22min

第 5 章

CHAPTER 5

Manifest 详解

Manifest(清单)文件是浏览器扩展的蓝图。广义上讲,它是一个配置文件,包含一系列键-值对,规定了扩展可以做什么及 Manifest 以何种方式做。

清单的内容因清单版本和浏览器而异,并非所有浏览器都能使用所有属性;有些浏览器会部分支持某个属性,甚至完全不支持。从 Manifest V2 到 Manifest V3 会添加新的属性,删除其他属性,有些属性的定义方式也会发生变化。

5.1 清单文件

清单文件是浏览器扩展的重要组成部分。它以 JSON 格式(键-值对的集合)定义了扩展的各种属性,包括名称、版本号、权限、页面、脚本等。这个清单告诉浏览器如何加载和管理扩展。以下展示 Manifest 的结构和常见用法。

(1) 最小化配置,代码如下:

```
//第 5 章,5.1
{
  "manifest_version": 3,
  "name": "Minimal Manifest",
  "version": "1.0.0",
  "description": "A basic example extension with only required keys",
  "icons": {
    "48": "images/icon - 48.png",
    "128": "images/icon - 128.png"
  },
}
```

(2) 注册内容脚本,代码如下:

```
//第 5 章,5.1
{
  "manifest_version": 3,
```

```
  "name": "Run script automatically",
  "description": "Runs a script on www.example.com automatically when user installs the
extension",
  "version": "1.0",
  "icons": {
    "16": "images/icon-16.png",
    "32": "images/icon-32.png",
    "48": "images/icon-48.png",
    "128": "images/icon-128.png"
  },
  "content_scripts": [
    {
      "js": [
        "content-script.js"
      ],
      "matches": [
        "http://*.example.com//"
      ]
    }
  ]
}
```

(3) 注入内容脚本,代码如下:

```
//第5章,5.1
{
  "manifest_version": 3,
  "name": "Click to run",
  "description": "Runs a script when the user clicks the action toolbar icon.",
  "version": "1.0",
  "icons": {
    "16": "images/icon-16.png",
    "32": "images/icon-32.png",
    "48": "images/icon-48.png",
    "128": "images/icon-128.png"
  },
  "background": {
    "service_worker": "service-worker.js"
  },
  "action": {
    "default_icon": {
      "16": "images/icon-16.png",
      "32": "images/icon-32.png",
      "48": "images/icon-48.png",
      "128": "images/icon-128.png"
    }
  },
  "permissions": ["scripting", "activeTab"]
}
```

(4) 配置弹出脚本,代码如下:

```
//第 5 章,5.1
{
  "manifest_version": 3,
  "name": "Popup extension that requests permissions",
  "description": "Extension that includes a popup and requests host permissions and storage
permissions .",
  "version": "1.0",
  "icons": {
    "16": "images/icon - 16.png",
    "32": "images/icon - 32.png",
    "48": "images/icon - 48.png",
    "128": "images/icon - 128.png"
  },
  "action": {
    "default_popup": "popup.html"
  },
  "host_permissions": [
    "https://*.example.com/"
  ],
  "permissions": [
    "storage"
  ]
}
```

(5) 配置侧边栏,代码如下:

```
//第 5 章,5.1
{
  "manifest_version": 3,
  "name": "Side panel extension",
  "version": "1.0",
  "description": "Extension with a default side panel.",
  "icons": {
    "16": "images/icon - 16.png",
    "48": "images/icon - 48.png",
    "128": "images/icon - 128.png"
  },
  "side_panel": {
    "default_path": "sidepanel.html"
  },
  "permissions": ["sidePanel"]
}
```

5.2 国际化与模式匹配

5.2.1 国际化配置

浏览器扩展的国际化配置是一种使扩展支持多种语言和地区的方法。它允许开发者根据用户的首选语言，动态地显示不同语言版本的扩展界面文本、按钮标签、提示信息等内容，提高了扩展的可用性和用户体验。

国际化配置通常使用JSON文件来管理不同语言的文本资源。在扩展开发中，需要创建一个消息目录，其中包含多个语言版本的JSON文件。每个文件都包含一个键-值对，其中键是标识符，值是对应语言下的文本内容。

以简单的浏览器扩展为例，展示如何配置国际化支持。这个扩展是一个简单的页面计数器，当用户单击按钮时，页面上的计数数字会增加。

1. 创建消息目录

首先，在扩展的根目录下创建一个名为 _locales 的文件夹，里面包含不同语言的文件夹。例如，创建英语(en)和法语(fr)版本：

```
_locales
    ├── en
    │   └── messages.json
    └── fr
        └── messages.json
```

