashReporter 崩溃日志报告
- nativeImage 应用层图标
- screen 获取屏幕相关信息
- shell 桌面集成相关功能



# 进程间的通信

1. 为什么进程之间要进行通信？
2. 主进程与渲染进程如何进行通信？
3. 渲染进程与渲染进程之间如何通信？

# 1. 为什么进程之间要进行通信？

从主进程与渲染进程对 Electron 模块的访问范围来看，很明显，主进程比渲染进程可访问的模块更多。比如 app 模块，只能在主进程中调用。在渲染进程中调用就会报错。

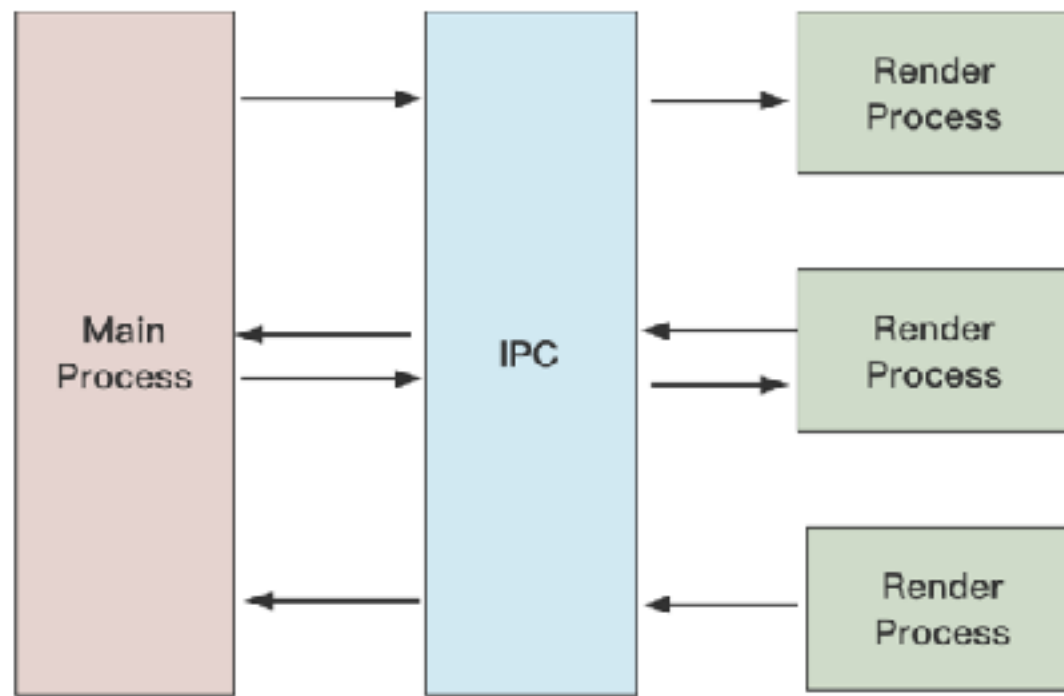
我们总会在渲染进程中用到某些数据，但该数据只能通过主进程访问特定模块才能获取，解决方式只能通过将主进程作为中间人，借助它的能力拿到数据之后，再通过 IPC 将数据发送给渲染进程。

## 2. 主进程与渲染进程如何进行通信？

Electron 中提供了 ipcMain 与 ipcRenderer 作为主进程与渲染进程通讯的桥梁。

ipcMain：作用于主进程中，处理从渲染器进程发送出来的异步和同步信息。

ipcRenderer：作用于渲染进程，将异步和同步信息发送到主进程，并且可以接收由主进程回复的消息。



## 2. 主进程与渲染进程如何进行通信？

在渲染进程中，需要得到应用程序在本机中的目录路径，可通过 `app.getAppPath` 方法获得，但 `app` 模块只能作用于主进程，所以这边需要实现进程间的通信。

```
// 在渲染进程中
import { ipcRenderer } from 'electron';

// 1. 向主进程发送消息，期望得到应用程序的目录路径
ipcRenderer.send('get-app-path');
// 2. 监听从主进程发送回来的消息
ipcRenderer.on('reply-app-path', (event, arg) => {
  if (arg) {
    console.log('应用程序路径: ', arg);
  } else {
    console.log('获取应用程序的路径出错');
  }
});
```

```
// 在主进程中
import { app, ipcMain } from 'electron';

// 获取应用程序的路径
const ROOT_PATH = app.getAppPath();

// 3. 监听渲染进程发送过来的消息
ipcMain.on('get-app-path', (event, arg) => {
  // 4. 监听到之后，主进程发送消息进行回复
  event.reply('reply-app-path', ROOT_PATH);
});
```

## 2. 主进程与渲染进程如何进行通信？

remote：它允许你在渲染进程中，调用主进程对象的方法，而不必显式地发送进程间消息。但官方在 v12.x 版本废弃掉了。

⚠ WARNING ⚠ The `remote` module is **deprecated**. Instead of `remote`, use `ipcRenderer` and `ipcMain`.

Read more about why the `remote` module is deprecated [here](#).

