如何开发 Chrome Extension

> 强烈推荐阅读 《Chrome插件开发全攻略》配套完整Demo
>
> 如有必要再对该文章进行精读提炼，本文仅从快速实现的角度进行讲解。

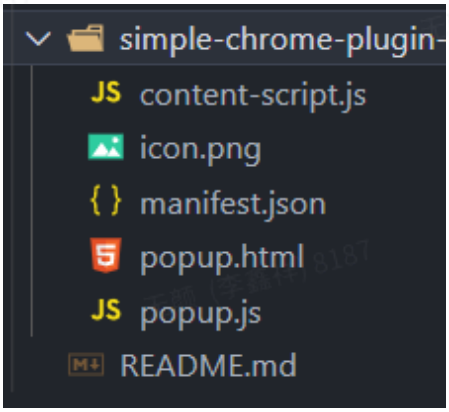
一、什么是谷歌扩展

对于 Chrome 来说，有两种提供给开发者扩展浏览器功能的方式：

- 谷歌插件 (Chrome Plugin)
- 谷歌扩展 (Chrome Extension)

谷歌插件是借助底层的浏览器功能来开发的，需要对浏览器源码有一定的了解。而谷歌扩展是一个用 Web 技术开发、用来增强浏览器功能的软件，通常由 HTML、CSS、JS、图片等资源组成一个 .crx 后缀的压缩包。

通常包含以下文件：



二、谷歌扩展能做什么

通常我们是出于开发提效的目的来开发一个谷歌，或者为了利用浏览器 API 方便我们更好地利用浏览器做某些事情。

2.1 API 功能

Chrome插件提供了很多实用 API 供我们使用，包括但不限于：

- 书签控制；
- 下载控制；
- 窗口控制；
- 标签控制；
- 网络请求控制，各类事件监听；
- 自定义原生菜单；
- 完善的通信机制；
- ...

2.2 常用扩展

下面罗列笔者常用的浏览器扩展 (需要翻墙)，以启发大家开发适合自己的谷歌扩展：

- Vimium：用于快捷开发
- Vue.js devtools：用于 Vue 项目调试
- 二维码生成器 (Quick QR)：用于生成当前浏览地址的二维码，方便手机访问

