开发文档

技术栈

脚本: typescript + vue + tsx

软件: typescript + electron + vue + ant design vue + playwright

项目结构

```
+ packages
    + core                  # 油猴脚本库
    + scripts                # 软件自动登录库
    + common                # 公共库 (一些工具方法)
    + web                    # 使用 vue3 + ts + ant design vue 构建的 electron
渲染进程
    + app                    # electron 主进程
+ scripts                   # 项目打包构建 gulp 文件
- webpack.config.js          # webpack 打包配置 :  打包  core 油猴脚本库 作为浏
览器端环境
```

package.json 命令介绍

```
{
        "build:core": "gulp -f ./scripts/build-core.js", /** core构建,  文件夹
清理，css文件打包等构建流程 */
        "build:app": "pnpm build && gulp -f ./scripts/build-app.js",  /** 打包
web 和 app 的软件端 */
        "build": "gulp -f ./scripts/tsc.js", /** 为每个 typescript 子项目运行
tsc 命令 */
}
```

项目构建

```
# 全局安装 pnpm ，如果已经安装，则无需执行，请勿安装7.x以上版本。
npm i pnpm@6.32.6 -g
```

```
# 下载项目
git clone https://github.com/enncy/online-course-script.git ocs
# 进入目录
cd ocs
# 使用 pnpm 安装依赖
pnpm i -w
# 构建项目 (为每个子项目进行 tsc 构建)
pnpm build
```

项目运行

接下来打开 2 个终端，分别执行：

```
# 进入 web 渲染进程
cd packages/web
# 运行 vue 项目
npm run dev
```

```
# 进入 app 主进程
cd packages/app
# 运行 electron 软件
npm run dev
```

软件打包

```
pnpm build:app
```

脚本打包

```
pnpm build:core
```

本地调试

在本地浏览器安装油猴，并且 `@require` 引用本地打包好的文件，即可本地调试。

油猴 API

```
// @require      file://E:\xxx\xxx\ocs\dist\index.min.js
// @resource     OCS_STYLE file://E:\xxx\xxx\ocs\dist\style.css
```

项目引入

> 请自行根据版本调整链接

官方CDN: https://cdn.ocs.enncy.cn/dist/3.7.4/index.min.js

JSD: https://cdn.jsdelivr.net/npm/ocsjs@3.7.4/dist/index.min.js

JSD-core: https://cdn.jsdelivr.net/npm/ocsjs@3.7.4/lib/src/core/index.min.js

API

类型

OCR

> 使用 tesseract.js 文本识别，解决自定义字体显示繁体字, 乱码的问题

```
const ocr = new OCS.OCR(options)
```

- `opions` API 请看官方文档 : https://github.com/naptha/tesseract.js/blob/master/docs/api.md
- `OCS.OCR.suit(element)` : 将元素自适应，改变大小，间距，方便OCR识别
- `OCS.OCR.unsuit(element)` : 将元素字体还原成原来的样子。
- `ocr.recognize(element)` : 识别文本

例子

```
// 创建OCR对象， OCR 对象只需创建一次，每次识别只需调用 recognize 方法即可
const ocr = new OCS.OCR({
    /**
      加载数据文件，这里请配置你的对象存储路径，或者任意CDN，如果不设置路径必须使用梯子才能访问到数据。
      例如 https://cdn.xxx.cn/tessdata
      https://cdn.xxx.cn/tessdata 路径下面放置 chi_sim.traineddata.gz
```

```
      chi_sim.traineddata.gz 下载地址请到 http://tessdata.projectnaptha.com/
下载
      下载例子
      eng : https://tessdata.projectnaptha.com/4.0.0/eng.traineddata.gz
      chi_sim :
https://tessdata.projectnaptha.com/4.0.0/chi_sim.traineddata.gz
    */
    langPath: 'https://cdn.xxx.cn/tessdata' // 只是例子
});
// 加载OCR数据文件
await ocr.load()
```

```
// 获取元素
const el = document.querySelector(...)
// 识别文本
const text = await ocr.recognize(OCS.OCR.suit(el))
// ...
```

```
// 多个元素
const els = [...]
for (const el of els) {
    const text = await ocr.recognize(OCR.suit(el));
    // ...
}
```

