

小茗同学的博客园

个人网站: <http://haoji.me>(好记么)

【干货】Chrome插件(扩展)开发全攻略

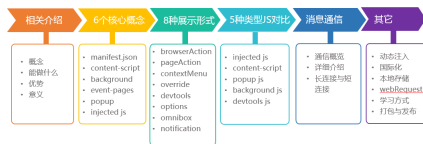
1. 写在前面

我花了将近一个多月的时间断断续续写下这篇博文,并精心写下完整demo,写博客的辛苦大家懂的,所以转载务必保留[出处](#)。本文所有涉及到的大部分代码均在这个demo里面: <https://github.com/sxei/chrome-plugin-demo>,大家可以直接下载下来运行。

另外,本文图片较多,且图片服务器带宽有限,右下角的目录滚动监听必须等到图片全部加载完毕之后才会触发,所以请耐心等待加载完毕。

本文目录:

详细目录



demo部分截图:



2. 前言

2.1. 什么是Chrome插件

严格来讲,我们正在说的东西应该叫Chrome扩展(Chrome Extension),真正意义上的Chrome插件是更底层的浏览器功能扩展,可能需要对浏览器源码有一定掌握才有能力去开发。鉴于Chrome插件的叫法已经习惯,本文也全部采用这种叫法,但读者需深知本文所描述的Chrome插件实际上指的是Chrome扩展。

Chrome插件是一个用Web技术开发、用来增强浏览器功能的软件,它其实就是一个由HTML、CSS、JS、图片等资源组成的一个.crx后缀的压缩包。

公告



欢迎访问我的个人博客^_^:

<http://haoji.me>(好记么)

您是本博客第_____位访客
所有文章首发于我的[个人博客](#)和[博客园](#)。我的[github](#)

昵称: 我是小茗同学

园龄: 7年9个月

粉丝: 702

关注: 18

+加关注

最新随笔

- 1.VSCode插件开发全攻略(九)常用API总结
- 2.VSCode插件开发全攻略(十)打包、发布、升级
- 3.VSCode插件开发全攻略(八)代码片段、设置、自定义欢迎页
- 4.VSCode插件开发全攻略(七)WebView
- 5.VSCode插件开发全攻略(六)开发调试技巧
- 6.VSCode插件开发全攻略(五)跳转到定义、自动补全、悬停提示
- 7.VSCode插件开发全攻略(四)命令、菜单、快捷键
- 8.VSCode插件开发全攻略(三)package.json详解
- 9.VSCode插件开发全攻略(二)Hello Word
- 10.VSCode插件开发全攻略(一)概览



查看
评论

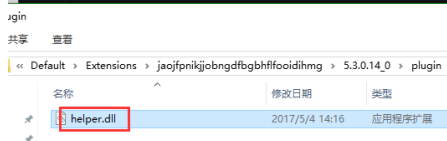
个人猜测 `crx` 可能是 `Chrome Extension` 如下3个字母的简写:



Chrome Extension

另外,其实不只是前端技术,Chrome插件还可以配合C++编写的dll动态链接库实现一些更底层的功能(NPAPI),比如全屏幕截图。

```
36 "name": "MSG_name",
37 "permissions": [ "notifications", "tabs", "
38 "plugins": {
39 "path": "plugin/helper.dll"
40 } ],
41 "update_url": "http://upest.chrome.360.cn/3
```



由于安全原因,Chrome浏览器42以上版本已经陆续不再支持NPAPI插件,取而代之的是更安全的PPAPI。

2.2. 学习Chrome插件开发有什么意义

增强浏览器功能,轻松实现属于自己的“定制版”浏览器,等等。

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;
- 等等;

2.3. 为什么是Chrome插件而不是Firefox插件

1. Chrome占有率更高,更多人用;
2. 开发更简单;
3. 应用场景更广泛,Firefox插件只能运行在Firefox上,而Chrome除了Chrome浏览器之外,还可以运行在所有webkit内核的国产浏览器,比如360极速浏览器、360安全浏览器、搜狗浏览器、QQ浏览器等等;
4. 除此之外,Firefox浏览器也对Chrome插件的运行提供了一定的支持;

3. 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,也不需要专门的IDE,普通的web开发工具即可。

从右上角菜单->更多工具->扩展程序可以进入 插件管理页面,也可以直接在地址栏输入 `chrome://extensions` 访问。

积分与排名

积分 - 181836

排名 - 2289

阅读排行榜

1. 一个非常标准的Java连接Oracle数据库的示例代码(199335)
2. 【干货】Chrome插件(扩展)开发全攻略(148331)
3. 使用hexo+github搭建免费个人博客详细教程(98228)
4. 彻底禁用Chrome的“请停用以开发者模式运行的扩展程序”提示(71051)
5. Linux下安装SVN服务端小白教程(66105)
6. 【入门教程】使用Eclipse搭建C/C++开发环境(62904)
7. 如何使用JavaScript实现纯前端读取和导出excel文件(52768)
8. 详细记录一下网站备案经过,备案真的很简单(36848)
9. Eclipse安装插件的“最好方法”: dr opins文件夹的妙用(34816)
10. 【超详细教程】使用Windows Live Writer 2012和Office Word 2013 发布文章到博客园全面总结(33934)

评论排行榜

1. 【干货】Chrome插件(扩展)开发全攻略(120)
2. 【超详细教程】使用Windows Live Writer 2012和Office Word 2013 发布文章到博客园全面总结(110)
3. 【干货】JS版汉字与拼音互转终极方案,附简单的JS拼音输入法(75)
4. HTTPS从认识到线上实战全记录(54)
5. 使用hexo+github搭建免费个人博客详细教程(49)
6. 坑爹坑娘坑祖宗的87端口(记一次tomcat故障排查)(45)
7. 彻底禁用Chrome的“请停用以开发者模式运行的扩展程序”提示(42)
8. 有强迫症的我只能自己写一个json格式化工具(35)
9. 巧用transform实现HTML5 video标签视频比例拉伸(25)
10. 分享自己写的JS版日期格式化和解析工具类,绝对好用!(24)



勾选 **开发者模式** 即可以文件夹的形式直接加载插件，否则只能安装 **.crx** 格式的文件。**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。

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。

4. 核心介绍

4.1. manifest.json

这是一个**Chrome**插件最重要也是必不可少的文件，用来配置所有和插件相关的配置，必须放在根目录。其中，**manifest_version**、**name**、**version** 3个是必不可少的，**description** 和 **icons** 是推荐的。