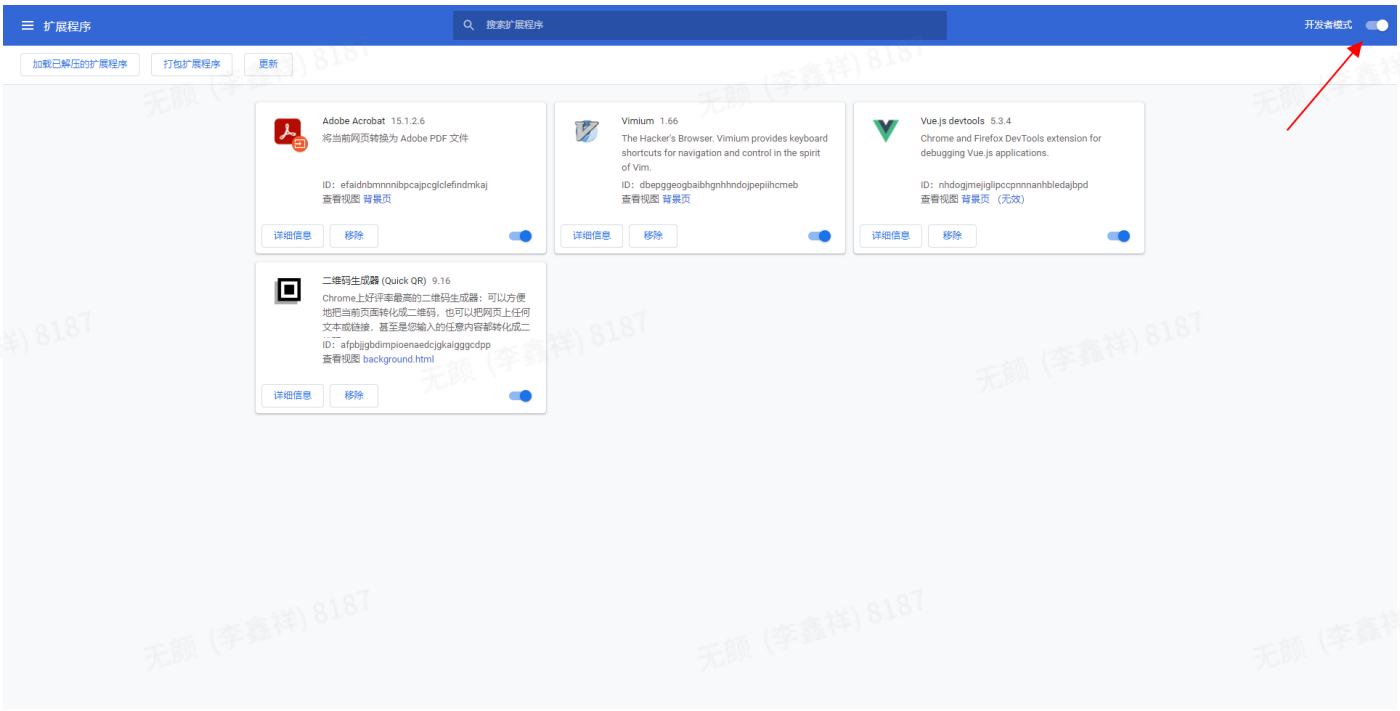
三、如何进行扩展开发

以下内容来自于对《Chrome插件开发全攻略》配套完整Demo 精读规整，需要进一步的了解的同学可以自行查看。

3.1 开发与调试

Chrome插件没有严格的项目结构要求，只要保证本目录有一个`manifest.json`即可。无异于Web 开发。

直接在谷歌浏览器地址栏输入 chrome://extensions 访问插件管理页面，并勾选开发者模式：



Chrome要求插件必须从它的Chrome应用商店安装，其它任何网站下载的都无法直接安装，所以，其实我们可以把`crx`文件解压，然后通过开发者模式直接加载。

注意：开发中，代码有任何改动都必须重新加载插件，只需要在插件管理页按下`Ctrl+R`即可，以防万一最好还把页面刷新一下。

3.2 核心介绍

以下为必知必会内容，建议读者认真阅读。

3.2.1 manifest.json

Chrome 扩展必不可少的文件，用于配置扩展，必须放在根目录。完整的配置文档请戳 这里

JSON

```json
{
  "manifest_version": 2, // 清单文件的版本，必须，且值必须是 2
  "name": "demo", // 扩展的名称
  "version": "1.0.0", // 扩展的版本
  "description": "简单的 Chrome 扩展 demo", // 扩展描述
  "icons": {
    // 图标，一般偷懒全部用一个尺寸的也没问题
    "16": "img/icon.png",
```

```
    "48": "img/icon.png",
    "128": "img/icon.png"
  },
  "background": {
    // 会一直常驻的后台 JS 或后台页面
    // 2 种指定方式，如果指定 JS，那么会自动生成一个背景页
    "page": "background.html"
    //"scripts": ["js/background.js"]
  },
  // 浏览器右上角图标设置，browser_action、page_action、app必须三选一
  "browser_action": {
    "default_icon": "img/icon.png",
    "default_title": "这是一个示例Chrome扩展", // 图标悬停时的标题，可选
    "default_popup": "popup.html"
  },
  // 当某些特定页面打开才显示的图标
  /*"page_action":
    {
        "default_icon": "img/icon.png",
        "default_title": "我是pageAction",
        "default_popup": "popup.html"
    },*/
  // 需要直接注入页面的JS