油猴使用

```
// ==UserScript==
// @name         文本破解例子
// @namespace    http://tampermonkey.net/
// @version      0.1
// @author       You
// @match        *://*.chaoxing.com/*
// @description  下面是引入 OCS 文件
// @require      https://cdn.jsdelivr.net/npm/ocsjs@3.6.4/dist/index.min.js
// ==/UserScript==

(async function() {
    'use strict';
    /**
    下面是OCS网课助手提供的OCR类

    你也可以参考 OCS 源码， https://github.com/enncy/online-course-
```

```
script/blob/3.0/packages/core/src/core/ocr.ts
    或者 https://github.com/naptha/tesseract.js 提供 的OCR API 等，进行OCR的实现。

    */

    // 判断是否有加密字体
    const fonts = document.querySelectorAll(".font-cxsecret")
    if(fonts.length){
        // 创建OCR对象， OCR 对象只需创建一次，每次识别只需调用 recognize 方法即可
        const ocr = new OCS.OCR({
            /**
              加载数据文件，这里请配置你的对象存储路径，或者任意CDN，如果不设置路径必须使用梯子才能访问到数据。
              例如 https://cdn.xxx.cn/tessdata
              https://cdn.xxx.cn/tessdata 路径下面放置 chi_sim.traineddata.gz
              chi_sim.traineddata.gz 下载地址请到 http://tessdata.projectnaptha.com/ 下载
              下载例子
              eng : https://tessdata.projectnaptha.com/4.0.0/eng.traineddata.gz
              chi_sim : https://tessdata.projectnaptha.com/4.0.0/chi_sim.traineddata.gz
            */
            langPath: 'https://cdn.xxx.cn/tessdata' // 只是例子 ，这里为啥不提供我的CDN呢，因为一堆人用的话估计要炸，所以大家还是用自己的服务器或者 对象储存+CDN吧。
        });
        // 加载OCR数据文件
        await ocr.load()
        for (const font of fonts) {
            // 字体适应
            OCS.OCR.suit(font)
            // 识别
            const text = await ocr.recognize(font);
            // 替换文本
            font.innerHTML = text;
        // 还原字体
        OCS.OCR.unsuit(font)
        }
    }

    // Your code here...
})();
```

DefineScript

> 脚本声明，油猴脚本的核心

- `name`: `string` 脚本名
- `routes`?: `Array<ScriptRoute>` 脚本路由
- `panels`?: `Array<ScriptPanel>` 脚本面板

ScriptRoute

> 脚本路由， 类似油猴的 include， 匹配页面路径执行脚本

- `name`: `string` 脚本名字
- `url`: `string | RegExp | GlobPattern | string[] | RegExp[] | GlobPattern[]` 页面路径匹配
  - 使用例子
    - **example.com** : 匹配所有包含 example.com 的路径
    - https://example.com/ : 只匹配 https://example.com/ 页面
- `onload`?: `Function` 等待页面加载完毕调用
- `start`?: `Function` 页面加载时 立即执行
- `priority`?: `number` 执行优先级, 默认0

ScriptPanel

> 脚本面板， 面板的组成对象类型。

- `name`: `string` 名字
- `url`: `string | RegExp | GlobPattern | string[] | RegExp[] | GlobPattern[]` 页面路径匹配， 与 `ScriptRoute.url` 类型一致
- `el`: `() => DefineComponent<any> | VNode | HTMLElement | string;` 支持 3种 元素 `VNode` ,`DefineComponent` , `string`
- 例子:

- el : ()=> \xxx\
- el : ()=> h("span", "xxx")
- `children`?: `Array<ScriptPanelChild>` 其余的子面板
- `priority`?: `number` 优先级, 默认0
- `default`?: `boolean` 当页面没有任何面板时，是否显示

AnswererWrapper

> 题库配置器的参数

- `url`: `string` 答题器请求路径
- `name`: `string` 题库名字
- `homepage`?: `string` 题库网址
- `data`?: `Record<string, string>` 传递的参数, get 请求将会添加到 url 后面， post 请求会生成请求体
- `method`: `"post" | "get"` 默认 `get`，请求方法
- `contentType`: `"json" | "text"` 默认 `json`，定义 handler 中的参数类型
- `type`: `"fetch"|"GM_xmlhttpRequest"` 默认 `fetch`，请求类型, `fetch` 是用浏览器原生API， `GM_xmlhttpRequest` 使用油猴自带API，可进行跨域。
- `headers`?: `Record<string, string>` 默认 `{}`，
- `handler`: `string` 此选项是个字符串， 使用 Function(string) 构造方法进行解析生成方法 方法传入一个参数：请求获取到的文本， 可以使用 contentType 定义文本类型 对返回的数据进行自定义解析 并且返回一个数组：`[题目, 答案]` 或者二维数组：`[[题目1, 答案1],[题目2, 答案2], ...]` 如果搜不到则返回 undefined 例子: `return (res)=> res.code === 0 ? undefined : [res.question, undefined]` 如果错误时你想带点提示，可以 `return (res)=> res.code === 0 ? ['抱歉找不到答案',undefined] : [res.question, undefined]`