If you still want to use `remote` despite the performance and security concerns, see [@electron/remote](#).

```
// 在渲染进程
const app = require('electron').remote.app;

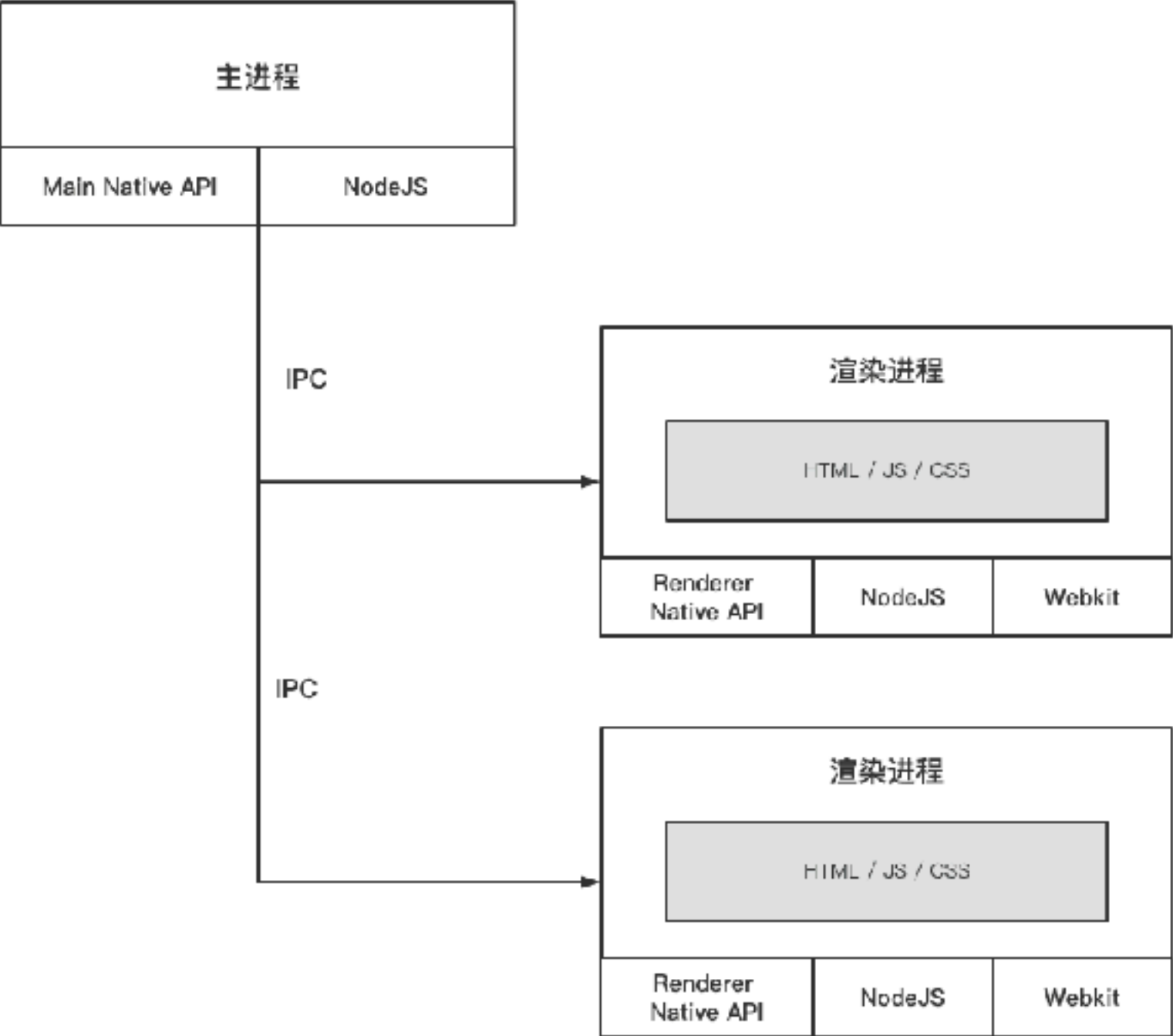
const rootPath = app.getAppPath();
console.log('应用程序路径: ', rootPath);
```

### 3. 渲染进程与渲染进程如何进行通信？

官方并没有提供渲染进程之间互相通信的方式，只能通过主进程建立一个消息中转。

渲染进程 A 与渲染进程 B 需要进行通信，那么渲染进程 A 先将消息发给主进程，主进程接收消息之后，再分发给渲染进程 B。

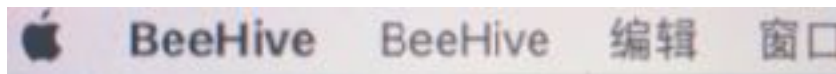
# Electron 结构图



# Electron 原生能力到底有多强？

## 1. 创建原生 GUI

- menu 自定义应用菜单



- dialog 对话框



## 2. 获取底层能力

- clipboard 剪切板

```
const { clipboard } = require('electron')  
  
const text = 'hello i am a bit of text!'  
clipboard.writeText(text)
```

- globalShortcut 定制快捷键
- desktopCapturer 实现桌面截屏，或者捕获音频/视频的媒体源信息
- shell 模块提供了集成其他桌面客户端的关联功能，比如说在用户默认浏览器中打开URL

```
const { shell } = require('electron')  
  
shell.openExternal('https://github.com')
```



# Electron 通过 NodeJS 获取底层能力

## 主进程和渲染进程都能使用 NodeJS 特性

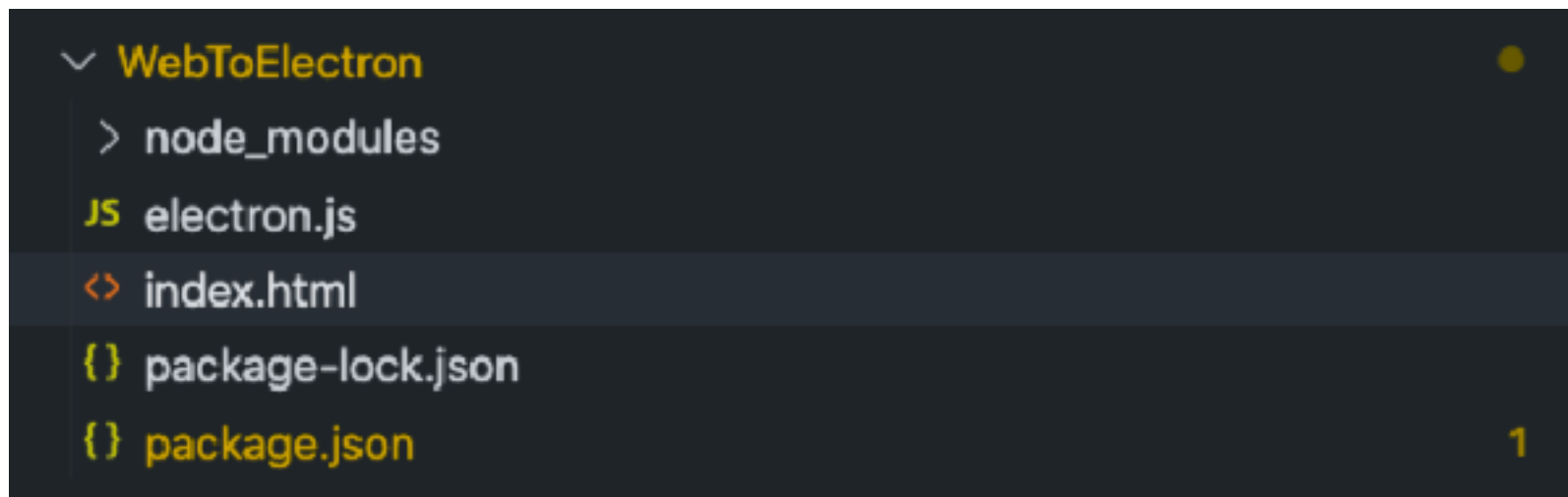
- path 模块
- fs 模块，进行本地文件的读写
- child\_process 模块
- node-ffi 调用原生模块

## **03 Web 端摇身一变桌面端**

# 三步实现掘金桌面应用

Step 1.