  "content_scripts": [
    {
      //"matches": ["http://*/*", "https://*/*"],
      // "<all_urls>" 表示匹配所有地址
      "matches": ["<all_urls>"],
      "js": ["js/jquery-1.8.3.js", "js/content-script.js"], // 多个 JS 按顺序注入
      "css": ["css/custom.css"], // 注意稍不注意可能影响全局样式
      "run_at": "document_start" // 代码注入的时间，可选值： "document_start", "document_end", or "document_idle"，最后一个表示页面空闲时，默认document_idle
    },
    // 这里仅仅是为了演示 content-script 可以配置多个规则
    {
      "matches": ["*://*/*.png", "*://*/*.jpg", "*://*/*.gif", "*://*/*.bmp"],
      "js": ["js/show-image-content-size.js"]
    }
  ],
  // 权限申请
  "permissions": [
    "contextMenus", // 右键菜单
    "tabs", // 标签
    "notifications", // 通知
    "webRequest", // web 请求
    "webRequestBlocking",
    "storage", // 扩展本地存储
    "http://*/*", // 可以通过 executeScript 或者 insertCSS 访问的网站
```

```
    "https://*/*" // 可以通过 executeScript 或者 insertCSS 访问的网站
  ],
  "web_accessible_resources": ["js/inject.js"], // 普通页面能够直接访问的扩展资源列表，如果不设置是无法直接访问的
  "homepage_url": "https://www.baidu.com", // 扩展主页，这个很重要，不要浪费了这个免费广告位
  "chrome_url_overrides": {
    // 覆盖浏览器默认页面
    "newtab": "newtab.html" // 覆盖浏览器默认的新标签页
  },
  "omnibox": { "keyword": "go" }, // 向地址栏注册一个关键字以提供搜索建议，只能设置一个关键字
  "default_locale": "zh_CN", // 默认语言
  "devtools_page": "devtools.html" // devtools 页面入口，注意只能指向一个 HTML 文件，不能是 JS 文件
}
```

