网易人力资源部

“简历助手插件”的研究分析与实践

姓名：贺世宇

工号：H17018

时间：2019年05月31日

目录

演示——“网易简历助手”的功能效果

Chrome Extensions的基础知识

浏览器插件

也叫“扩展程序”。它可以使用户在访问网页时，具备更多不平凡的功能。因为它除了可以使用浏览器、页面提供的**所有API**，还具有一些特殊行为。

=

浏览器行为：**监听标签页/窗口等事件**、LocalStorage

网页行为：**XMLHttpRequest**、**操作DOM**

特殊行为：**无限跨域请求**、**消息通信**、cookies读写、storage

这些行为都需要在 **清单列表（manifest.json）** 里声明

Chrome Extensions的基础知识

插件界面

(popup.html / popup.js)

后台脚本

(background js)

内容脚本

(content script)

```
{
    "name": "网易简历助手",
    "version": "1.0.0",
    "description": ""简历助手"是由EHR-招聘产品团队开发的、用于帮助HR将招
    "author": "Netease EHR",
    "manifest_version": 2,
    "icons": {
        "16": "logo.png",
        "48": "logo.png",
        "128": "logo.png"
    },
    "permissions": ["*://*/", "storage", "contextMenus", "clipboa
    "page_action": {
        "default_icon": "logo.png",
        "default_title": "网易简历助手",
        "default_popup": "pages/index.html"
    },
    "background": {
        "persistent": false,
        "scripts": ["js/background.js"]
    },
    "options_page": "pages/options.html",
    "content_scripts": [{
        "js": ["js/content.js"],
        "run_at": "document_start",
        "matches": ["*://ehire.51job.com/Candidate/ResumeView*"],
        "all_frames": false
    }],
    "content_security_policy": "script-src 'self' 'unsafe-eval';
    "web_accessible_resources": ["js/main.js"],
    "homepage_url": "https://zhaopin.netease.com"
}
```

清单文件（manifest.json） —— 为“插件”赋予能力

各部分JS的职责

Chrome插件的JS主要可以分为这4类：background js、popup js、contentscript、injected script。

JS种类	可访问的API	目标页DOM访问情况	目标页JS访问情况	直接跨域
background js	可访问绝大部分API	✘	✘	✔
popup js	可访问绝大部分API	✘	✘	✔
content script	只能访问插件的部分API	✔	✘	✘
injected script	和普通JS无任何差别， 不能访问插件所有的API	✔	✔	✘

Live in isolated world

“网易简历助手”的设计思路

分 析

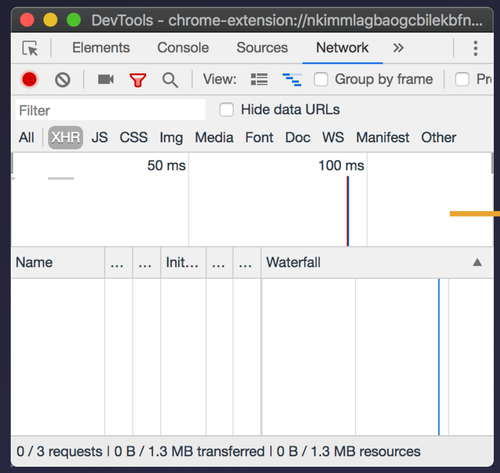
功能需求：

HR打开51Job的简历页时，插件**自动弹出**。他可以快速了解候选人**是否已在人才库**中；也可以将页面上的简历**转成图片**，**直接保存**到社招后台。

“网易简历助手”的设计思路

分 析

JS种类	可访问的API	目标页DOM访问情况	目标页JS访问情况	直接跨域
background js	可访问插件绝大部分API	✘	✘	✔
content script	只能访问插件的部分API	✔	✘	✘
injected script	和普通JS无任何差别，不能访问插件所有的API	✔	✔	✘



background.js：
1、监听标签页
2、跨域发送HTTP请求
3、和content script通信

content script：
1、在目标页面执行
2、嵌入插件界面
3、获取网页HTML
4、和background js、injected script通信；