下面给出的是一些常见的配置项，均有中文注释，完整的配置文档请戳 [这里](#)。

推荐排行榜

1. 【干货】Chrome插件(扩展)开发全攻略(228)
2. 使用hexo+github搭建免费个人博客详细教程(219)
3. 【超详细教程】使用Windows Live Writer 2012和Office Word 2013 发布文章到博客园全面总结(202)
4. 【干货】JS版汉字与拼音互转终极方案，附简单的JS拼音输入法(125)
5. HTTPS从认识到线上实战全记录(85)
6. 坑爹坑娘坑祖宗的87端口（记一次tomcat故障排查）(65)
7. 分享自己写的JS版日期格式化和解析工具类，绝对好用！（24）
8. 如何自定义博客园代码高亮主题，同时分享自己使用的黑色主题(22)
9. 彻底禁用Chrome的“请停用以开发者模式运行的扩展程序”提示(17)
10. 巧用transform实现HTML5 video 标签视频比例拉伸(15)

```

{
  // 清单文件的版本，这个必须写，而且必须是2
  "manifest_version": 2,
  // 插件的名称
  "name": "demo",
  // 插件的版本
  "version": "1.0.0",
  // 插件描述
  "description": "简单的Chrome扩展demo",
  // 图标，一般偷懒全部用一个尺寸的也没问题
  "icons":
  {
    "16": "img/icon.png",
    "48": "img/icon.png",
    "128": "img/icon.png"
  },
  // 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"]
  },
  // 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",
    // 图标悬停时的标题，可选
    "default_title": "这是一个示例Chrome插件",
  }
}

```

4.2. content-scripts

所谓 `content-scripts`，其实就是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的），借助 `content-scripts` 我们可以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：

```

{
  // 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
      "css": ["css/custom.css"],
      // 代码注入的时间，可选值： "document_start", "document_end", or "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：

```
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`

其实看到这里不要悲观，这些API绝大部分时候都够用了，非要调用其它API的话，你还可以通过通信来实现让background来帮你调用（关于通信，后文有详细介绍）。

好了，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玩弄浏览器了。

4.3. background

后台（姑且这么翻译吧），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`。

经过测试，其实不止是background，所有的直接通过 `chrome-extension://id/xx.html` 这种方式打开的网页都可以无限制跨域。

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：

```
{
  // 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"]
  },
}
```

需要特别说明的是，虽然你可以通过 `chrome-extension://xxx/background.html` 直接打开后台页，但是你打开的后台页和真正一直在后台运行的那个页面不是同一个，换句话说，你可以打开无数个 `background.html`，但是真正在后台常驻的只有一个，而且这个你永远看不到它的界面，只能调试它的代码。

4.4. event-pages

这里顺带介绍一下 `event-pages`，它是一个什么东西呢？鉴于background生

命周期太长，长时间挂载后台可能会影响性能，所以Google又弄一个 `event-pages`，在配置文件上，它与`background`的唯一区别就是多了一个 `persistent` 参数：

```
{
  "background":
  {
    "scripts":["event-page.js"],
    "persistent": false
  },
}
```

它的生命周期是：在被需要时加载，在空闲时被关闭，什么叫被需要时呢？比如第一次安装、插件更新、有`content-script`向它发送消息，等等。

除了配置文件的变化，代码上也有一些细微变化，个人这个简单了解一下就行了，一般情况下`background`也不会很消耗性能的。

4.5. popup

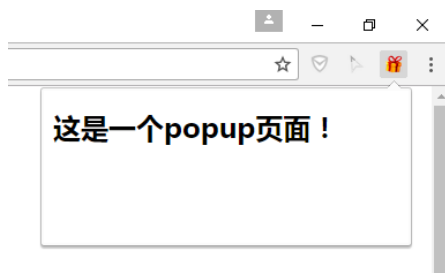
`popup` 是点击 `browser_action` 或者 `page_action` 图标时打开的一个小窗口网页，焦点离开网页就立即关闭，一般用来做一些临时性的交互。



`popup` 可以包含任意你想要的HTML内容，并且会自适应大小。可以通过 `default_popup` 字段来指定`popup`页面，也可以调用 `setPopup()` 方法。

配置方式：

```
{
  "browser_action":
  {
    "default_icon": "img/icon.png",
    // 图标悬停时的标题，可选
    "default_title": "这是一个示例Chrome插件",
    "default_popup": "popup.html"
  }
}
```



需要特别注意的是，由于单击图标打开`popup`，焦点离开又立即关闭，所以`popup`页面的生命周期一般很短，需要长时间运行的代码千万不要写在`popup`里

面。

在权限上，它和background非常类似，它们之间最大的不同是生命周期的不同，popup中可以直接通过 `chrome.extension.getBackgroundPage()` 获取background的window对象。

4.6. injected-script

这里的 `injected-script` 是我给它取的，指的是通过DOM操作的方式向页面注入的一种JS。为什么要把这种JS单独拿出来讨论呢？又或者说为什么需要通过这种方式注入JS呢？

这是因为 `content-script` 有一个很大的“缺陷”，也就是无法访问页面中的JS，虽然它可以操作DOM，但是DOM却不能调用它，也就是无法在DOM中通过绑定事件的方式调用 `content-script` 中的代码（包括直接写 `onclick` 和 `addEventListener` 2种方式都不行），但是，“在页面上添加一个按钮并调用插件的扩展API”是一个很常见的需求，那该怎么办呢？其实这就是本小节要讲的。

在 `content-script` 中通过DOM方式向页面注入 `injected-script` 代码示例：

```
// 向页面注入JS
function injectCustomJs(jsPath)
{
  jsPath = jsPath || 'js/inject.js';
  var temp = document.createElement('script');
  temp.setAttribute('type', 'text/javascript');
  // 获得的地址类似: chrome-extension://ihcokhadfjfhcaeagdoclbnjdiokfakg/js/inject.js
  temp.src = chrome.extension.getURL(jsPath);
  temp.onload = function()
  {
    // 放在页面不好看，执行完后移除掉
    this.parentNode.removeChild(this);
  };
  document.head.appendChild(temp);
}
```

你以为这样就行了？执行一下你会看到如下报错：

```
Denying load of chrome-extension://efblncjkijijkpapehoekjojdcl/js/inject.js. Resource
```

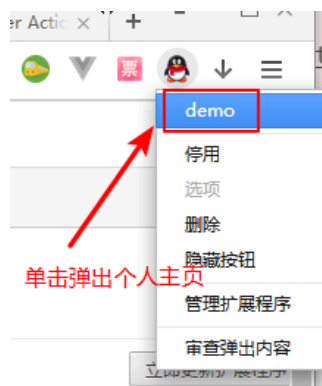
意思就是你想要在web中直接访问插件中的资源的话必须显示声明才行，配置文件中增加如下：

```
{
  // 普通页面能够直接访问的插件资源列表，如果不设置是无法直接访问的
  "web_accessible_resources": ["js/inject.js"],
}
```

至于 `inject-script` 如何调用 `content-script` 中的代码，后面我会在专门的一个消息通信章节详细介绍。

4.7. homepage_url

开发者或者插件主页设置，一般会在如下2个地方显示：



5. Chrome插件的8种展示形式

5.1. browserAction(浏览器右上角)

通过配置 `browser_action` 可以在浏览器的右上角增加一个图标，一个 `browser_action` 可以拥有一个图标，一个 `tooltip`，一个 `badge` 和一个 `popup`。