- 新增文件夹名为 WebToElectron，此文件夹下新增 **主进程入口文件 electron.js** 、  
**渲染进程入口的 index.html 文件** 以及 package.json
- 接着 npm install 安装 Electron 框架



# 三步实现掘金桌面应用

## Step 2.

进入 package.json 脚本命令中，将我们的主进程 **electron.js** 作为应用程序的入口文件，并且配置启动命令脚本

```
WebToElectron > {} package.json > ...
1  {
2    "name": "WebToElectron",
3    "version": "0.0.1",
4    "author": "彭道宽",
5    "main": "electron.js",
6    "description": "从Web端快速打造成桌面端",
7    "scripts": {
8      "start": "electron ./electron.js"
9    },
10   "dependencies": {
11     "electron": "^11.1.1"
12   }
13 }
```

# 三步实现掘金桌面应用

Step 3.

编写**主进程 electron.js**，在主进程中，通过 `BrowserWindow` 创建渲染进程以及浏览器窗口，通过在浏览器窗口中 `loadURL` 加载远端链接或者是加载本地 **index.html 入口文件**。

```
WebToElectron > JS electron.js > ...
1  const { app, BrowserWindow } = require('electron');
2
3  function createWindow() {
4    // 1. 创建浏览器窗口
5    const courseWindow = new BrowserWindow();
6    // 2.1 加载远端链接
7    courseWindow.loadURL('https://juejin.cn/');
8    // 2.2 也可以载入本地 index.html 文件
9    // courseWindow.loadURL(`file://${app.getAppPath()}/index.html`)
10 }
11
12 app.whenReady().then(() => {
13   createWindow();
14   app.on('activate', function () {
15     if (BrowserWindow.getAllWindows().length === 0) createWindow();
16   });
17 });
```

执行 npm run start 将应用启动，可以看到最终效果图



loadURL 加载入口 index.html



loadURL 加载远端链接

# 如何打包成安装包呢？



electron-builder

Electron Builder 是一个完备的 Electron 应用打包和分发的解决方案，通过 Electron Builder 可以将我们的桌面应用打包成安装包。

**Electron 就是套了个壳？ 它的可玩性在哪？**

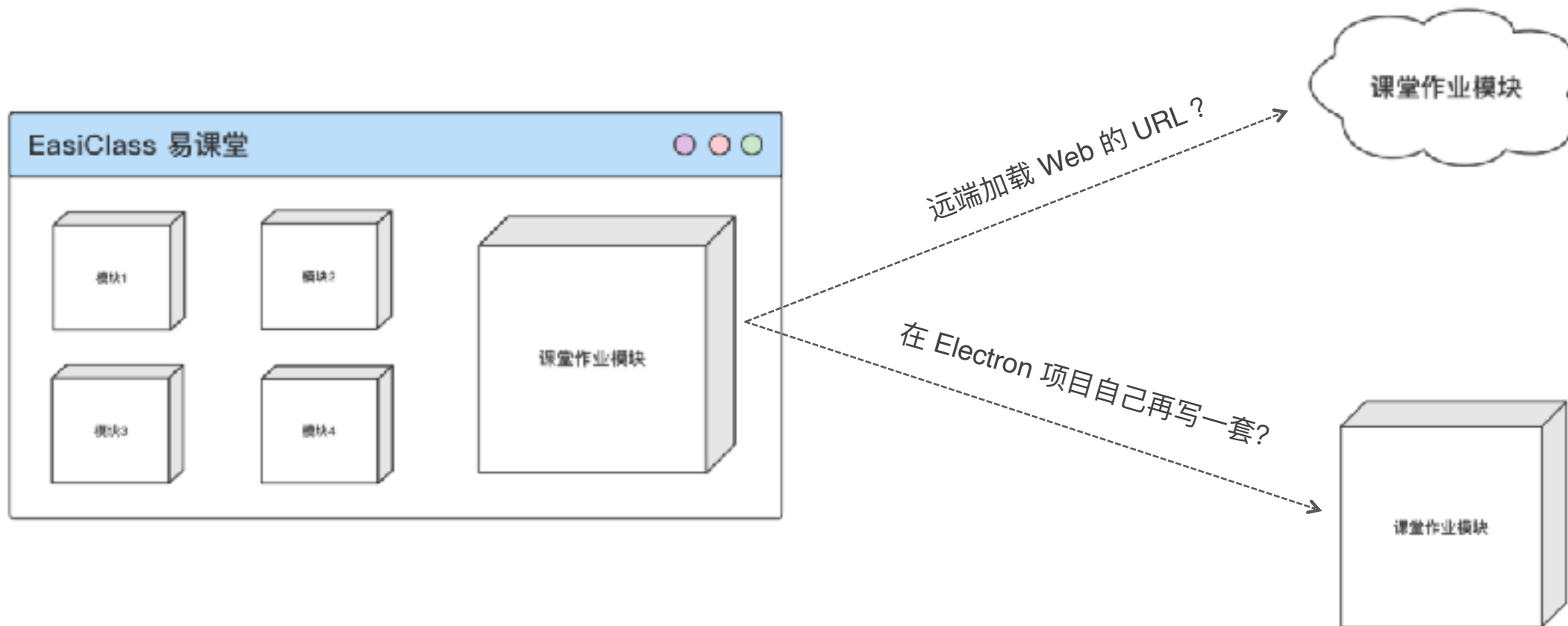


# 04 Electron 与智慧课堂结合

# 场景：

我们在 Web 端有一个课程作业模块，现在产品要求将该模块进行移植到PC端。同时要求：

1. 支持定制快捷键截屏+截图图片本地保存
2. 保证稳定性，断网离线状态下的课程作业打开能正常使用



# 模块移植

模块移植是非常容易的，我们只需要通过 loadURL 加载远端的课程作业模块即可。

```
// 1. 创建浏览器窗口
const courseWindow = new BrowserWindow();
// 2. 加载远端链接
courseWindow.loadURL('https://xxx.xxx.com/app/course');
```

# 定制化快捷键截屏

Electron 可以获取系统底层的能力，通过它提供的 `globalShortcut` 模块可以在操作系统中注册/注销全局快捷键，从而达到我们定制各种快捷键的要求。

```
const { app, globalShortcut } = require('electron')

app.whenReady().then(() => {
  // 注册 CommandOrControl+H 截屏快捷键
  // 当用户按下注册快捷键时， 触发截屏事件
  const ret = globalShortcut.register('CommandOrControl+H', () => {
    console.log('执行快捷键截屏的回调事件....')
  })
})

app.on('will-quit', () => {
  // 注销快捷键
  globalShortcut.unregister('CommandOrControl+H')
})
```