injected script（插件界面）：
1、自动弹出
2、展示响应的数据
3、接收用户的行为
4、和content script通信

“网易简历助手”的设计思路

分 析

- 各部分JS的通信

“网易简历助手”的设计思路

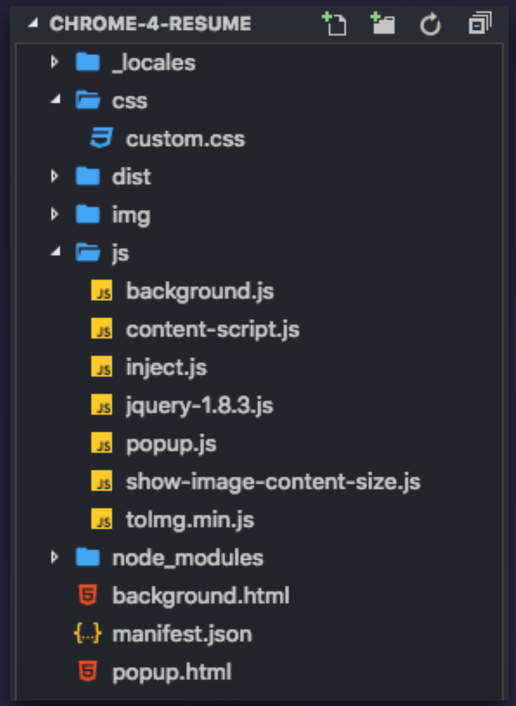
分 析

前端模块化

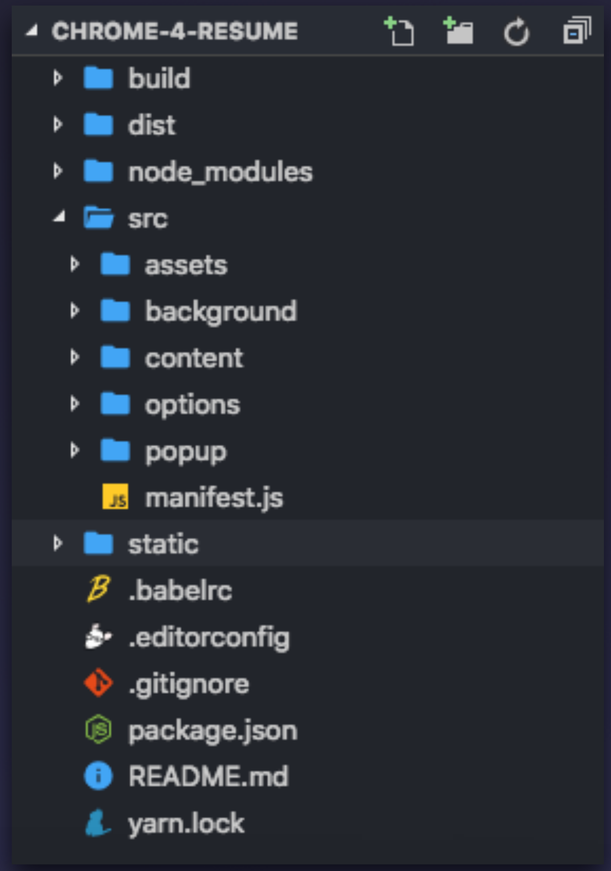
采用了基于Vue.js && webpack打包

请求工具：Axios

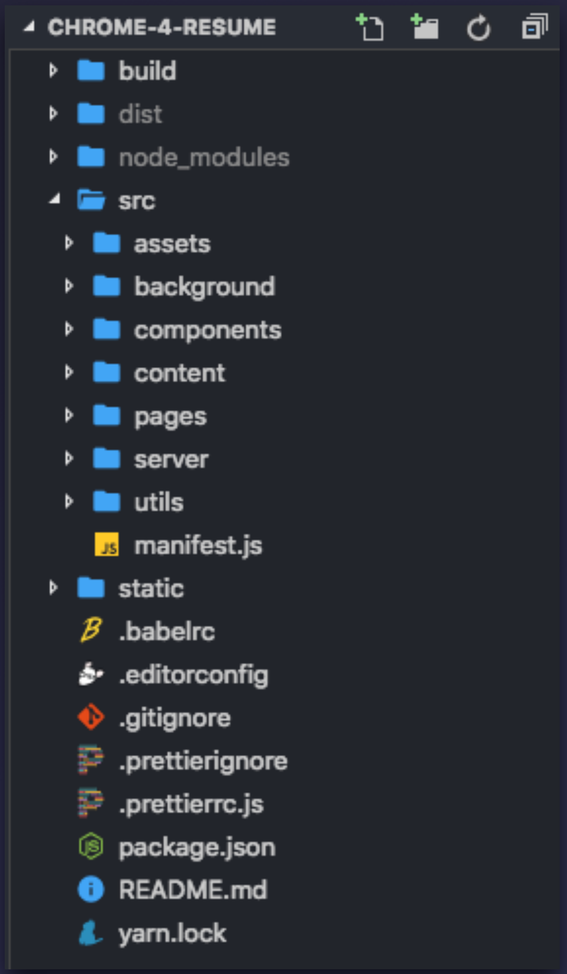
格式化工具：Prettier



V0



V1



V1.1

开发中遇到的难点 && 总结

实 践

background js监听tabs事件

假设每次打开简历页都只会进行 一次 解析请求，需要使用到chrome.tabs的API，并且考虑到HR打开简历页时的每种情况。

事件
onCreated
onUpdated
onMoved
onSelectionChanged
onActiveChanged
onActivated
onHighlightChanged
onHighlighted
onDetached
onAttached
onRemoved
onReplaced
onZoomChange

onActivated

Fires when the active tab in a window changes. Note that the tab's URL may not be set at the time this event fired, but you can listen to onUpdated events so as to be notified when a URL is set.