2. messages.json 文件内容

messages.json文件是用来存储不同语言的键-值对的。例如，在英语版本的 messages.json 文件中的代码如下：

```
//第 5 章,5.2.1

//en/messages.json
{
  "pageTitle": {
    "message": "Page Counter"
  },
  "countLabel": {
    "message": "Count: "
  },
  "incrementButton": {
    "message": "Increment"
  }
}
```

而在法语版本的 messages.json 文件中的代码如下：

```
//第 5 章,5.2.1
//fr/messages.json
{
  "pageTitle": {
    "message": "Compteur de Pages"
  },
  "countLabel": {
    "message": "Compte : "
  },
  "incrementButton": {
    "message": "Incrémenter"
  }
}
```

3. 引用国际化文本

在扩展的 HTML 文件中,使用国际化消息的标识符来引用文本内容。例如,HTML 文件中的一段示例代码如下：

```
//第 5 章,5.2.1
<!DOCTYPE html>
<html>
<head>
  <title>Page Counter</title>
</head>
<body>
  <h1><span id="pageTitle">Page Counter</span></h1>
  <p><span id="countLabel">Count: </span><span id="count">0</span></p>
  <button id="incrementButton">Increment</button>

  <script>
    document.getElementById('incrementButton').addEventListener('click', function() {
      let countElement = document.getElementById('count');
      let count = parseInt(countElement.innerText);
      countElement.innerText = count + 1;
    });
  </script>
</body>
</html>
```

和<button>的文本内容使用了一个特殊的 ID,而不是直接写文本。这些 ID 对应了 messages.json 文件中的键。浏览器扩展在加载页面时会根据用户的语言设置动态地将这些文本内容替换为对应语言版本的文本。

国际化配置使浏览器扩展能够在不同的语言环境下提供友好的用户界面。通过创建不同语言版本的消息文件,并在扩展中引用这些消息文件的键,可以实现在用户选择的语言环

境下动态地显示对应的文本内容，为用户提供更加舒适和贴近本地化的使用体验。

5.2.2 模式匹配

清单文件中的一些属性可以使用模式匹配同时指定多个 URL 或文件。

匹配模式是具有这种< scheme >://< host >/< path >结构的 URL，用于指定一组 URL。

(1) scheme：支持的类型有 http、https、file、*，* 号表示支持 https 和 http。

(2) host：是主机名(www.example.com)。主机名前的 * 用于匹配子域(*.example.com)或通配符 * 。如果在主机模式中使用通配符，则通配符必须是第 1 个或唯一的字符，并且后面必须跟一个句点(.)或正斜线(/)。

(3) path：必须至少包含一条正斜线。斜线本身可以匹配任何路径，就像后面跟了一个通配符(/ *)。

一些特殊情况如下。

(1) all_urls：匹配任何以允许方案开头的 URL，包括有效模式下列出的任何模式。由于它会影响所有主机，因此 Chrome 浏览器网站商店对使用该功能的扩展的审核时间可能会更长。

(2) file://：允许扩展在本地文件上运行。这种模式要求用户手动授予访问权限。需要注意，这种情况需要 3 条斜线，而不是两条。

(3) 本地主机 URL 和 IP 地址：如果要在开发过程中匹配任何本地主机端口，则可使用 http://localhost/ * 。对于 IP 地址，应在路径中指定地址加通配符，如 http://127.0.0.1/ * 。也可以使用 http://*:*/*来匹配 localhost、IP 地址和任何端口。

(4) 顶级域匹配模式：Chrome 浏览器不支持顶级域(TLD)的匹配模式。应在单个 TLD 中指定匹配模式，如 http://google.es/ * 和 http://google.fr/ * 。

5.3 Manifest 属性

5.3.1 必填属性

1. manifest_version

manifest_version 属性指定了使用的清单文件的版本。在 Manifest V3 中，这个属性的值应为 3，以表明使用的是 Manifest V3 版本格式，代码如下：

```
{
  "manifest_version": 3,
  ...
}
```

强制性属性：在 Manifest V3 中，这是一个必需的属性，指定了使用的清单文件格式版本。如果未指定或指定错误，则扩展将无法正确加载并运行。不同于 V2：Manifest V3 与 V2 相比有许多不同之处，包括权限管理、生命周期和 API 的改变。使用 V3 需要考虑这些变化，并相应地修改扩展。V3 版本可能带来性能提升和更好的安全性，但需要更严格的代码规范和适应新的 API。

注意 未定义或错误定义 manifest_version 属性会导致扩展无法正确识别清单文件的版本，因此无法加载或运行。在迁移或创建新的扩展时，正确设置 manifest_version 至关重要，以确保扩展可以在 Manifest V3 环境下正常工作。