# 如何截屏？

在 Web 端我们常用的一个截屏方案是：通过 html2canvas 将 DOM 节点转成 canvas，再通过 canvas.toDataURL 转成一张 base64 图片  
最终实现我们截屏效果。

**我们期望的是桌面级截屏，非网页截屏**

# 桌面级别的截屏 + 截屏图片本地保存

通过 Electron 提供的 desktopCapturer 模块得到从桌面捕获的音频/视频的媒体源信息

得益于 Electron 内置了 NodeJS 模块，意味着我们可以使用 Node 的特性，通过 fs 文件系统，通过读取指定的文件夹，将图片保存到指定目录下。

```
const fs = require('fs')
const os = require('os')
const path = require('path')
const { desktopCapturer, shell } = require('electron');

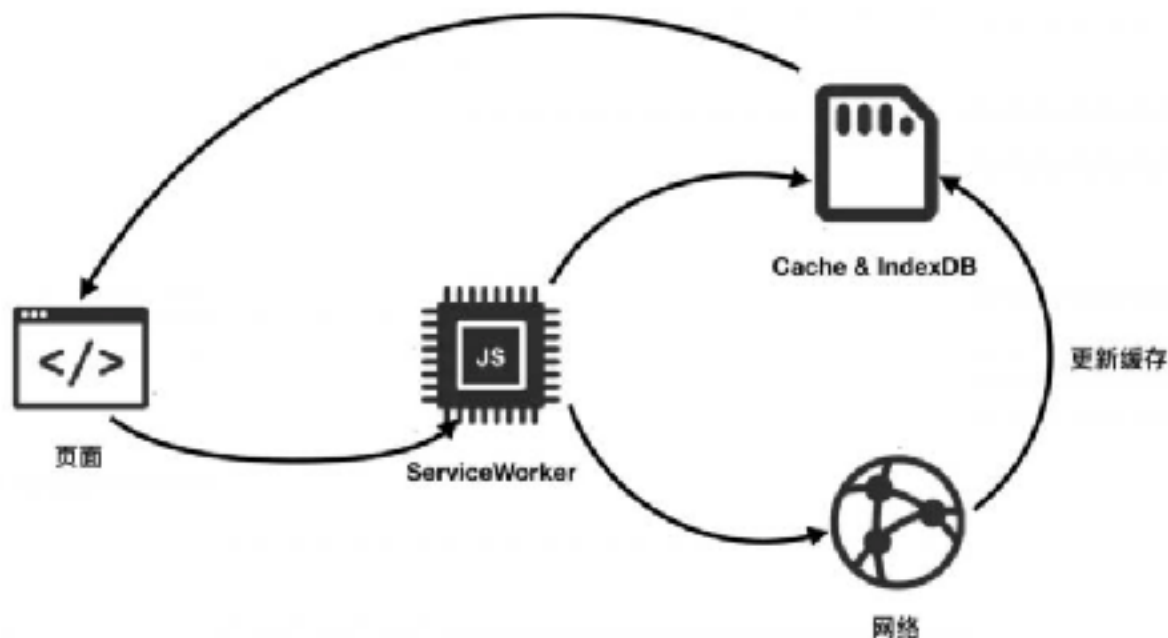
desktopCapturer.getSources({ types: ['window', 'screen'] }).then(async sources => {
  sources.forEach(function (source) {
    if (source.name === 'Entire screen' || source.name === 'Screen 1') {
      // 1. 截图图片的路径
      const screenshotPath = path.join(os.tmpdir(), 'screenshot.png');
      // 2. 将截图图片写入
      fs.writeFile(screenshotPath, source.thumbnail.toPng(), function (error) {
        if (error) return console.log(error)
        // 3. 打开截屏图片预览
        shell.openExternal('file://' + screenshotPath)
        console.log(`截图保存到: ${screenshotPath}`)
      })
    }
  })
})
```

离线状态下可用

# Service Worker

页面初始时是没有 sw，如果没有 sw，页面将会进行注册、安装且激活。什么时候 sw 才生效？

在页面下一次进入时，也就是我们需要重新加载页面，sw 才会接管页面进行控制。



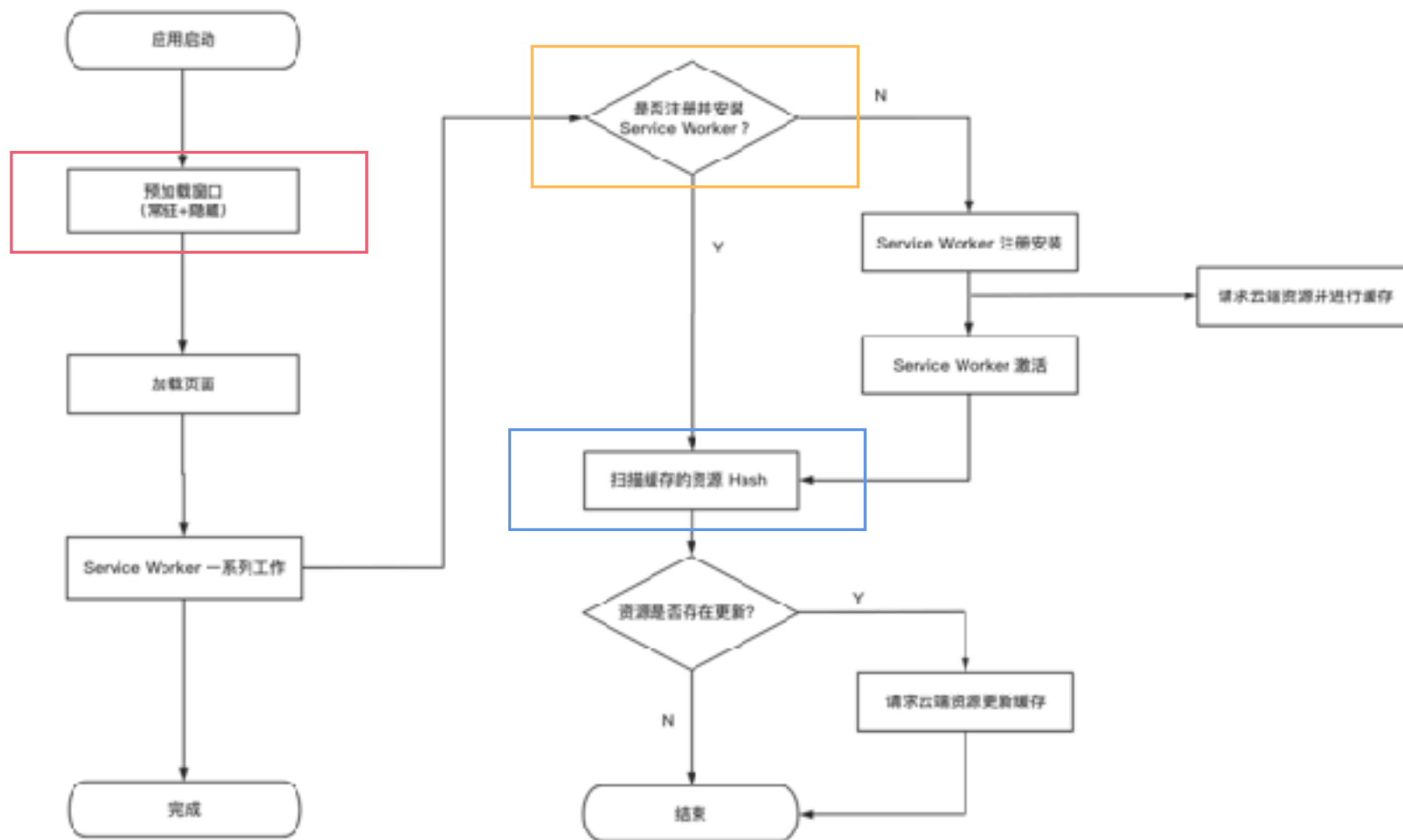
通过 sw 将资源进行缓存，从而减少下次打开页面的网络请求耗时，使得我们的云端web应用可以秒开，并且在离线环境下也变得可用。