addListener

```
chrome.tabs.onActivated.addListener(function callback)
```

Parameters

function	callback	The *callback* parameter should be a function that looks like this: `function(object activeInfo) {...};`

object	activeInfo		
		integer	tabId — The ID of the tab that has become active.
		integer	windowId — The ID of the window the active tab changed inside of.

onUpdated

Fired when a tab is updated.

addListener

```
chrome.tabs.onUpdated.addListener(function callback)
```

Parameters

function	callback	The *callback* parameter should be a function that looks like this: `function(integer tabId, object changeInfo, Tab tab) {...};`

integer	tabId	
object	changeInfo	Lists the changes to the state of the tab that was updated.

string	(optional) status	The status of the tab. Can be either *loading* or *complete*.
string	(optional) url	The tab's URL if it has changed.
boolean	(optional) pinned	The tab's new pinned state.
boolean	(optional) audible	Since Chrome 45. The tab's new audible state.

开发中遇到的难点 && 总结

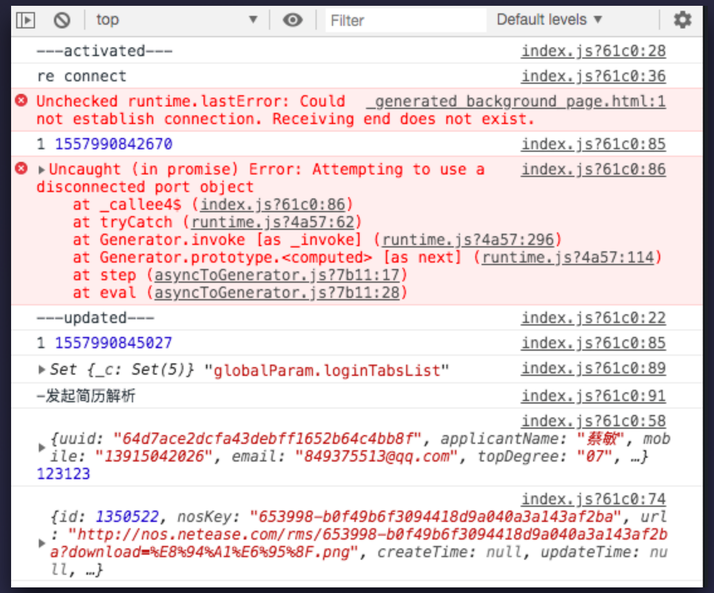
实 践

background js监听tabs事件

	onActivated	onUpdated
打开新tab（跳转）	✔（先）	✔（后）
打开新tab（不跳转）	✔（后）	✔（先）
在当前tab打开/刷新		✔
标签之间切换时	✔	

① **使用onActivated事件时，要考虑页面的content script是否已加载完成。（DOMContentLoaded）**

② **使用onUpdated事件时，要考虑未登录时，切换回可重新解析；还要考虑页面的加载状态。（有两种，否则会重复触发）**



content script未加载完
而导致的“长连接”port不存在

访问url -> 触发updated事件 -> uploading -> 访问url结束 -> 触发updated事件 -> Complete

开发中遇到的难点 && 总结

实 践

Content script的注入时机

```
content_scripts: [
    {
        js: ['js/content.js'],
        run_at: 'document_start',
        matches: [`*://ehire.51job.com/Candidate/ResumeView*`],
        all_frames: false
    }
],
```

有三种方式：

Name	Type	Description
document_idle	string	*Prefered.* Use "document_idle" whenever possible. The browser chooses a time to inject scripts between "document_end" and immediately after the window.onload event fires. The exact moment of injection depends on how complex the document is and how long it is taking to load, and is optimized for page load speed. Content scripts running at "document_idle" do not need to listen for the window.onload event, they are guaranteed to run after the DOM is complete. If a script definitely needs to run after window.onload, the extension can check if onload has already fired by using the document.readyState property.
document_start	string	Scripts are injected after any files from css, but before any other DOM is constructed or any other script is run.
document_end	string	Scripts are injected immediately after the DOM is complete, but before subresources like images and frames have loaded.

开发中遇到的难点 && 总结

Content script的注入时机

Chrome extension *Content scripts* (run from a manifest.json) that are run at document_start, *do* fire before document.readyState[Doc] has reached interactive -- which is the earliest you want to start messing with most page elements.