示例配置如下：

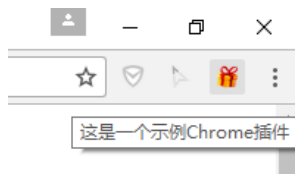
```
"browser_action":
{
  "default_icon": "img/icon.png",
  "default_title": "这是一个示例Chrome插件",
  "default_popup": "popup.html"
}
```

5.1.1. 图标

`browser_action` 图标推荐使用宽高都为19像素的图片，更大的图标会被缩小，格式随意，一般推荐png，可以通过manifest中 `default_icon` 字段配置，也可以调用 `setIcon()` 方法。

5.1.2. tooltip

修改 `browser_action` 的manifest中 `default_title` 字段，或者调用 `setTitle()` 方法。

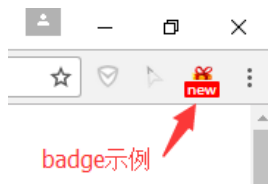


5.1.3. badge

所谓 `badge` 就是在图标上显示一些文本，可以用来更新一些小的扩展状态提示信息。因为badge空间有限，所以只支持4个以下的字符（英文4个，中文2个）。badge无法通过配置文件来指定，必须通过代码实现，设置badge文字和颜色可以分别使用 `setBadgeText()` 和 `setBadgeBackgroundColor()`。

```
chrome.browserAction.setBadgeText({text: 'new'});
chrome.browserAction.setBadgeBackgroundColor({color: [255, 0, 0, 255]});
```

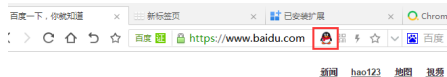

效果:



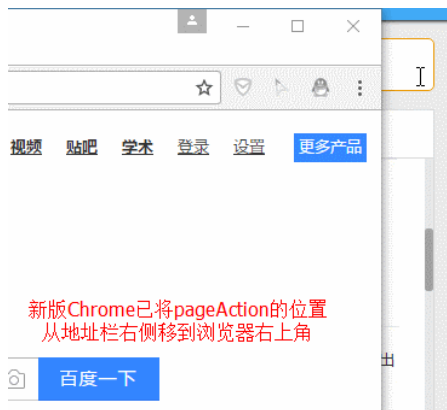
5.2. pageAction(地址栏右侧)

所谓 **pageAction**，指的是只有当某些特定页面打开才显示的图标，它和 **browserAction** 最大的区别是一个始终都显示，一个只在特定情况才显示。

需要特别说明的是早些版本的Chrome是将pageAction放在地址栏的最右边，左键单击弹出popup，右键单击则弹出相关默认选项菜单：



而新版的Chrome更改了这一策略，pageAction和普通的browserAction一样也是放在浏览器右上角，只不过没有点亮时是灰色的，点亮了才是彩色的，灰色时无论左键还是右键单击都是弹出选项：



具体是从哪一版本开始改的没去仔细考究，反正知道v50.0的时候还是前者，v58.0的时候已改为后者。

调整之后的 **pageAction** 我们可以简单地把它看成是可以置灰的 **browserAction**。

- `chrome.pageAction.show(tabId)` 显示图标；
- `chrome.pageAction.hide(tabId)` 隐藏图标；

示例(只有打开百度才显示图标):

```
// manifest.json
{
  "page_action": {
    {
      "default_icon": "img/icon.png",
      "default_title": "我是pageAction",
      "default_popup": "popup.html"
    },
    "permissions": ["declarativeContent"]
  }
}

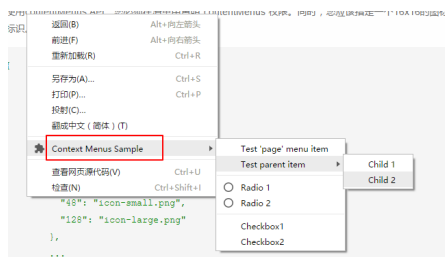
// background.js
chrome.runtime.onInstalled.addListener(function(){
  chrome.declarativeContent.onPageChanged.removeRules(undefined, function(){
    chrome.declarativeContent.onPageChanged.addRules([
      {
        conditions: [
          // 只有打开百度才显示pageAction
          new chrome.declarativeContent.PageStateMatcher({pageUrl: {urlContains:
        ],
        actions: [new chrome.declarativeContent.ShowPageAction()]
      }
    ]);
  });
});
});
```

效果图：



5.3. 右键菜单

通过开发Chrome插件可以自定义浏览器的右键菜单，主要是通过 `chrome.contextMenus` API实现，右键菜单可以出现在不同的上下文，比如普通页面、选中的文字、图片、链接，等等，如果有同一个插件里面定义了多个菜单，Chrome会自动组合放到以插件名字命名的二级菜单里，如下：



5.3.1. 最简单的右键菜单示例

```
// manifest.json
{"permissions": ["contextMenus"]}

// background.js
chrome.contextMenus.create({
  title: "测试右键菜单",
  onclick: function(){alert("您点击了右键菜单！");}
});
```

效果：

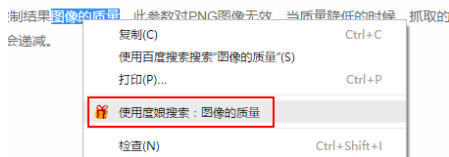


5.3.2. 添加右键百度搜索

```
// manifest.json
{"permissions": ["contextMenus", "tabs"]}

// background.js
chrome.contextMenus.create({
  title: '使用度娘搜索: %s', // %s表示选中的文字
  contexts: ['selection'], // 只有当选中文字时才会出现此右键菜单
  onclick: function(params)
  {
    // 注意不能使用location.href, 因为location是属于background的window对象
    chrome.tabs.create({url: 'https://www.baidu.com/s?ie=utf-8&wd=' + encodeURIComponent(params.selectionText)});
  }
});
```

效果如下:



5.3.3. 语法说明

这里只是简单列举一些常用的, 完整API参

见: <https://developer.chrome.com/extensions/contextMenus>

```
chrome.contextMenus.create({
  type: 'normal', // 类型, 可选: ['normal', 'checkbox', 'radio', 'separator'], 默认
  title: '菜单的名字', // 显示的文字, 除非为"separator"类型否则此参数必需, 如
  contexts: ['page'], // 上下文环境, 可选: ['all', 'page', 'frame', 'selection', 'link', 'image'],
  onclick: function() {}, // 单击时触发的方法
  parentId: 1, // 右键菜单项的父菜单项ID. 指定父菜单项将会使此菜单项成为
  documentUrlPatterns: ['https://*.baidu.com/*'] // 只在某些页面显示此右键菜单
});

// 删除某一个菜单项
chrome.contextMenus.remove(menuItemId);

// 删除所有自定义右键菜单
chrome.contextMenus.removeAll();

// 更新某一个菜单项
chrome.contextMenus.update(menuItemId, updateProperties);
```