# 如何提高体验

# 预加载窗口

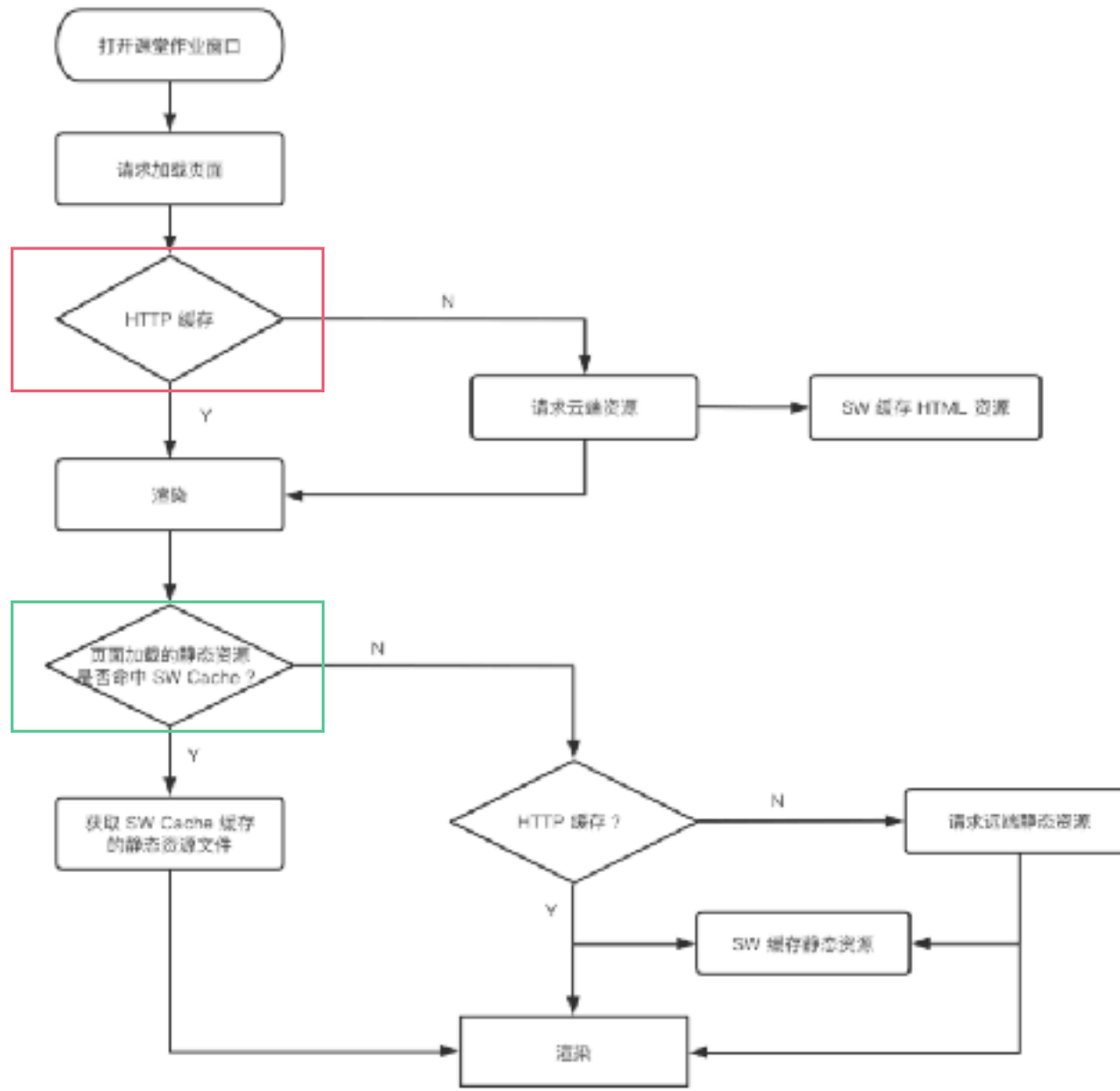
由于 sw 的机制，在第一次只会进行注册、安装、拉缓存，想要缓存生效需要在二次进入页面。  
为了让老师在打开窗口时就能感受到缓存所带来的舒适体验，我们通过预加载窗口提高体验。