However, you can inject most <script> nodes right away if you wish. Just not to document.head or document.body because they don't exist yet.

DOM中唯一存在的只有<html>

If you are adding or modifying *other* DOM elements, in a script running at document_start, wait until the DOMContentLoaded event like so:

```
document.addEventListener('DOMContentLoaded', fireContentLoadedEvent,

function fireContentLoadedEvent () {
    console.log ("DOMContentLoaded");
    // PUT YOUR CODE HERE.
    //document.body.textContent = "Changed this!";
}
```

document_start + DOMContentLoaded 对比 document_end，速度更快。

前者会使得css优先加载，随后DOM加载完后再执行js；

后者使得css、js都会等到DOM加载完后才执行。

开发中遇到的难点 && 总结

①、chrome API 大部分都是异步操作、且参数不能传多（包括属性名）。若需要多个并发且按顺序返回时，可以借用Promise.all

②、tabID在浏览器中不会冲突，且在同一个tab下更换url也不会改变tabID；

③、permissions里的域名设置为无限跨域时，发布审核会更严格；且版本迭代时，会通知给使用者跨域列表的变更。

④、Chrome浏览器不允许开发者模拟popup.html自动弹出

⑤、chrome从v38版后不会隐藏插件图标，但能置灰（无论page_action还是browser_action），网上有些教程已经过时了

未来支持的功能

后 续

职位发布

在“社招后台”新增职位时，同步发布到其他各大招聘网站（无限跨域请求 && 消息通信）

简历解析可支持其他招聘网站

支持各大招聘网站下的插件弹出/常驻，以及对不同场景下的简历解析。（content.js的匹配植入 && 简历节点捕捉 && 错误收集）

分享链接

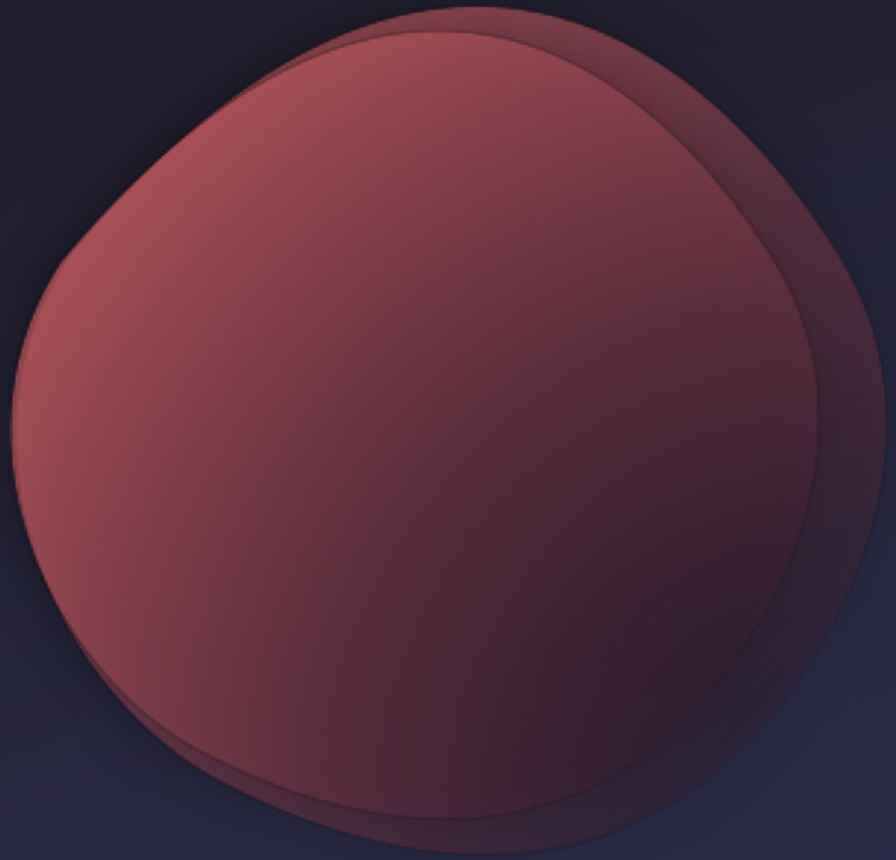
Chrome Extensions官网（英文）：https://developer.chrome.com/home

Chrome Extensions官网（中文）：https://crxdoc-zh.appspot.com/extensions/

Chrome插件(扩展)开发全攻略： http://blog.haoji.me/chrome-plugin-develop.html

“简历助手插件”安装地址：

https://chrome.google.com/webstore/detail/简历助手插件/lomjhpfinblphednogodghkiiaomjofm?authuser=1



Thanks