2. name

name 属性指定了扩展的名称，显示在 Chrome 扩展管理器中或在其他扩展相关的用户界面中。这个属性定义了用户能够识别和区分扩展的标识名称，代码如下：

```
{
  "name": "My Awesome Extension",
  …
}
```

扩展的名称应该清晰、简洁，并且能够准确地描述扩展的功能或用途，以便用户能够轻松识别和记住。名称应符合 Chrome Web Store 的命名规范，不包含违反规定的内容或敏感信息，以避免审核或展示问题。好的名称能够提升用户对扩展的印象，有助于用户选择和使用扩展。

注意 如果未定义 name 属性，或者定义了一个不明确或不合规的名称，则会影响用户对扩展的理解和使用体验。正确设置"name"属性能够帮助用户准确地识别和记住扩展。

3. version

version 属性指定了扩展的版本号。这个属性用于标识扩展的不同版本，帮助用户和开发者了解扩展的更新情况，代码如下：

```
{
  "version": "1.0.0",
  …
}
```

通常遵循“主版本号.次版本号.修订号”的格式，每次更新版本号时应根据变化程度进行适当修改。版本号变化应该遵循一定的规律和逻辑，确保高版本号代表更新或功能增加。版本号的变化应反映扩展的内容和功能变化，帮助用户了解更新内容。

注意 未定义 version 属性或者版本号格式不规范会导致扩展在发布、更新或管理时出现问题。准确设置版本号能够帮助用户和开发者了解扩展的变化和更新情况，从而更好地管理和使用扩展。

5.3.2 推荐属性

1. action

用于控制扩展在谷歌 Chrome 工具栏中的图标。每个扩展都有一个图标显示在 Chrome 工具栏中，即使在 Manifest 中没有添加 action。当它未包含在内时，扩展仍然会显示在工具栏中，以提供对扩展菜单的访问，代码如下：

```
//第 5 章,5.3.2 manifest_1.json

{
"manifest_version": 3,
"name": "我的插件",
"version": "1.0",
"action": {
"default_icon": {
"16": "images/icon16.png",
"24": "images/icon24.png",
"32": "images/icon32.png"
},
"default_title": "单击我",
"default_popup": "popup.html"
},
...
}
```

在这个示例中，action 属性包含了 default_icon、default_title 和 default_popup 共 3 个子属性。default_icon 定义了扩展图标的路径和大小，default_title 定义了鼠标悬停在扩展图标上时显示的提示信息，default_popup 定义了单击扩展图标后弹出的页面的位置。

注意 建议总是至少包含 action 和 default_icon 键，因为所有的扩展都会在工具栏中显示图标，如果扩展没有图标，则 Chrome 会为它自动生成一个。此外，还可以通过 action.setIcon()方法来动态地设置扩展图标。

2. default_locale

这个属性用于定义扩展支持多语言时的默认语言。它是_locales 子目录中包含此扩展的默认语言的子目录的名称，代码如下：

```
//第 5 章,5.3.2 manifest_2.json

{
"manifest_version": 3,
"name": "我的插件",
"version": "1.0",
"default_locale": "en"
}
```

在这个示例中,default_locale 属性被设置为"en",这意味着这个扩展的默认语言是英语。

注意 default_locale 字段对于本地化的扩展(那些有_locales 目录的扩展)是必需的,但是对于没有_locales 目录的扩展,这个字段必须缺失,因此,如果扩展支持多语言,就需要定义 default_locale 属性。

3. description

这个属性是一个纯文本字符串(没有 HTML 或其他格式化;不超过 132 个字符),用于描述扩展,例如 description: A description of my extension。这个描述应该适用于浏览器的扩展页面(chrome://extensions)和 Chrome 网上商店,代码如下:

```
//第 5 章,5.3.3 manifest_3.json

{
"manifest_version": 3,
"name": "我的插件",
"version": "1.0",
"description": "这是我的插件的描述"
}
```

注意 description 字段是一个纯文本字符串,不应包含 HTML 或其他格式化内容,并且长度不应超过 132 个字符。此外,这个描述应该适合于浏览器的扩展页面和 Chrome 网上商店。

4. icons

icons 是 Chrome 浏览器插件 Manifest V3 版本的清单文件中的一个属性。这个属性用于表示扩展或主题的一个或多个图标。应该总是提供一个 128×128 大小的图标;它在安装过程中及在 Chrome 网上商店中使用。扩展还应该提供一个 48×48 大小的图标,它在扩展管理页面(chrome://extensions)中使用。还可以指定一个 16×16 大小的图标,用作扩展页面的 favicon。图标通常应该是 PNG 格式,因为 PNG 对透明度的支持最好,然而,它们