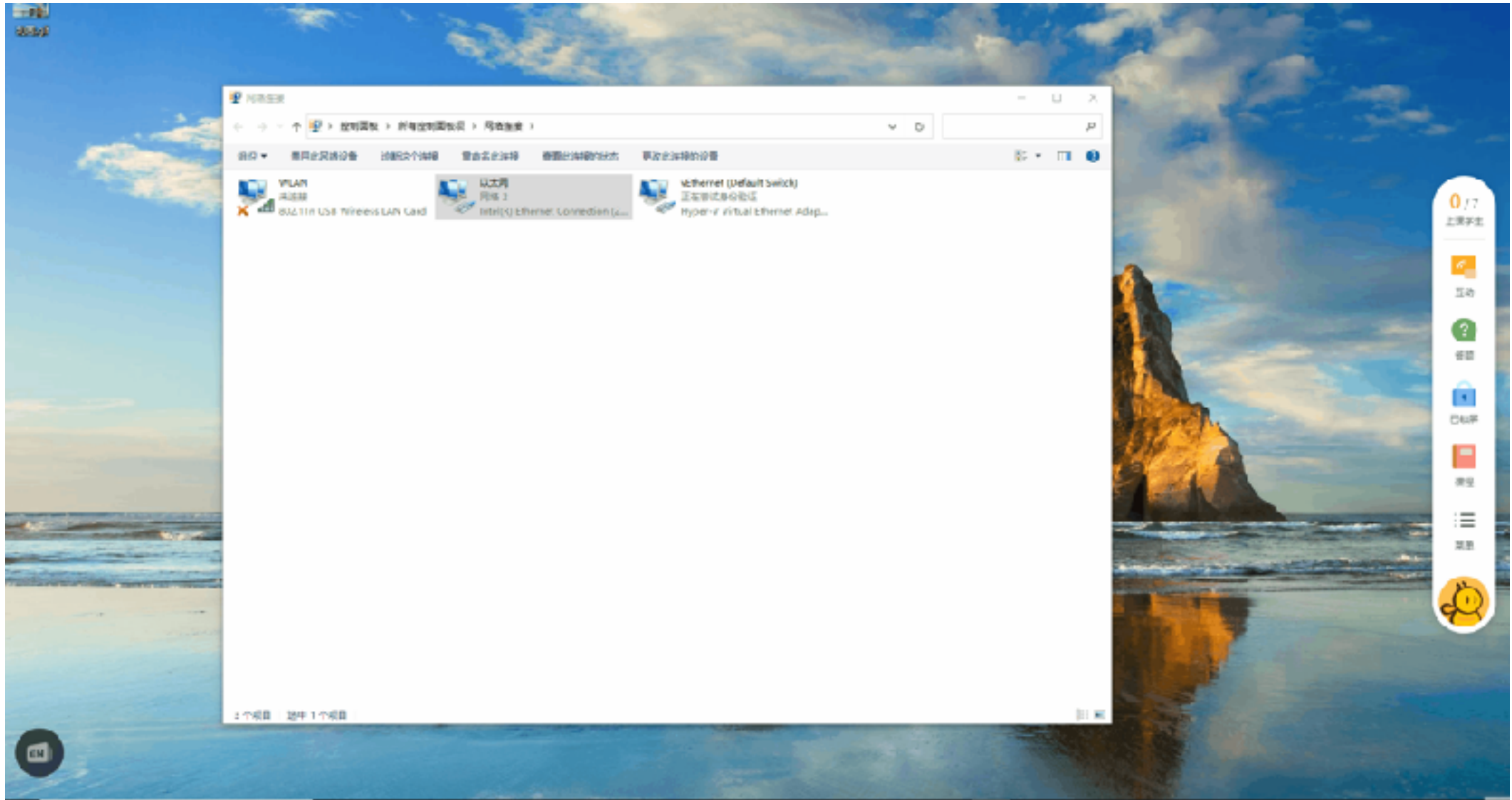
# 老师打开课堂作业窗口（网络稳定状态下）



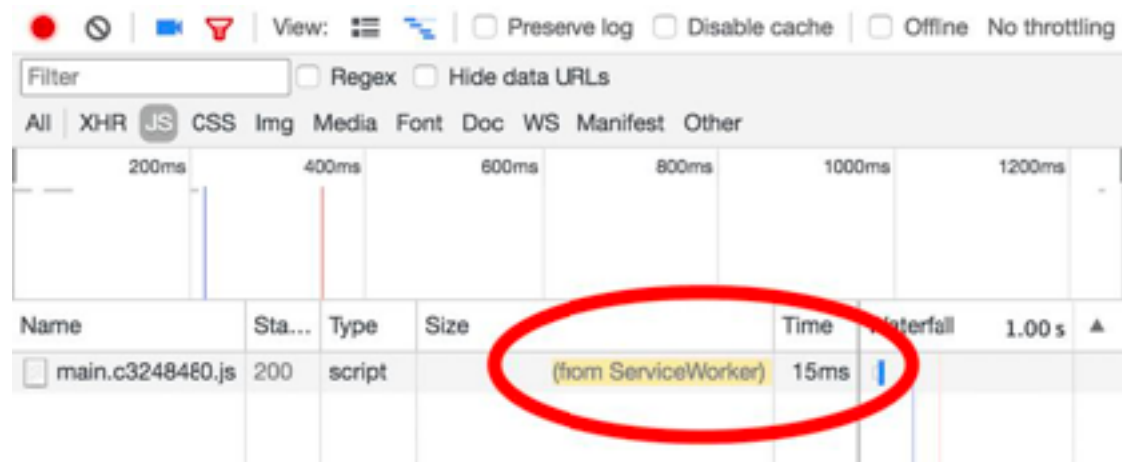
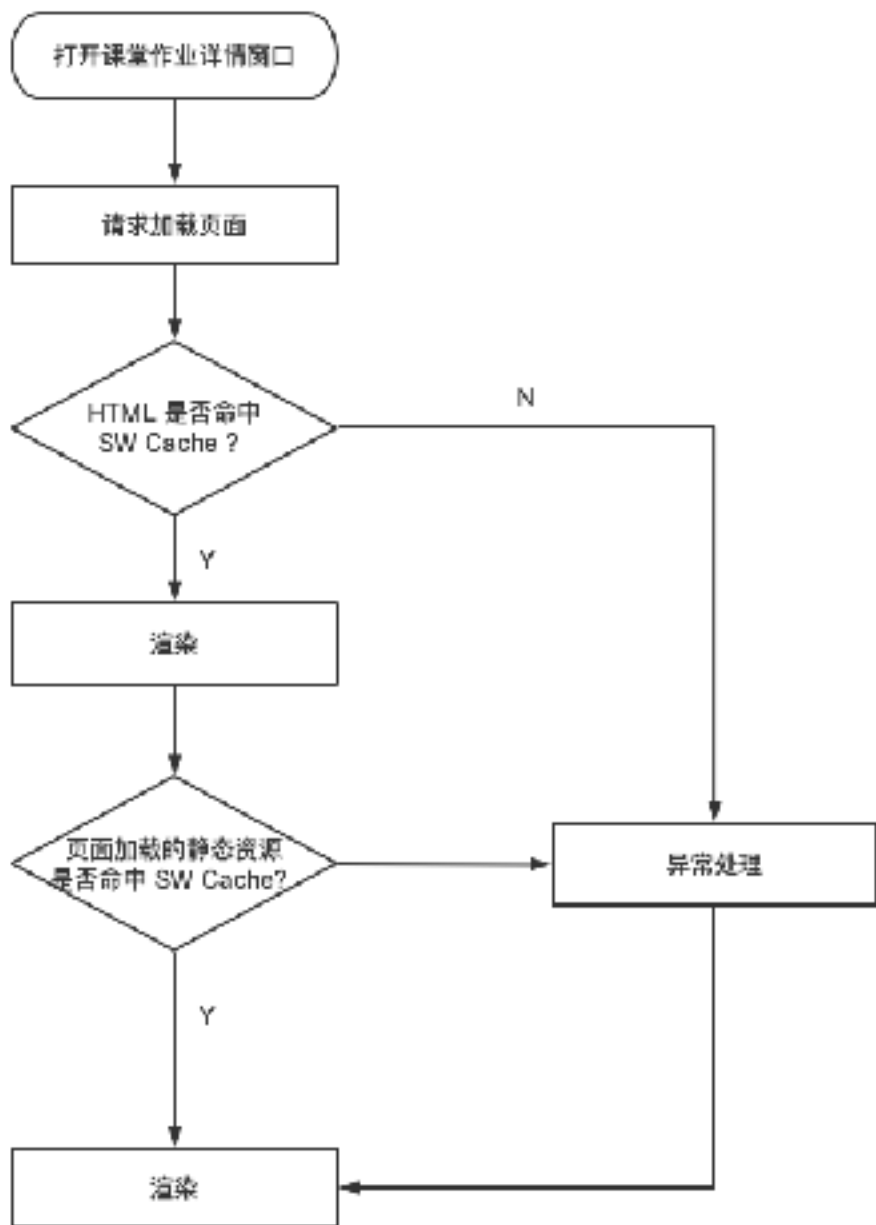
# 老师打开课堂作业窗口（网络稳定状态下）