5.4. override(覆盖特定页面)

使用 **override** 页可以将Chrome默认的一些特定页面替换掉，改为使用扩展提供的页面。

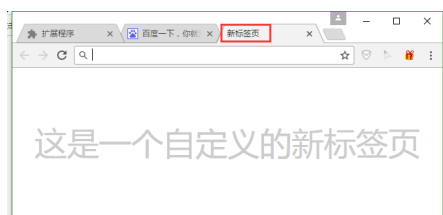
扩展可以替代如下页面：

- 历史记录：从工具菜单上点击历史记录时访问的页面，或者从地址栏直接输入 `chrome://history`
- 新标签页：当创建新标签的时候访问的页面，或者从地址栏直接输入 `chrome://newtab`
- 书签：浏览器的书签，或者直接输入 `chrome://bookmarks`

注意：

- 一个扩展只能替代一个页面；
- 不能替代隐身窗口的新标签页；
- 网页必须设置title，否则用户可能会看到网页的URL，造成困扰；

下面的截图是默认的新标签页和被扩展替换掉的新标签页。



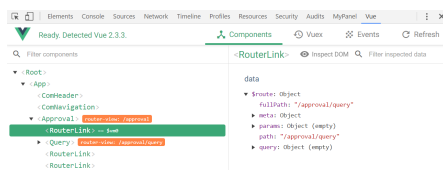
代码（注意，一个插件只能替代一个默认页，以下仅为演示）：

```
"chrome_url_overrides":
{
  "newtab": "newtab.html",
  "history": "history.html",
  "bookmarks": "bookmarks.html"
}
```

5.5. devtools(开发者工具)

5.5.1. 预热

使用过vue的应该见过这种类型的插件：



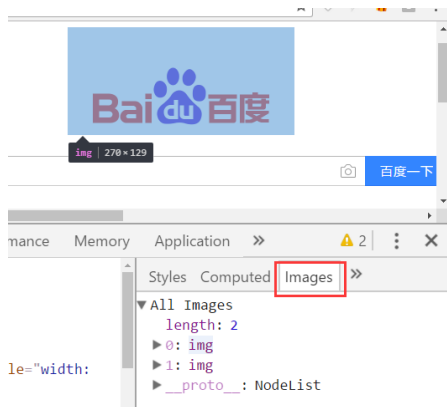
是的，Chrome允许插件在开发者工具(devtools)上动手脚，主要表现在：

- 自定义一个和多个和 **Elements**、**Console**、**Sources** 等等级别的面板；
- 自定义侧边栏(sidebar)，目前只能自定义 **Elements** 面板的侧边栏；

先来看2张简单的demo截图，自定义面板（判断当前页面是否使用了jQuery）：



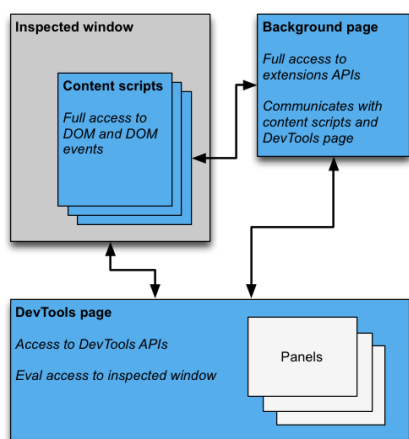
自定义侧边栏（获取当前页面所有图片）：



5.5.2. devtools扩展介绍

主页: <https://developer.chrome.com/extensions/devtools>

来一张官方图片:



每打开一个开发者工具窗口, 都会创建devtools页面的实例, F12窗口关闭, 页面也随着关闭, 所以devtools页面的生命周期和devtools窗口是一致的。devtools页面可以访问一组特有的 **DevTools API** 以及有限的扩展API, 这组特有的 **DevTools API** 只有devtools页面才可以访问, background都无权访问, 这些API包括:

- **chrome.devtools.panels**: 面板相关;
- **chrome.devtools.inspectedWindow**: 获取被审查窗口的有关信息;
- **chrome.devtools.network**: 获取有关网络请求的信息;

大部分扩展API都无法直接被 **DevTools** 页面调用, 但它可以像 **content-script** 一样直接调用 **chrome.extension** 和 **chrome.runtime** API, 同时它也可以像 **content-script** 一样使用Message交互的方式与background页面进行通信。

5.5.3. 实例: 创建一个devtools扩展

首先, 要针对开发者工具开发插件, 需要在清单文件声明如下:

```
{
  // 只能指向一个HTML文件, 不能是JS文件
  "devtools_page": "devtools.html"
}
```

这个 **devtools.html** 里面一般什么都没有, 就引入一个js:

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <script type="text/javascript" src="js/devtools.js"></script>
</body>
</html>
```

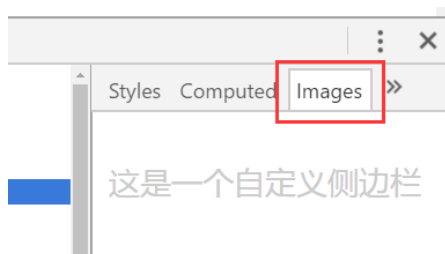
可以看出来，其实真正代码是 `devtools.js`，`html`文件是“多余”的，所以这里觉得有点坑，`devtools_page` 干嘛不允许直接指定JS呢？

再来看`devtools.js`的代码：

```
// 创建自定义面板，同一个插件可以创建多个自定义面板
// 几个参数依次为：panel标题、图标（其实设置了也没地方显示）、要加载的
chrome.devtools.panels.create('MyPanel', 'img/icon.png', 'mypanel.html', function(panel)
{
  console.log('自定义面板创建成功！'); // 注意这个log一般看不到
});

// 创建自定义侧边栏
chrome.devtools.panels.elements.createSidebarPane('Images', function(sidebar)
{
  // sidebar.setPage('../sidebar.html'); // 指定加载某个页面
  sidebar.setExpression('document.querySelectorAll("img")', 'All Images'); // 通过表达式
  // sidebar.setObject({aaa: 111, bbb: 'Hello World!'}); // 直接设置显示某个对象
});
```

`setPage`时的效果：



以下截图示例的代码：

```

检测当前页面jQuery
查看当前页面HTML代码的第20行
审查页面第一张图片
获取当前页面所有Resources
```

```
// 检测jQuery
document.getElementById('check_jquery').addEventListener('click', function()
{
    // 访问被检查的页面DOM需要使用inspectedWindow
    // 简单例子：检测被检查页面是否使用了jQuery
    chrome.devtools.inspectedWindow.eval("jQuery.fn.jquery", function(result, isException)
    {
        var html = "";
        if (isException) html = '当前页面没有使用jQuery。';
        else html = '当前页面使用了jQuery，版本为：' + result;
        alert(html);
    });
});