可以是 Blink 支持的任何光栅格式，包括 BMP、GIF、ICO 和 JPEG，代码如下：

```
//第 5 章,5.3.4 manifest_4.json

{
  "manifest_version": 3,
  "name": "我的插件",
  "version": "1.0",
  "icons": {
    "16": "icon16.png",
    "32": "icon32.png",
    "48": "icon48.png",
    "128": "icon128.png"
  },
  …
}
```

在这个示例中，icons 属性包含了 16、32、48 和 128 共 4 个子属性，分别对应着不同大小的图标的路径。

注意 可以提供任何所希望的其他大小的图标，Chrome 会尝试在适当的地方使用最佳大小，但是 WebP 和 SVG 文件是不支持的。

5. author

author 接受一个带有 email 键的对象。这是扩展的作者的电子邮件地址。当将 CRX 文件发布到 Chrome 网上商店时，这个字符串必须匹配用于发布扩展的账号的电子邮件地址，代码如下：

```
//第 5 章,5.3.5 manifest_5.json

{
  "manifest_version": 3,
  "name": "我的插件",
  "version": "1.0",
  "author": {
    "email": "user@example.com"
  },
  …
}
```

在这个示例中，author 属性包含了一个 email 子属性，对应着扩展的作者的电子邮件地址。

注意 当将 CRX 文件发布到 Chrome 网上商店时，author 属性中的 email 键的值必须匹配用于发布扩展的账号的电子邮件地址。

6. background

background 属性用于指定一个 JavaScript 文件作为扩展的服务工作线程。服务工作线程是一个后台脚本,充当扩展的主要事件处理程序,代码如下:

```
//第 5 章,5.3.6 manifest_6.json

{
"manifest_version": 3,
"name": "我的插件",
"version": "1.0",
"background": {
"service_worker": "background.js",
"type": "module"
}
}
```

在这个示例中,background 属性包含了 Service Worker 和 type 两个子属性。Service Worker 定义了服务工作线程的路径,type 定义了服务工作线程的类型。

注意 在 Manifest V3 扩展中,Chrome 只支持服务工作线程作为后台脚本。此外,还可以通过将 type 属性设置为 module 来将服务工作线程加载为 ES 模块,这样就可以在服务工作线程中使用 import 关键字来导入其他模块了。

7. chrome_settings_overrides

chrome_settings_overrides 属性是一个对象,可以包含属性 homepage 和 search_provider。homepage 用于定义浏览器的主页页面。search_provider 用于定义要添加到浏览器的搜索提供商,代码如下:

```
//第 5 章,5.3.7 manifest_7.json

{
"manifest_version": 3,
"name": "我的插件",
"version": "1.0",
"chrome_settings_overrides": {
"homepage": "https://www.homepage.com",
"search_provider": {
"name": "name.__MSG_url_domain__",
"keyword": "keyword.__MSG_url_domain__",
"search_url": "https://bing.com/search?q=%s{searchTerms}",
"favicon_url": "https://bing.com/search?q=",
"suggest_url": "https://bing.com/search?q=%s{searchTerms}",
"instant_url": "https://bing.com/search?q=%s{searchTerms}",
"image_url": "https://bing.com/search?q=%s{searchTerms}",
```

```
"search_url_post_params": "search_lang = __MSG_url_domain__",
"suggest_url_post_params": "suggest_lang = __MSG_url_domain__",
"instant_url_post_params": "instant_lang = __MSG_url_domain__",
"image_url_post_params": "image_lang = __MSG_url_domain__",
"alternate_urls": [
"https://bing.com/search?q = {searchTerms}",
"https://bing.com/search?q = {searchTerms}"
],
"encoding": "UTF - 8",
"is_default": true
}
}
}
```

在这个示例中，chrome_settings_overrides 属性包含 homepage 和 search_provider 两个子属性。homepage 定义了浏览器的主页页面，search_provider 定义了要添加到浏览器的搜索提供商。

注意 chrome_settings_overrides 属性可以用来覆盖浏览器的主页和添加新的搜索引擎。如果两个或更多的扩展都设置了这个值，则最近安装的那个扩展的设置将优先。

8. chrome_url_overrides

chrome_url_overrides 属性是一个对象，可以包含以下属性。

(1) newtab：提供一个替代的新标签页文档。

(2) bookmarks：提供一个替代的显示书签的页面。

(3) history：提供一个替代的显示浏览历史的页面。

代码如下：

```
//第 5 章,5.3.8 manifest_8.json

{
  "manifest_version": 3,
  "name": "我的插件",
  "version": "1.0",
  "chrome_url_overrides": {
    "newtab": "my-new-tab.html",
    "bookmarks": "my-bookmarks.html",
    "history": "my-history.html"
  },
  ...
}
```