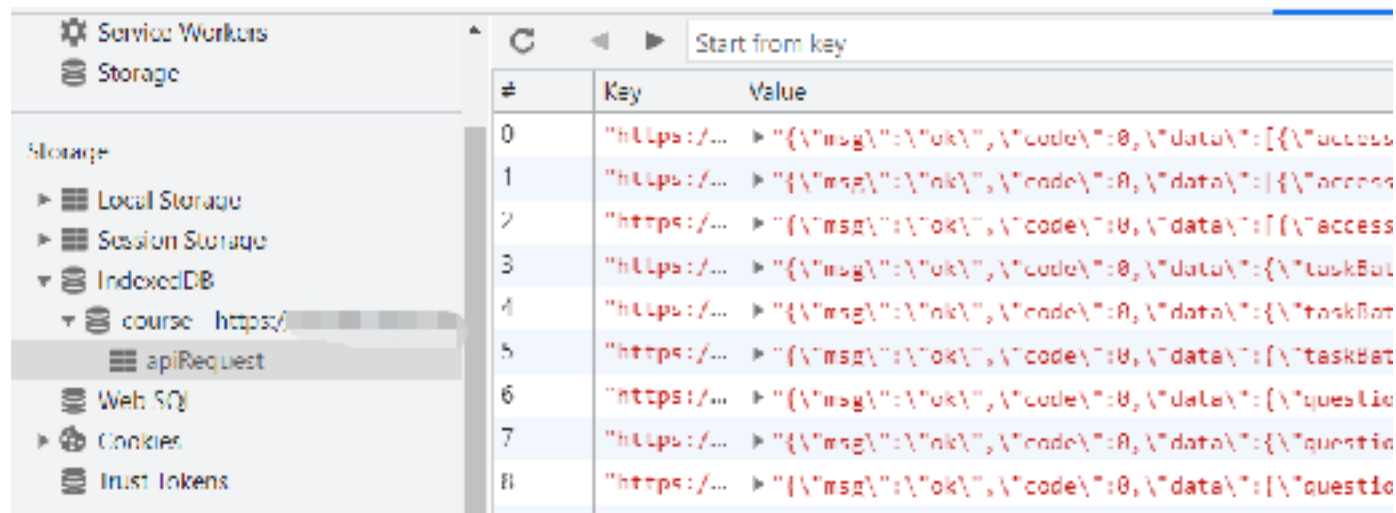
# 老师打开课堂作业窗口（网络不稳定断网状态）



# 老师打开课堂作业窗口（网络不稳定断网状态）

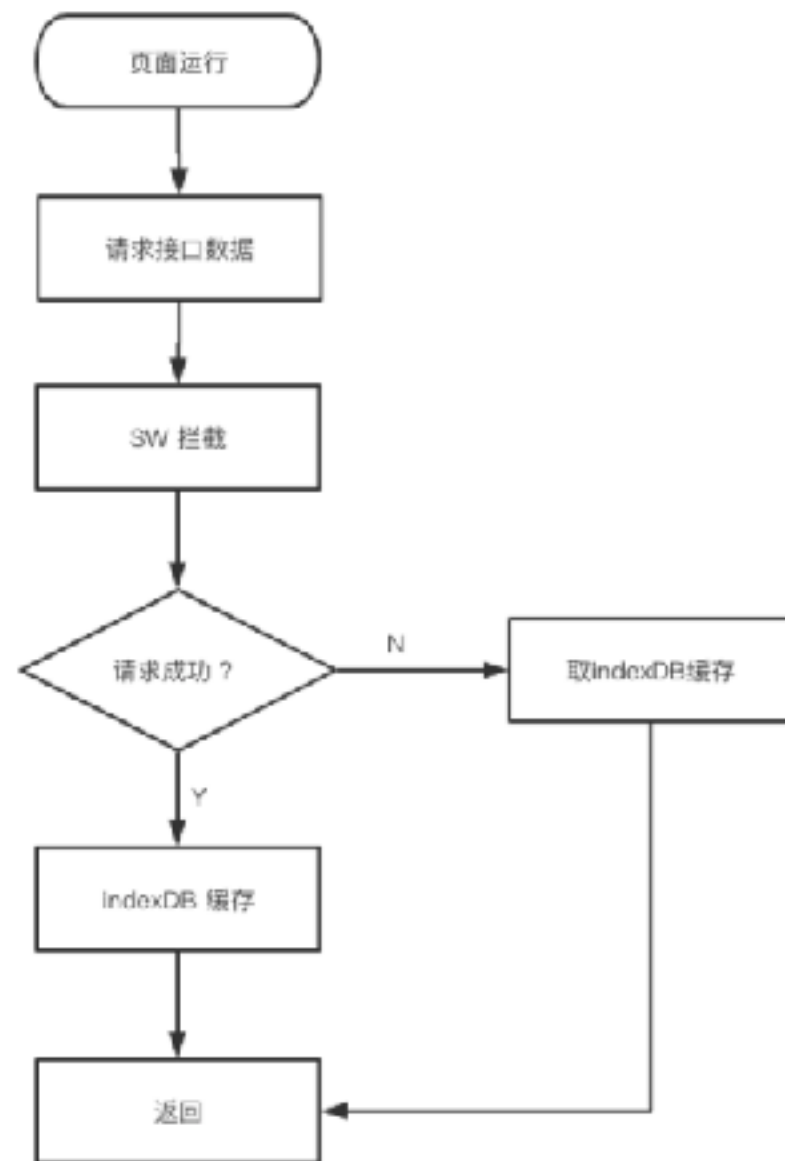


# 页面接口数据的缓存



The screenshot shows the Chrome DevTools Storage Inspector. The left sidebar lists storage areas: Service Workers, Storage, Local Storage, Session Storage, IndexedDB, course, apiRequest, Web SQL, Cookies, and Trust Tokens. The 'Storage' section is expanded, and 'IndexedDB' is selected. The main pane shows a table of IndexedDB data with columns '#', 'Key', and 'Value'. The table contains 9 rows of data, each representing a different API endpoint and its response structure. The 'Value' column shows JSON objects with 'msg', 'code', and 'data' fields.

#	Key	Value
0	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "access": ... } }
1	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "access": ... } }
2	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "access": ... } }
3	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "taskBot": ... } }
4	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "taskBot": ... } }
5	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "taskBot": ... } }
6	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "question": ... } }
7	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "question": ... } }
8	"https://..."	{ "msg": "\u0022ok\u0022", "code": 0, "data": { "question": ... } }





# 小册推荐





# 加入我们

易课堂团队专注于智慧教育相关业务，欢迎小伙伴们加入 🙌

产品主页：<https://class.seewo.com>

团队主页：<https://github.com/SugarTurboS>

团队掘金号：<https://juejin.cn/team/6948219916164022279/posts>

简历投递：[pengdaokuan@cvte.com](mailto:pengdaokuan@cvte.com)



我们团队致力于开源，如 Sugar-Electron 框架，rc-redux-model 中间件等  