// 打开某个资源
document.getElementById('open_resource').addEventListener('click', function()
{
    chrome.devtools.inspectedWindow.eval("window.location.href", function(result, isException)
    {
        chrome.devtools.panels.openResource(result, 20, function()
        {
            console.log('资源打开成功！');
        });
    });
});

// 审查元素
document.getElementById('test_inspect').addEventListener('click', function()
{

```

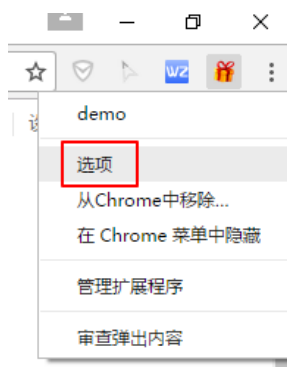
5.5.4. 调试技巧

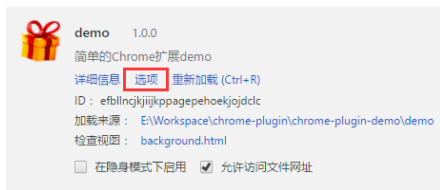
修改了devtools页面的代码时，需要先在 [chrome://extensions](#) 页面按下 **Ctrl+R** 重新加载插件，然后关闭再打开开发者工具即可，无需刷新页面（而且只刷新页面不刷新开发者工具的话是不会生效的）。

由于devtools本身就是开发者工具页面，所以几乎没有方法可以直接调试它，直接用 `chrome-extension://extid/devtools.html` 的方式打开页面肯定报错，因为不支持相关特殊API，只能先自己写一些方法屏蔽这些错误，调试通了再放开。

5.6. option(选项页)

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：





在Chrome40以前，options页面和其它普通页面没什么区别，Chrome40以后则有了一些变化。

我们先看老版的 options：

```
{
  // Chrome40以前的插件配置页写法
  "options_page": "options.html",
}
```

这个页面里面的内容就随你自己发挥了，配置之后在插件管理页就会看到一个 **选项** 按钮入口，点进去就是打开一个网页，没啥好讲的。

效果：



简单的配置页

(功能很无聊，纯属演示功能)

请选择popup页面背景色：yellow 保存配置

再来看新版的 optionsV2：

```
{
  "options_ui": {
    "page": "options.html",
    // 添加一些默认的样式，推荐使用
    "chrome_style": true
  },
}
```

options.html 的代码我们没有任何改动，只是配置文件改了，之后效果如下：



看起来是不是高大上了？

几点注意：

- 为了兼容，建议2种都写，如果都写了，Chrome40以后会默认读取新版的方式；
- 新版options中不能使用alert；
- 数据存储建议用chrome.storage，因为会随用户自动同步；

5.7. omnibox

`omnibox` 是向用户提供搜索建议的一种方式。先来看个 `gif` 图以便了解一下这东西到底是个什么鬼：



通过某个关键字触发 `omnibox`，本例中是 "go"

注册某个关键字以触发插件自己的搜索建议界面，然后可以任意发挥了。

首先，配置文件如下：

```
{
  // 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
  "omnibox": { "keyword": "go" },
}
```

然后 `background.js` 中注册监听事件：

```
// omnibox 演示
chrome.omnibox.onInputChanged.addListener((text, suggest) => {
  console.log('inputChanged: ' + text);
  if(!text) return;
  if(text === '美女') {
    suggest([
      {content: '中国' + text, description: '你要找“中国美女”吗？'},
      {content: '日本' + text, description: '你要找“日本美女”吗？'},
      {content: '泰国' + text, description: '你要找“泰国美女或人妖”吗？'},
      {content: '韩国' + text, description: '你要找“韩国美女”吗？'}
    ]);
  }
  else if(text === '微博') {
    suggest([
      {content: '新浪' + text, description: '新浪' + text},
      {content: '腾讯' + text, description: '腾讯' + text},
      {content: '搜狐' + text, description: '搜索' + text},
    ]);
  }
  else {
    suggest([
      {content: '百度搜索' + text, description: '百度搜索' + text},
      {content: '谷歌搜索' + text, description: '谷歌搜索' + text},
    ]);
  }
});

// 当用户接收关键字建议时触发
chrome.omnibox.onInputEntered.addListener((text) => {
```

5.8. 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。

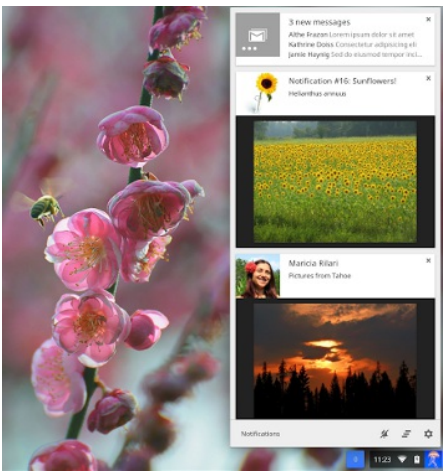
最简单的通知：



代码：

```
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！'
});
```

通知的样式可以很丰富：



这个没有深入研究，有需要的可以去看官方文档。

6. 5种类型的JS对比

Chrome插件的JS主要可以分为这5类：`injected script`、`content-script`、`popup.js`、`background.js`和`devtools.js`，

6.1. 权限对比

JS种类	可访问的API	DOM访问情况
injected script	和普通JS无任何差别，不能访问任何扩展API	可以访问
content script	只能访问 <code>extension</code> 、 <code>runtime</code> 等部分API	可以访问
popup.js	可访问绝大部分API，除了 <code>devtools</code> 系列	不可直接访问
background.js	可访问绝大部分API，除了 <code>devtools</code> 系列	不可直接访问
devtools.js	只能访问 <code>devtools</code> 、 <code>extension</code> 、 <code>runtime</code> 等部分API	可以

6.2. 调试方式对比

JS类型	调试方式	图片说明
injected script	直接普通的F12即可	懒得截图
content-script	打开Console,如图切换	
popup-js	popup页面右键审查元素	
background	插件管理页点击背景页即可	
devtools-js	暂未找到有效方法	-

7. 消息通信

通信主页: <https://developer.chrome.com/extensions/messaging>

前面我们介绍了Chrome插件中存在的5种JS, 那么它们之间如何互相通信呢? 下面先来系统概况一下, 然后再分类细说。需要知道的是, popup和background其实几乎可以视为一种东西, 因为它们可访问的API都一样、通信机制一样、都可以跨域。

7.1. 互相通信概览

注: - 表示不存在或者无意义, 或者待验证。

	injected-script	content-script
injected-script	-	window.postMessage
content-script	window.postMessage	-
popup-js	-	chrome.tabs.sendMessage chr
background-js	-	chrome.tabs.sendMessage chr
devtools-js	chrome.devtools. inspectedWindow.eval	-

7.2. 通信详细介绍

7.2.1. popup和background

popup可以直接调用background中的JS方法，也可以直接访问background的DOM:

```
// background.js
function test()
{
    alert('我是background! ');
}

// popup.js
var bg = chrome.extension.getBackgroundPage();
bg.test(); // 访问bg的函数
alert(bg.document.body.innerHTML); // 访问bg的DOM
```

小插曲，今天碰到一个情况，发现popup无法获取background的任何方法，找了半天才发现是因为background的js报错了，而你如果不主动查看background的js的话，是看不到错误信息的，特此提醒。

至于 background 访问 popup 如下（前提是 popup 已经打开）：

```
var views = chrome.extension.getViews( {type:'popup'} );
if(views.length > 0) {
    console.log(views[0].location.href);
}
```

7.2.2. popup或者bg向content主动发送消息

background.js或者popup.js:

```
function sendMessageToContentScript(message, callback)
{
    chrome.tabs.query({active: true, currentWindow: true}, function(tabs)
    {
        chrome.tabs.sendMessage(tabs[0].id, message, function(response)
        {
            if(callback) callback(response);
        });
    });
}

sendMessageToContentScript({cmd:'test', value:'你好，我是popup!'}, function(respo
{
    console.log('来自content的回复: '+response);
});
```

`content-script.js` 接收:

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse)
{
  // console.log(sender.tab ? "from a content script." + sender.tab.url : "from the extension");
  if(request.cmd === 'test') alert(request.value);
  sendResponse('我收到了你的消息! ');
});
```

双方通信直接发送的都是JSON对象，不是JSON字符串，所以无需解析，很方便（当然也可以直接发送字符串）。

网上有些老代码中用的是 `chrome.extension.onMessage`，没有完全查清二者的区别(貌似是别名)，但是建议统一使用 `chrome.runtime.onMessage`。

7.2.3. content-script主动发消息给后台

`content-script.js`:

```
chrome.runtime.sendMessage({greeting: '你好，我是content-script呀，我主动发消息'}, function(response) {
  console.log('收到来自后台的回复: ' + response);
});
```

`background.js` 或者 `popup.js`:

```
// 监听来自 content-script 的消息
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse)
{
  console.log('收到来自 content-script 的消息: ');
  console.log(request, sender, sendResponse);
  sendResponse('我是后台，我已收到你的消息: ' + JSON.stringify(request));
});
```

注意事项:

- `content_scripts`向 `popup` 主动发消息的前提是`popup`必须打开！否则需要利用`background`作中转；
- 如果`background`和`popup`同时监听，那么它们都可以同时收到消息，但是只有一个可以`sendResponse`，一个先发送了，那么另外一个再发送就无效；

7.2.4. injected script和 content-script

`content-script` 和页面内的脚本（`injected-script` 自然也属于页面内的脚本）之间唯一共享的东西就是页面的DOM元素，有2种方法可以实现二者通讯：

1. 可以通过 `window.postMessage` 和 `window.addEventListener` 来实现二者消息通讯；
2. 通过自定义DOM事件来实现；

第一种方法（推荐）：

`injected-script` 中：

```
window.postMessage({test: '你好!'}, '*');
```

`content script`中：

```
window.addEventListener("message", function(e)
{
    console.log(e.data);
}, false);
```

第二种方法:

`injected-script` 中:

```
var customEvent = document.createEvent('Event');
customEvent.initEvent('myCustomEvent', true, true);
function fireCustomEvent(data) {
    hiddenDiv = document.getElementById('myCustomEventDiv');
    hiddenDiv.innerText = data;
    hiddenDiv.dispatchEvent(customEvent);
}
fireCustomEvent('你好, 我是普通JS!');
```

`content-script.js` 中:

```
var hiddenDiv = document.getElementById('myCustomEventDiv');
if(!hiddenDiv) {
    hiddenDiv = document.createElement('div');
    hiddenDiv.style.display = 'none';
    document.body.appendChild(hiddenDiv);
}
hiddenDiv.addEventListener('myCustomEvent', function() {
    var eventData = document.getElementById('myCustomEventDiv').innerText;
    console.log('收到自定义事件消息: ' + eventData);
});
```

7.3. 长连接和短连接

其实上面已经涉及到了, 这里再单独说明一下。Chrome 插件中有 2 种通信方式, 一个是短连接 (`chrome.tabs.sendMessage` 和 `chrome.runtime.sendMessage`), 一个是长连接 (`chrome.tabs.connect` 和 `chrome.runtime.connect`)。

短连接的话就是挤牙膏一样, 我发送一下, 你收到了再回复一下, 如果对方不回复, 你只能重新发, 而长连接类似 `WebSocket` 会一直建立连接, 双方可以随时互发消息。

短连接上面已经有代码示例了, 这里只讲一下长连接。

`popup.js`:

```
getCurrentTabId((tabId) => {
    var port = chrome.tabs.connect(tabId, {name: 'test-connect'});
    port.postMessage({question: '你是谁啊?'});
    port.onMessage.addListener(function(msg) {
        alert('收到消息: ' + msg.answer);
        if(msg.answer && msg.answer.startsWith('我是'))
        {
            port.postMessage({question: '哦, 原来是你啊!'});
        }
    });
});
```

`content-script.js`:


```
// 监听长连接
chrome.runtime.onConnect.addListener(function(port) {
  console.log(port);
  if(port.name === 'test-connect') {
    port.onMessage.addListener(function(msg) {
      console.log('收到长连接消息: ', msg);
      if(msg.question === '你是谁啊? ') port.postMessage({answer: '我是你爸! '});
    });
  }
});
```

8. 其它补充

8.1. 动态注入或执行JS

虽然在 `background` 和 `popup` 中无法直接访问页面DOM，但是可以通过 `chrome.tabs.executeScript` 来执行脚本，从而实现访问web页面的DOM（注意，这种方式也不能直接访问页面JS）。

示例 `manifest.json` 配置：

```
{
  "name": "动态JS注入演示",
  ...
  "permissions": [
    "tabs", "http://*/*", "https://*/*"
  ],
  ...
}
```

JS:

```
// 动态执行JS代码
chrome.tabs.executeScript(tabId, {code: 'document.body.style.backgroundColor="red"'})
// 动态执行JS文件
chrome.tabs.executeScript(tabId, {file: 'some-script.js'});
```

8.2. 动态注入CSS

示例 `manifest.json` 配置：

```
{
  "name": "动态CSS注入演示",
  ...
  "permissions": [
    "tabs", "http://*/*", "https://*/*"
  ],
  ...
}
```

JS代码：

```
// 动态执行CSS代码，TODO，这里有待验证
chrome.tabs.insertCSS(tabId, {code: 'xxx'});
// 动态执行CSS文件
chrome.tabs.insertCSS(tabId, {file: 'some-style.css'});
```

8.3. 获取当前窗口ID

```
chrome.windows.getCurrent(function(currentWindow)
{
  console.log('当前窗口ID: ' + currentWindow.id);
});
```

8.4. 获取当前标签页ID

一般有2种方法:

```
// 获取当前选项卡ID
function getCurrentTabId(callback)
{
  chrome.tabs.query({active: true, currentWindow: true}, function(tabs)
  {
    if(callback) callback(tabs.length ? tabs[0].id: null);
  });
}
```