在这个示例中，chrome_url_overrides 属性包含 newtab、bookmarks 和 history 共 3 个子属性。这些属性分别定义了新标签页、书签页和历史页的替代页面。

注意 chrome_url_overrides 属性可以用来提供浏览器的特殊页面的自定义替代。如果两个或更多的扩展都定义了自定义的新标签页,则最后安装或启用的那个扩展的值将被使用。

9. commands

commands 属性是一个对象,每个快捷键都是它的一个属性。每个快捷键的值是一个对象,最多可以有两个属性。

(1) suggested_key:激活快捷键的键组合。

(2) description:为用户提供快捷键的目的的简短描述。

代码如下:

```
//第 5 章,5.3.9 manifest_9.json

{
  "manifest_version": 3,
  "name": "我的插件",
  "version": "1.0",
  "commands": {
    "toggle - feature": {
      "suggested_key": {
        "default": "Ctrl + Shift + Y",
        "mac": "Command + Shift + Y"
      },
      "description": "切换功能"
    }
  },
  …
}
```

在这个示例中,commands 属性包含一个 toggle-feature 子属性。这个子属性定义了一个快捷键,包括它的键组合和描述。

注意 扩展命令快捷键必须包含 Ctrl 或 Alt。不能在媒体键上使用修饰符。在 macOS 系统中,Ctrl 会被自动转换为 Command。如果要在 macOS 上使用 Control 键,则可以在定义 macOS 快捷键时将 Ctrl 替换为 MacCtrl。在其他平台的组合中使用 MacCtrl 会导致验证错误,并阻止扩展安装。

10. content_scripts

content_scripts 属性用于指定一个静态加载的 JavaScript 或 CSS 文件,每当打开匹配某个 URL 模式的页面时都会使用这个文件。扩展也可以以编程的方式注入内容脚本,代码如下:

```
//第 5 章,5.3.10 manifest_10.json

{
  "manifest_version": 3,
  "name": "我的插件",
  "version": "1.0",
  "content_scripts": [
    {
      "matches": ["https://*.example.com/*"],
      "css": ["my-styles.css"],
      "js": ["content-script.js"],
      "Exclude_matches": ["*://*/*foo*"],
      "include_globs": ["*example.com/???s/*"],
      "Exclude_globs": ["*bar*"],
      "all_frames": false,
      "match_origin_as_fallback": false,
      "match_about_blank": false,
      "run_at": "document_idle",
      "world": "ISOLATED"
    }
  ],
  ...
}
```

在这个示例中,content_scripts 属性包含一个对象,这个对象定义了一个内容脚本,包括它的匹配模式、CSS 文件、JavaScript 文件、排除的匹配模式、包含的全局模式、排除的全局模式、是否在所有框架中运行、是否在原始匹配失败时作为备选匹配、是否匹配 about:blank 页面、何时运行,以及运行的事件。

注意 content_scripts 属性可以用来指定一个静态加载的 JavaScript 或 CSS 文件,每当打开匹配某个 URL 模式的页面时都会使用这个文件。扩展也可以以编程的方式注入内容脚本。

11. content_security_policy

content_security_policy 用于定义扩展可以使用的脚本、样式和其他资源的限制。在这个属性中,可以为扩展页面和沙箱扩展页面定义不同的策略,代码如下:

```
//第 5 章,5.3.11 manifest_11.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "content_security_policy": {
        "extension_pages": "script-src 'self' 'wasm-unsafe-eval'; object-src 'self';"
```

```
    },
    "permissions": ["activeTab"]
}
```

在这个示例中,扩展只能加载其自己打包资源中的本地脚本和对象。

注意 如果没有在 Manifest 中定义 content_security_policy,则将使用默认的属性。对于扩展页面,Chrome 强制执行最小的内容安全策略。这个策略不能被放宽。对于沙箱页面,内容安全策略可以根据需要进行自定义。

在 Manifest V3 中,所有引用外部或非静态内容的 CSP 来源都是被禁止的。只允许 none、self 和 wasm-unsafe-eval 这些值。

在设计扩展时,需要根据需求来选择使用 content_security_policy 属性还是处理单击事件。如果同时定义了 content_security_policy 属性和单击事件的处理函数,则单击事件的处理函数将不会被触发。因为 content_security_policy 属性的优先级更高,所以在设计扩展时,需要根据需求来选择使用 content_security_policy 属性还是处理单击事件。