3.2.2 content-scripts

content-scripts 是 Chrome 插件中向页面注入脚本的一种形式（虽然名为script，其实还可以包括 css），借助 `content-scripts` 我们可以实现通过配置的方式轻松向指定页面注入 JS 和 CSS（如果需要动态注入，可以参考下文），最常见的比如：广告屏蔽、页面 CSS 定制，等等。

示例配置：

JSON

```
{
  // 需要直接注入页面的 JS
  "content_scripts": [
    {
      //"matches": ["http://*/*", "https://*/*"],
      // "<all_urls>" 表示匹配所有地址
      "matches": ["<all_urls>"],
      // 多个JS按顺序注入
      "js": ["js/jquery-1.8.3.js", "js/content-script.js"],
      // JS的注入可以随便一点，但是CSS的注意就要千万小心了，因为一不小心就可能影响全局样式
      "css": ["css/custom.css"],
      // 代码注入的时间，可选值： "document_start", "document_end", or "document_idle"，最后一个表示页面空闲时，默认document_idle
      "run_at": "document_start"
    }
  ]
}
```

特别注意，如果没有主动指定 `run_at` 为 `document_start` （默认为 `document_idle` ），下面这种代码是不会生效的：

JavaScript

```
document.addEventListener("DOMContentLoaded", function () {
  console.log("我被执行了！");
});
```

`content-scripts` 和原始页面共享 DOM，但是不共享 JS，如要访问页面 JS（例如某个 JS 变量），只能通过 `injected js` 来实现。`content-scripts` 不能访问绝大部分 `chrome.xxx.api` ，除了下面这4种：

- chrome.extension(getURL , inIncognitoContext , lastError , onRequest , sendRequest)
- chrome.i18n
- chrome.runtime(connect , getManifest , getURL , id , onConnect , onMessage , sendMessage)
- chrome.storage

Chrome 插件给我们提供了这么强大的 JS 注入功能，剩下的就是发挥你的想象力去扩展浏览器了。

3.2.3 background

background 是一个常驻的页面，它的生命周期是插件中所有类型页面中最长的，它随着浏览器的打开而打开，随着浏览器的关闭而关闭，所以通常把需要一直运行的、启动就运行的、全局的代码放在 background 里面。

background 的权限非常高，几乎可以调用所有的 Chrome 扩展 API（除了 devtools），而且它可以无限制跨域，也就是可以跨域访问任何网站而无需要求对方设置 `CORS` 。

配置中，`background` 可以通过 `page` 指定一张网页，也可以通过 `scripts` 直接指定一个 JS，Chrome 会自动为这个 JS 生成一个默认的网页：

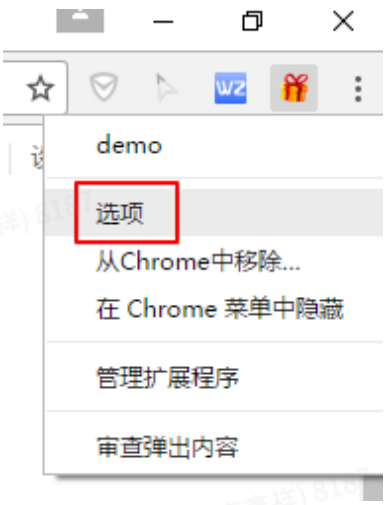
JSON

```
{
  // 会一直常驻的后台JS或后台页面
  "background": {
    // 2种指定方式，如果指定JS，那么会自动生成一个背景页
    "page": "background.html"
    //"scripts": ["js/background.js"]
  }
}
```