获取当前选项卡id的另一种方法, 大部分时候都类似, 只有少部分时候会不一样 (例如当窗口最小化时)

```
// 获取当前选项卡ID
function getCurrentTabId2()
{
  chrome.windows.getCurrent(function(currentWindow)
  {
    chrome.tabs.query({active: true, windowId: currentWindow.id}, function(tabs)
    {
      if(callback) callback(tabs.length ? tabs[0].id: null);
    });
  });
}
```

8.5. 本地存储

本地存储建议用 `chrome.storage` 而不是普通的 `localStorage`, 区别有好几点, 个人认为最重要的2点区别是:

- `chrome.storage` 是针对插件全局的, 即使你在 `background` 中保存的数据, 在 `content-script` 也能获取到;
- `chrome.storage.sync` 可以跟随当前登录用户自动同步, 这台电脑修改的设置会自动同步到其它电脑, 很方便, 如果没有登录或者未联网则先保存到本地, 等登录了再同步至网络;

需要声明 `storage` 权限, 有 `chrome.storage.sync` 和 `chrome.storage.local` 2种方式可供选择, 使用示例如下:

```
// 读取数据, 第一个参数是指定要读取的key以及设置默认值
chrome.storage.sync.get({color: 'red', age: 18}, function(items) {
  console.log(items.color, items.age);
});
// 保存数据
chrome.storage.sync.set({color: 'blue'}, function() {
  console.log('保存成功! ');
});
```

8.6. webRequest

通过webRequest系列API可以对HTTP请求进行任性地修改、定制, 这里通

过 `beforeRequest` 来简单演示一下它的冰山一角：

```
//manifest.json
{
  // 权限申请
  "permissions":
  [
    "webRequest", // web请求
    "webRequestBlocking", // 阻塞式web请求
    "storage", // 插件本地存储
    "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
  ],
}

// background.js
// 是否显示图片
var showImage;
chrome.storage.sync.get({showImage: true}, function(items) {
  showImage = items.showImage;
});
// web请求监听，最后一个参数表示阻塞式，需单独声明权限：webRequestBlocking
chrome.webRequest.onBeforeRequest.addListener(details => {
  // cancel 表示取消本次请求
  if(!showImage && details.type === 'image') return {cancel: true};
  // 简单的音视频检测
  // 大部分网站视频的type并不是media，且视频做了防下载处理，所以这里
  if(details.type === 'media') {
    chrome.notifications.create(null, {
      type: 'basic',
```

8.7. 国际化

插件根目录新建一个名为 `_locales` 的文件夹，再在下面新建一些语言的文件夹，如 `en`、`zh_CN`、`zh_TW`，然后再在每个文件夹放入一个 `messages.json`，同时必须在清单文件中设置 `default_locale`。

`_locales/en/messages.json` 内容：

```
{
  "pluginDesc": {"message": "A simple chrome extension demo"},
  "helloWorld": {"message": "Hello World!"}
}
```

`_locales/zh_CN/messages.json` 内容：

```
{
  "pluginDesc": {"message": "一个简单的Chrome插件demo"},
  "helloWorld": {"message": "你好啊，世界！"}
}
```

在 `manifest.json` 和 `CSS` 文件中通过 `__MSG_messagename__` 引入，如：

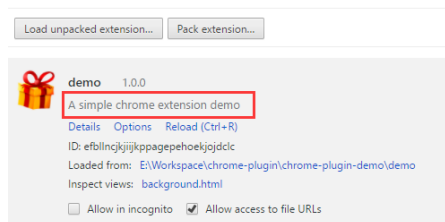
```
{
  "description": "__MSG_pluginDesc__",
  // 默认语言
  "default_locale": "zh_CN",
}
```

JS中则直接 `chrome.i18n.getMessage("helloWorld")`。

测试时，通过给chrome建立一个不同的快捷方式 `chrome.exe --lang=en` 来切换语言，如：



英文效果：



中文效果：



8.8. API总结

比较常用的一些API系列：

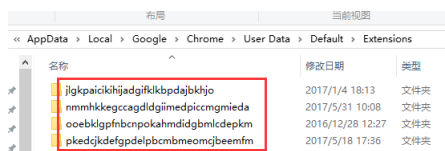
- `chrome.tabs`
- `chrome.runtime`
- `chrome.webRequest`
- `chrome.window`
- `chrome.storage`
- `chrome.contextMenus`
- `chrome.devtools`
- `chrome.extension`

9. 经验总结

9.1. 查看已安装插件路径

已安装的插件源码路径：

`C:\Users\用户名\AppData\Local\Google\Chrome\User Data\Default\Extensions`，每一个插件被放在以插件ID为名的文件夹里面，想要学习某个插件的某个功能是如何实现的，看人家的源码是最好的方法了：



如何查看某个插件的ID? 进入 `chrome://extensions`，然后勾选开发者模式即

可看到了。



9.2. 特别注意background的报错

很多时候你发现你的代码会莫名其妙的失效，找来找去又找不到原因，这时打开background的控制台才发现原来某个地方写错了导致代码没生效，正式由于background报错的隐蔽性(需要主动打开对应的控制台才能看到错误)，所以特别注意这点。

9.3. 如何让popup页面不关闭

在对popup页面审查元素的时候popup会被强制打开无法关闭，只有控制台关闭了才可以关闭popup，原因很简单：如果popup关闭了控制台就没用了。这种方法在某些情况下很实用！

9.4. 不支持内联JavaScript的执行

也就是不支持将js直接写在html中，比如：

```
<input id="btn" type="button" value="收藏" onclick="test()"/>
```

报错如下：

```
Refused to execute inline event handler because it violates the following Content Security
```

解决方法就是用JS绑定事件：

```
$('#btn').on('click', function(){alert('测试')});
```

另外，对于A标签，这样写 `href="javascript:;"` 然后用JS绑定事件虽然控制台会报错，但是不受影响，当然强迫症患者受不了的话只能写成 `href="#"` 了。

如果这样写：

```
<a href="javascript:;" id="get_secret">请求secret</a>
```

报错如下：

```
Refused to execute JavaScript URL because it violates the following Content Security Po
```

9.5. 注入CSS的时候必须小心

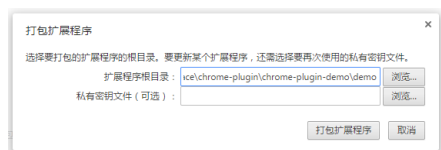
由于通过 `content_scripts` 注入的CSS优先级非常高，几乎仅次于浏览器默认样式，稍不注意就会影响一些网站的展示效果，所以尽量不要写一些影响全局的样式。

之所以强调这个，是因为这个带来的问题非常隐蔽，不太容易找到，可能你正在写某个网页，昨天样式还是好好的，怎么今天就突然不行了？然后你辛辛苦苦找来找去，找了半天才发现竟然是因为插件里面的一个样式影响的！