12. cross_origin_embedder_policy

cross_origin_embedder_policy 是 Chrome 扩展的一个属性,这个属性在 Chrome 93 中引入,它允许扩展为其来源的请求指定 Cross-Origin-Embedder-Policy (COEP) 响应头的值。这包括扩展的服务工作线程、弹出窗口、选项页面、打开到扩展资源的标签等,代码如下:

```
//第 5 章,5.3.12 manifest_12.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "cross_origin_embedder_policy": {
        "value": "require-corp"
    },
    "permissions": ["activeTab"]
}
```

在这个示例中,Chrome 会使用 require-corp 作为从扩展的来源提供资源时的 Cross-Origin-Embedder-Policy 头的值。

注意 cross_origin_embedder_policy 属性包含一个名为 value 的属性,该属性接受一个字符串,Chrome 使用这个字符串作为从扩展提供资源时的 Cross-Origin-Embedder-Policy 头的值。

13. cross_origin_opener_policy

cross_origin_opener_policy 属性在 Chrome 93 中引入。它允许扩展选择加入跨源隔离，允许扩展为其请求指定 Cross-Origin-Opener-Policy（COOP）响应头的值。这包括扩展的服务工作线程、弹出窗口、选项页面、打开到扩展资源的标签等，代码如下：

```
//第 5 章,5.3.13 manifest_13.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "cross_origin_opener_policy": {
        "value": "same - origin"
    },
    "permissions": ["activeTab"]
}
```

在这个示例中，Chrome 会使用 same-origin 作为从扩展提供资源时的 Cross-Origin-Opener-Policy 头的值。

注意 cross_origin_opener_policy 属性包含一个名为 value 的属性，该属性接受一个字符串。

---

14. declarative_net_request

declarative_net_request 属性在 Chrome 94 中引入。它允许扩展通过指定声明性规则来阻止或修改网络请求。这让扩展可以在不拦截和查看内容的情况下修改网络请求，从而提供更多的隐私，代码如下：

```
//第 5 章,5.3.14 manifest_14.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "declarative_net_request": {
        "rule_resources": [
            {
                "id": "ruleset_1",
                "enabled": true,
                "path": "rules_1.json"
            },
            {
                "id": "ruleset_2",
                "enabled": false,
```

```
                "path": "rules_2.json"
            }
        ]
    },
    "permissions": [
        "declarativeNetRequest",
        "declarativeNetRequestFeedback"
    ],
    "host_permissions": [
        "http://www.blogger.com/*",
        "http://*.google.com/*"
    ]
}
```

在这个示例中,Chrome 扩展定义了两个规则集,其中一个用于启用;另一个用于禁用,示例代码如下:

```
{
  "id" : 1,
  "priority": 1,
  "action" : { "type" : "block" },
  "condition" : {
    "urlFilter" : "abc",
    "initiatorDomains" : ["foo.com"],
    "resourceTypes" : ["script"]
  }
}
```

注意 declarative_net_request 属性包含一个名为 rule_resources 的属性,该属性接受一个规则集数组。Chrome 使用这个数组来确定从扩展的来源提供资源时的网络请求规则。

15. devtools_page

devtools_page 属性允许扩展浏览器的内置开发者工具。它允许扩展通过指定一个 HTML 文件来扩展浏览器的内置开发者工具。这个 HTML 文件必须与扩展一起打包,URL 是相对于扩展的根目录。当开发者工具窗口打开时,扩展会创建其开发者工具页面的一个实例,只要窗口打开,该实例就会存在,代码如下:

```
//第 5 章,5.3.15 manifest_15.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "devtools_page": "devtools.html",
    "permissions": ["activeTab"]
}
```

注意 devtools_page 属性的值必须是一个指向 HTML 文件的 URL。因为开发者工具页面必须是扩展的本地页面，建议使用相对 URL 来指定它。

使用这个 Manifest 键会触发安装时的权限警告。为了避免安装时的权限警告，可以通过在 optional_permissions Manifest 键中列出 DevTools 权限来将该功能标记为可选。

16. event_rules

event_rules 它提供了一种机制，可以添加规则来拦截、阻止或修改正在传输的网络请求，或者根据页面的内容采取行动，而不需要读取页面的内容的权限，代码如下：

```
//第 5 章,5.3.16 manifest_16.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "event_rules": [
        {
            "event": "declarativeContent.onPageChanged",
            "actions": [
                {
                    "type": "declarativeContent.ShowPageAction"
                }
            ],
            "conditions": [
                {
                    "type": "declarativeContent.PageStateMatcher",
                    "css": ["video"]
                }
            ]
        }
    ],
    "permissions": ["declarativeContent", "activeTab"]
}
```