3.2.4 popup

3.2.5 option

所谓`options`页，就是插件的设置页面，有2个入口，一个是右键图标有一个“选项”菜单，还有一个在插件管理页面：



3.2.6 桌面通知

Chrome 提供了一个`chrome.notifications` API 以便插件推送桌面通知，暂未找到`chrome.notifications`和 HTML5 自带的`Notification`的显著区别及优势。

在后台JS中，无论是使用 `chrome.notifications` 还是 `Notification` 都不需要申请权限（HTML5 方式需要申请权限），直接使用即可。

最简单的通知：

代码：

JavaScript

```
chrome.notifications.create(null, {
  type: "basic",
  iconUrl: "img/icon.png",
  title: "这是标题",
  message: "您刚才点击了自定义右键菜单！",
});
```

3.2.7 5 种类型的 JS 对比

	A	B	C
1	**JS种类**	**可访问的API**	**DOM访问情况**
2	injected script	和普通JS无任何差别，不能访问任何扩展API	可以访问
3	content script	只能访问 extension、runtime等部分API	可以访问
4	popup js	可访问绝大部分API，除了devtools系列	不可直接访问
5	background js	可访问绝大部分API，除了devtools系列	不可直接访问
6	devtools js	只能访问 devtools、extension、runtime等部分API	可以

3.2.8 常用 API

- chrome.tabs
- chrome.runtime
- chrome.webRequest

- chrome.window
- chrome.storage
- chrome.contextMenus
- chrome.devtools
- chrome.extension

四、实战

下面以最简单的方式和读者一起完成一个谷歌扩展 MVP 版本。

4.1 项目启动

首次进行谷歌扩展开发时，不一定要从零开发。可以选用自己熟悉得技术栈 (比如 vue) 来进行工程级别的开发，这里可以通过改造 vue3-chrome-ext-template 来进行。

克隆项目：

```Bash
git clone https://github.com/leer0911/vue3-chrome-ext-template.git
```

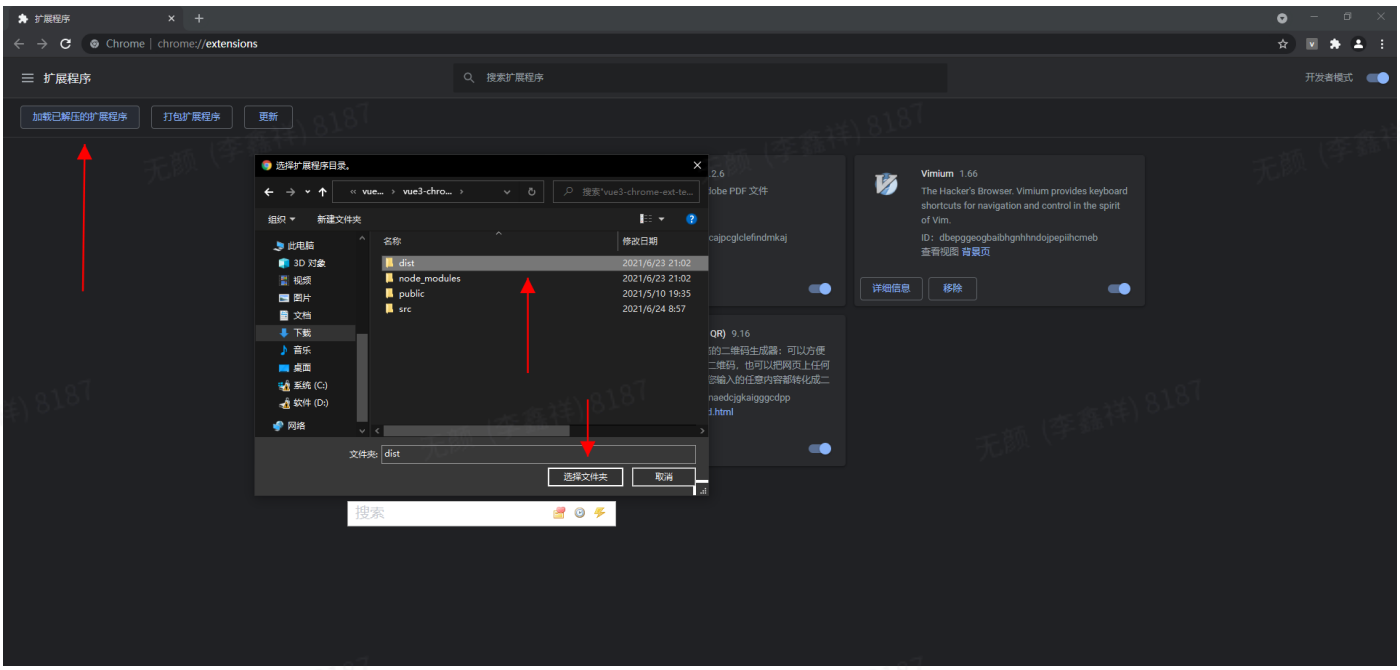
目录说明：

```Bash
├──pages
   ├──background  // 常驻后台的插件脚本
   ├──content     // 注入页面的脚本 （可以获取页面dom...）
   ├──options     // 插件的配置页
   └──popup       // 点击右上角插件图标展示的页面
├──manifest.json // 谷歌插件配置文件
```