10. 打包与发布

打包的话直接在插件管理页有一个打包按钮：



然后会生成一个 `.crx` 文件，要发布到Google应用商店的话需要先登录你的Google账号，然后花5个\$注册为开发者，本人太穷，就懒得亲自验证了，有发布需求的自己去整吧。

Chrome 网上应用商店
Chrome 网上应用商店是一个针对网络应用的开放式购物平台。立即开始上传您的应用吧！

要验证您的开发和发布项目，您需要支付 US\$5.00 的一次性开发者注册费。立即支付此费用。

11. 参考

11.1. 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](#)
- [Chrome插件官方示例](#)
- [manifest清单文件](#)
- [permissions权限](#)
- [chrome.xxx.api文档](#)
- [模糊匹配规则语法详解](#)

11.2. 第三方资料

部分中文资料，不是特别推荐：

- [360安全浏览器开发文档](#)
- [360极速浏览器Chrome扩展开发文档](#)
- [Chrome扩展开发极客系列博客](#)

12. 附图

附图：Chrome高清png格式logo：



posted @ 2017-07-11 09:30 我是小茗同学 阅读(148334) 评论(120) 编辑 收藏

< Prev

1

2

3

评论列表

#101楼 2018-08-14 15:58 s_p

google 设置 通过命令如何修改呢？

#102楼 2018-08-16 17:03 谣言似山

火钳刘明，你的URL出现在RedCore的注释里了~~~哈哈哈哈哈
<https://www.zhihu.com/question/290426793/answer/470060833>

#103楼 2018-08-17 09:53 jason31

瞻仰价值2亿的代码

#104楼 [楼主] 2018-08-18 22:27 我是小茗同学

@ 谣言似山

[引用](#)

火钳刘明，你的URL出现在RedCore的注释里了~~~哈哈哈哈哈
<https://www.zhihu.com/question/290426793/answer/470060833>

哈哈，感谢告知，我也是今天才发现，哥，可不可以发一个红芯浏览器给我“学习学习”，哈哈~，我邮箱 me@haoji.me

#105楼 [楼主] 2018-08-18 22:31 我是小茗同学

@ jason31

[引用](#)

瞻仰价值2亿的代码

这个红芯浏览器还真是把我震惊了，哈哈，注释都懒得改

#106楼 2018-10-15 11:49 MyNameIsDream

博主，你好，我在js中用chrome.runtime.connect，建立长连接，可是控制台报错Uncaught TypeError: Cannot read property 'connect' of undefined，请问你知道原因吗，谢谢

#107楼 2018-10-19 12:35 stono

非常感谢！我按着教程写了一个屏蔽某度的广告推荐，比之前下载的xxblock好用多了！
非常感谢！

#108楼 2018-11-13 14:32 -云-

最近再看chrome插件开发，一般情况，这篇文章就够用了。
价值2.5亿的代码。

#109楼 2018-11-26 10:49 —Renyi—

看完了 谢谢分享

#110楼 2018-12-11 14:26 vkbokeyuan

大神 page action只在百度显示图标这样写不生效啊
"background": {
"scripts": ["js/background.js"] (background.js内容是按照您给的)
},
"permissions": ["declarativeContent"],

chrome.pageAction.show(tabId) 显示图标；这个要怎么用呢

#111楼 2018-12-21 15:31 李秋豪

感谢分享，正好用上。

#112楼 2018-12-27 09:53 imba久期

我好像看到了两个亿 哈哈哈哈哈

#113楼 2018-12-27 20:54 caobingbing

博主，你好
我在用popup.js往content.js发送消息 无法成功，返回的是undefined
content.js
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
// console.log(sender.tab ? "from a content script:" + sender.tab.url
// : "from the extension");
console.log("cotent");
if (request.cmd === 'test')
alert(request.value);
sendResponse('我收到了你的消息！');
});

popup.js
function sendMessageToContentScript(message, callback) {
chrome.tabs.query({
active : true,
currentWindow : true
}, function(tabs) {
chrome.tabs.sendMessage(tabs[0].id, message,
function(response) {
if (callback)
callback(response);
});
});

```
});  
}  
sendMessageToContentScript({  
  cmd : 'test',  
  value : 'popup=====',  
}, function(response) {  
  console.log('content---' + response);  
});
```

#114楼 2019-02-01 10:07 imba久期

上传代码时manifest.json里不能有注释

#115楼 2019-03-05 13:55 凌霄phxism

深度好文

#116楼 2019-03-07 11:28 dounine007

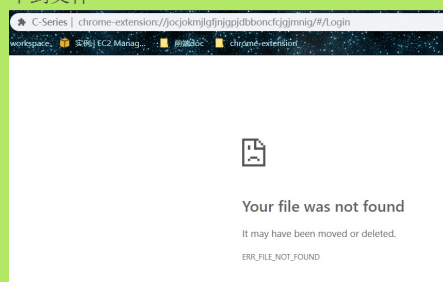
大神您好，我碰到一个问题，我在页面注入了一个content_script.js后点击按钮可与background.js通讯（长连接）【逻辑：点击 --> content_script 向 background 通讯 --> background接受到请求后处理完毕再回传给 content_script --> content_script 监听到数据 打印 console.log(message)】；
问题：页面刷新后 console.log打印一次，再次点击的时候，打印2次，再次点击，打印3次；以此类推；
谢谢指教！

#117楼 2019-05-13 23:51 samous

您好大佬，想知道插件和项目如何通信，比如我想项目做一个类似postman接口测试的工具 想通过chrome插件提供cookie和解决跨域 但是不知项目和插件怎么通信

#118楼 2019-05-14 18:00 初心，你好吗

大神，如果有路由的话，正常点击可以跳转，一刷新浏览器就显示找不到文件



#119楼 2019-07-26 09:28 悟空师兄

楼主有联系方式吗？有项目可以合作,2996475653

#120楼 2019-07-30 18:23 一字见心

楼主，我想做个扩展程序，访问每个网页时能自动生成页面的大纲，就像你这个页面也有大纲一样，能联系一下吗，有预算费用的。qq: 28488238

文章目录:

1. 写在前面

2. 前言

2.1. 什么是Chrome插件

2.2. 学习Chrome插件开发有什...

2.3. 为什么是Chrome插件而不...

3. 开发与调试

4. 核心介绍

4.1. manifest.json

4.2. content-scripts

4.3. background

4.4. event-pages

4.5. popup

4.6. injected-script

4.7. homepage_url

5. Chrome插件的8种展示形式


< Prev

1

2

3

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)， [访问](#) 网站首页。

~~~~~ 华丽的分割线 ~~~~~

【推荐】超50万C++/C#源码: 大型实时仿真组态图形源码

【推荐】零基础轻松玩转云上产品，获壕礼加返百元大礼

【推荐】华为IoT平台开发者套餐9.9元起，购买即送免费课程