在这个示例中，当页面发生变化时，如果当前页面有视频 CSS 标签，Chrome 则会显示页面操作，代码如下：

```
chrome.declarativeContent.onPageChanged.addRules([{
  actions: [
    new chrome.declarativeContent.ShowPageAction()
  ],
  conditions: [
    new chrome.declarativeContent.PageStateMatcher(
        {css: ["video"]}
    )
  ]
}]);
```

注意 event_rules 属性的值必须是一个规则数组。Chrome 使用这个数组来确定从扩展的来源提供资源时的网络请求规则。

17. export

export 是 Chrome 扩展的一个属性,它允许扩展将其功能导出为模块,以便其他扩展可以使用,这个属性允许在 Manifest V3 或更高版本中使用,代码如下:

```
//第 5 章,5.3.17 manifest_17.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "export": {
        "resources": ["moduleA.js", "moduleB.js"]
    },
"permissions": ["activeTab"]
}
```

在这个示例中,Chrome 扩展将 moduleA.js 和 moduleB.js 这两个模块导出,以便其他扩展可以使用。

注意 export 属性的值必须是一个包含模块资源的数组。Chrome 使用这个数组来确定从扩展的来源提供资源时的模块导出规则。

18. externally_connectable

externally_connectable 是 Chrome 扩展的一个属性,它声明了哪些扩展和网页可以通过 runtime.connect 和 runtime.sendMessage 连接到扩展。externally_connectable manifest 键包括以下可选属性。

(1) ids:允许连接的扩展的 ID。如果留空或未指定,则没有扩展或应用可以连接。通配符 * 将允许所有的扩展和应用连接。

(2) matches:允许连接的网页的 URL 模式。如果留空或未指定,则没有网页可以连接。模式不能包括通配符域名,也不能是顶级域名的子域名。

(3) accepts_tls_channel_id:允许扩展使用连接到它的网页的 TLS 通道 ID。网页也必须选择向扩展发送 TLS 通道 ID,方法是在 runtime.connect 的 connectInfo 或 runtime.sendMessage 的选项中将 includeTlsChannelId 设置为 true。如果设置为 false,则 runtime.MessageSender.tlsChannelId 将在任何情况下都不会被设置,代码如下:

```
//第 5 章,5.3.18 manifest_18.json

{
```

```
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "externally_connectable": {
        "ids": [
            "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
            "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb"
        ],
        "matches": [
            "https://*.google.com/*",
            "*://*.chromium.org/*"
        ],
        "accepts_tls_channel_id": false
    },
"permissions": ["activeTab"]
}
```

在这个示例中，只有 ID 为 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 和 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 的扩展，以及 URL 匹配 https://*.google.com/* 和 *://*.chromium.org/* 的网页才可以连接到扩展。

注意 如果扩展的 Manifest 中没有声明 externally_connectable，则所有的扩展都可以连接，但是没有网页可以连接，因此，当更新 Manifest 来使用 externally_connectable 时，如果没有指定 "ids":["*"]，则其他的扩展将失去连接到扩展的能力。

---

19. file_browser_handlers

file_browser_handlers 允许扩展注册为至少一种文件类型的处理程序。开发者必须在扩展的 Manifest 中声明 fileBrowserHandler 权限，并且必须使用 file_browser_handlers 字段来注册扩展为至少一种文件类型的处理程序，代码如下：

```
//第 5 章,5.3.19 manifest_19.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "file_browser_handlers": [
        {
            "id": "open",
            "default_title": "Open",
            "file_filters": [
                "filesystem:*.txt"
            ]
        }
    ],
```

```
    "permissions": ["fileBrowserHandler"]
}
```

在这个示例中,当用户在文件浏览器中选择一个.txt 文件并选择 Open 操作时,Chrome 会调用扩展来处理这个文件。

注意 file_browser_handlers 属性允许扩展注册为至少一种文件类型的处理程序。file_browser_handlers 属性的值必须是一个处理程序数组。Chrome 使用这个数组来确定从扩展的来源提供资源时的文件处理规则。

20. file_handlers

file_handlers 用于指定由 ChromeOS 扩展处理的文件类型。如果要处理文件,则可使用 Web 平台的 Launch Handler API,代码如下:

```
//第 5 章,5.3.20 manifest_20.json

{
    "manifest_version": 3,
    "name": "我的扩展",
    "version": "1.0",
    "file_handlers": [
        {
            "action": "/open_text.html",
            "name": "Plain text",
            "accept": {
                "text/plain": [".txt"]
            },
            "launch_type": "single-client"
        }
    ],
    "permissions": ["activeTab"]
}
```