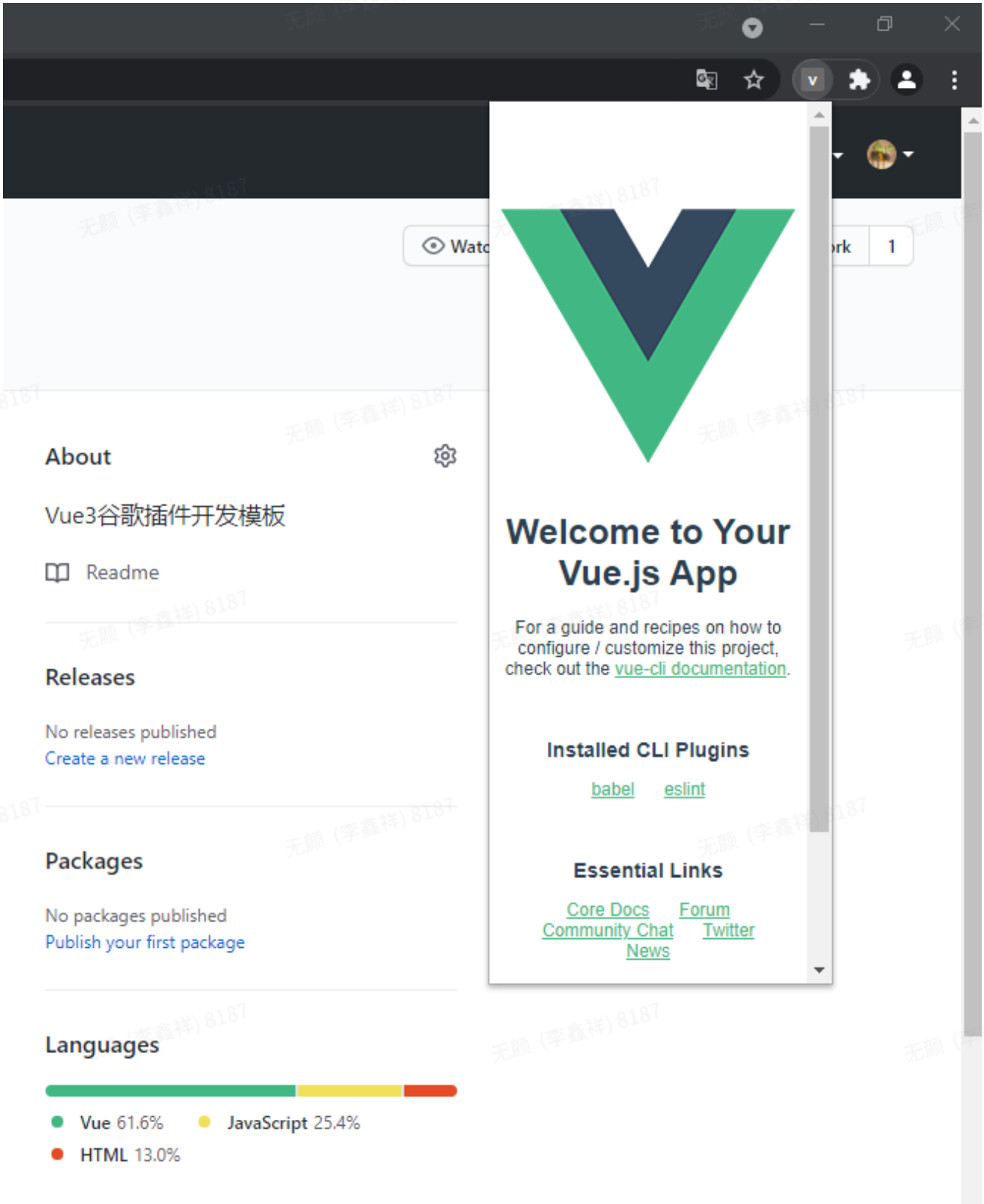
启动项目：

```Bash
yarn serve
```

项目启动后，会启用打包命令并生成 dist 目录。通过谷歌浏览器访问 chrome://extensions/，开启开发者模式。将 dist 的文件加载进来以此来安装谷歌扩展：



在谷歌右上角启用扩展：



4.2 SSO 登录场景

在公司内部开发环境下，SSO 登录无法将 token 设置到 localhost 的域名下。故采用谷歌扩展的方式来解决 (临时方案) ：

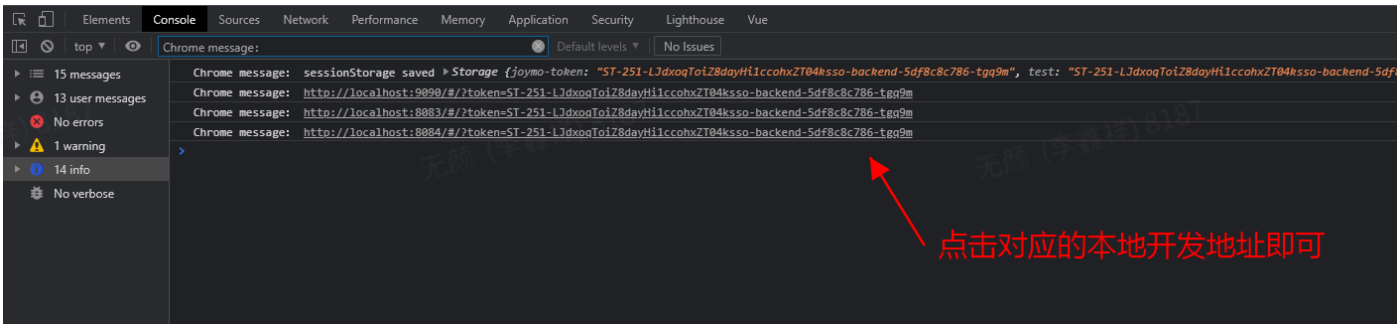
在 `src/pages/content/main.js` 文件下添加如下代码：

JavaScript

```
chrome.storage.sync.set(sessionStorage, function() {
  // 将 sessionStorage 存储到 chrome storage 下
  console.log('Chrome message: ', 'sessionStorage saved', sessionStorage)
})

chrome.storage.sync.get(['joymo-token'], function(data) {
  // 对于开发者本地有的所有开发端口号
  const ports = ['9090', '8083', '8084', '8085']
  ports.forEach(port => {
    console.log(
      'Chrome message: ',
      `http://localhost:${port}/#/?token=${data['joymo-token']}`
    )
  })
})
```

通过谷歌浏览器的控制台查看打印的对于地址：



以上仅从最简单的角度开发一款可以解决业务场景的谷歌扩展，更多可能还有待各位开发者自行发觉。

五、资料

5.1 官方资料

推荐查看官方文档，虽然是英文，但是全且新，国内的中文资料都比较旧（注意以下全部需要翻墙）：

- Chrome插件官方文档主页：https://developer.chrome.com/extensions
- Chrome插件官方示例：https://developer.chrome.com/extensions/samples
- manifest清单文件：https://developer.chrome.com/extensions/manifest
- permissions权限：https://developer.chrome.com/extensions/permissions

- chrome.xxx.api文档：https://developer.chrome.com/extensions/api_index
- 模糊匹配规则语法详解：https://developer.chrome.com/extensions/match_patterns

5.2 第三方资料