如果你对 Electron 感兴趣或对开源有热情，可以一起交流探索



# THANKS

感谢关注

早

## 第二十八届前端早早聊大会

数字孪生 | 3D 地图 | 并行计算 | 渲染 | WebGL 前景



掘金

6月26日  
全天直播

2021 全年行程

1/9 自由职业/副业

1/23 研发团队管理

2/5 小程序/组件化

2/27 页面搭建场

3/20 前端面试

4/10 前端摸CI/CD

4/24 前端摸算法

5/09 前端摸述职

5/15 前端摸互动

6/5 跨端 Flutter

6/12 前端摸编译

6/26 前端 WebGL

7/10 WebAssembly

7/24 前端摸 BFF

8/14 前端摸 AI

8/28 前端摸安全

9/11 Serverless

9/25 前端摸 IDE

10/16 前端摸监控

11/20 玩 Node.js

稼力	贝壳找房	资深 WebGL 工程师	《如何在 WebGL 的生态现状中尝试最佳实践》	09:00
图卓	阿里云	「DataV 团队」 地图数据可视化	《如何以 WebGL 构建数字孪生应用》	10:00
慎思	蚂蚁集团	「Oasis 引擎」 核心开发者	《如何实现 WebGL 的高性能渲染管线》	11:00
兆康	奇安信	奇安信雷尔平台 核心开发	《如何利用 WebGL 构建 3D 地图可视化引擎》	13:00
鑫月	奇安信	奇安信雷尔平台 核心开发	《如何实现 Web 大屏 3D 主视觉的关键技术》	14:00
泽辉	小米	「体验效能」 前端可视化方向	《如何用 WebGL 尝试全流程的活动交互实践》	15:00
剑鑫	阿里 UC	「Lottie-pixi」 核心开发	《如何借助 WebGL 玩转 Lottie 动画》	16:00
沧东	蚂蚁集团	「体验技术部」 图可视化方向	《如何玩转 WebGL 的并行计算》	17:00
木的树	美团	「WebGL」 技术专家	《如何从 0 到 1 快速学习 WebGL》	18:00

# 前端早早聊大会直播

## 2020 PK 2021

已举办 16 期 100 场

计划举办至少 20 期 140 场

1/11 前端转管理	6/20 前端跨端跨栈
2/29 前端搞基建	6/27 前端女生专场
3/28 前端搞搭建	7/18 前端搞可视化
4/11 前端搞规划	8/15 前端搞构建
4/25 前端搞监控	8/29 前端成长晋升
5/16 Serverless	9/26 前端搞报表
5/30 前端搞微前端	10/17 前端搞组件
5/31 前端搞面试	11/21 前端搞框架
6/13 前端搞文档	12/26 前端搞性能

1/10 前端搞副业	5/29 工程化/Flutter
1/23 前端搞管理	6/05 跨端 Flutter
2/06 前端搞小程序	6/19 福利专场
2/27 可视化搭建	6/26 前端搞 WebGL
3/06 前端搞搭建	6/27 前端搞微前端
3/20 前端搞面试	7/17 前端搞可视化
3/27 菜鸟大前端	7/24 前端搞 BFF
4/10 CI/CD	8/28 前端搞安全
4/24 前端搞算法	9/11 Serverless
5/09 前端搞述职	9/25 前端搞 IoT
5/15 前端搞互动	10/16 前端搞监控
11/20 前端搞 IDE	12/11 玩转 Node.js

(以实际举办为准, 行程/话题/场次会做动态调整)

早

一招鲜走天下 组合拳闯四海  
单主多讲群 听得懂抄得走

# 前端早早聊大会

## 2021年票

解锁 2021 年 140 场干货技术直播

单场大会用户

年票 VIP 用户

平均每期 77 元	平均每期 25 元
平均每场 12 元	平均每场 5 元
-	不限次数发布招聘
-	获得优质简历模板
-	Scott 简历指导及内推
-	2022 年票 <= 7 折
-	其他神秘福利...

¥660

每个月有直播  
每一场有录播