方法

`start(...)`

> 加载 OCS

参数

- `draggable`:`boolean` 是否开启拖拽功能
- `scripts`：`Array<DefineScript>` 需要运行的脚本

`defineScript(...)`

> 定义一个脚本

参数：

- `options`:`DefineScript`

例子

```
export const ExampleScript = defineScript({
    name: "脚本例子",
    routes: [
        {
            name: "页面加载后运行的脚本",
            url: "**example.com/video.html**",
            async onload() {},
        },
        {
            name: "页面加载前运行的脚本",
            url: "**example.com/work.html**",
            async start() {},
        },
    ],
    panels: [
        {
            name: "脚本助手",
            url: "https://www.example.com",
            el: () => ...,
        },
        {
            name: "脚本助手，带有子面板",
            url: "**example.com**",
            el: () =>...,
            children: [
                {
                    name: "子面板",
                    el: () => ...,
                },
            ],
        },
    ],
```

```
});

// 运行脚本
OCS.start({
    scripts: [ExampleScript]
})
```

定义

OCSWorker 答题器

> 强大的答题处理器，什么你还在用dom操作去实现每个脚本的答题？我只需要往OCSWorker传入几个参数即可。

具体参数请看代码

实例：https://github.com/enncy/online-course-script/blob/3.0/packages/core/src/script/zhs/work.ts#L117-L169

你可能会说这不是得50行嘛，确实，但是除了 elements 中的和其他几个参数，剩下的只需要复制粘贴之前的OCSWorker即可，逻辑都是相同的。

AnswererWrapper 题库配置

> OCS 提供了强大的 `题库配置解析器`，你可以对接大多数的题库进行使用

类型 : Array\<`AnswererWrapper`>

简单例子

```
// 假设有一个接口 : https://example.com/search?title=1+2,2+3
// 此接口返回 {code: 1, data: { answers: [3 , 5] , title:'1+2' }, msg:'成功'}

defaultAnswerWrapperHandler(
    {
        // 题目
        title: '1+2,2+3',
        // 题目类型
        type: 'single'
    },
    [
        // 可以有多个构造器，最终通过 answerPath 一起合并到一个列表并返回
```

```
        {
            url: "https://example.com/search/"// url 也可以进行解析 ${title} ,例如 https://example.com/search/${title}/,
            method: "get",
            contentType: "json",
            data: {
                title: "${title}", // 1+2,2+3,
                abc: "123", // 自定义参数,
                platform: "${platform}" // 解析本地 localStorage.OCS 参数,
                upload: "${setting.cx.work.upload}" // 解析本地 localStorage.OCS 参数
            },
            handler: `return (res)=> res.code === 0 ? undefined : [res.data.title, res.data.answers[0]]`  // 取第一个结果
        },
    ]
);
```

注意:

- 文本为 json 数组， 数组意味着你可以配置多个题库
- `${xxx}` 是变量占位符
  - 可以使用在 `data` 和 `url` 字段中
  - 可以解析
    - $title: 题目标题
    - $type: 题目类型
    - $root: 题目元素，具体元素请看各脚本的代码实现。

所以最终填写的 `题库配置` 为： （不要使用这个，这个只是例子！！！！）

```
[{"url":"https://example.com/search/","method":"get","contentType":"json","data":"(res)=> res.code === 0 ? undefined : [res.data.title, res.data.answers[0]]"}]
```

变量

`definedScripts`：Array\<DefineScript>

> 内置定义的全部脚本列表

`app` : App

> vue对象

`panel` : HTMLElement

> OCS面板的元素对象

`store` : OCSStore

> OCS存储，存储一些临时元素，以及本地存储数据。

标签: 开发文档 API

编辑此页