在这个示例中,当用户在文件浏览器中选择一个.txt 文件并选择 Open 操作时,Chrome 会调用扩展来处理这个文件。

注意 file_handlers 属性允许扩展选择加入跨源隔离,这个属性在 Chrome 120 beta 中引入。

21. file_system_provider_capabilities

file_system_provider_capabilities 属性用于定义扩展的文件系统提供者的能力。这个属性告诉 ChromeOS 文件管理器扩展的文件系统提供者支持哪些功能,代码如下:

```
//第 5 章,5.3.21
{
    …
    "permissions": [
        "fileSystemProvider"
    ],
    …
    "file_system_provider_capabilities": {
        "configurable": true,
        "watchable": false,
        "multiple_mounts": true,
        "source": "network"
    },
    …
}
```

在这个示例中,定义了一个文件系统提供者,它支持配置(configurable)、不支持设置观察者和通知更改(watchable),支持多个挂载点(multiple_mounts)并且数据源来自网络(source)。

注意 在定义 file_system_provider_capabilities 时,需要注意以下几点。

(1) configurable:可选,表示是否支持通过 onConfigureRequested 进行配置,默认值为 false。

(2) watchable:可选,表示是否支持设置观察者和通知更改,默认值为 false。如果可能,则应该添加对观察者的支持,以便文件系统上的更改可以立即自动反映出来。

(3) multiple_mounts:可选,表示是否支持多个(超过一个)挂载的文件系统,默认值为 false。

(4) source:必须,表示挂载的文件系统的数据源。可以是 file、device 或 network。

22. homepage_url

homepage_url 是一个可选的 Manifest 键,包含一个有效的主页 URL 的字符串。开发者可以选择将扩展的主页设置为个人或公司的网站。如果此参数未定义,则默认的主页将是扩展的 Chrome Web Store 页面,该页面列在扩展管理页面(chrome://extensions)上,代码如下:

```
{
    …
    "homepage_url": "https://example.com",
    …
}
```

在这个示例中,将扩展的主页设置为 https://example.com。

注意 homepage_url 是一个可选的属性,如果没有定义它,则扩展的主页将默认为扩展的 Chrome Web Store 页面。此外,如果在自己的网站上托管扩展,则这个字段会特别有用。

23. host_permissions

host_permissions 属性用于请求扩展中需要读取或修改主机数据的 API 的访问权限，例如 cookies、webRequest 和 tabs。这个键是一个字符串数组，每个字符串都是一个权限请求。这些权限允许扩展与匹配模式的 URL 进行交互，代码如下：

```
{
    …
    "host_permissions": [
        "https://example.org/foo/bar.html"
    ],
    …
}
```

在这个示例中，将扩展的主机权限设置为"https://example.org/foo/bar.html"。

注意 大多数浏览器将 host_permissions 视为可选的。如果使用这个键请求权限，则用户可能会在安装过程中被提示授予这些权限。扩展可以在安装后立即使用 permissions.contains 检查是否具有所有必需的权限。如果没有必要的权限，则可以使用 permissions.request 请求它们。在请求授予主机权限之前，提供一个解释为什么需要某些权限的入门步骤可能会有所帮助。由于请求授予主机权限可能会影响用户愿意安装扩展，因此请求主机权限值得仔细考虑。

24. import

在 Manifest V3 中，import 属性并不存在，然而，可以在服务工作线程中使用 importScripts 函数来导入其他脚本。此外，也可以将 background 对象的 type 属性设置为 module，这样就可以在服务工作线程中使用 import 关键字来导入其他模块，代码如下：

```
{
    …
    "background": {
        "service_worker": "background.js",
        "type": "module"
    },
    …
}
```

在 background.js 文件中，可以使用 import 关键字来导入其他模块：

```
import { myFunction } from './myModule.js';

myFunction();
```

注意 在使用 import 关键字时，需要确保模块是 ES 模块，并且它们都遵循相同源策略。此外，需要注意 importScripts 函数和 import 关键字的区别。importScripts 函数是同步的，它会阻塞服务工作线程，直到脚本被完全加载和执行，而 import 关键字则是异步的，它不会阻塞服务工作线程。

25. incognito

incognito 属性用于指定扩展在隐身模式下的行为。可以选择 spanning、split 或 not_allowed 来定义扩展在隐身模式下的行为。

(1) spanning：扩展将在一个共享的进程中运行。来自隐身标签的任何事件或消息都将被发送到这个共享的进程，带有一个隐身标志来指示它来自哪里。

(2) split：所有在隐身窗口中的页面将在它们自己的隐身进程中运行。如果扩展包含一个背景页面，则将在隐身进程中运行。

(3) not_allowed：扩展不能在隐身模式下启用。

代码如下：

```
{
    ...
    "incognito": "split",
    ...
}
```

在这个示例中，将扩展的隐身模式设置为 split。

注意 当选择 incognito 属性的值时，需要考虑扩展的需求。例如，如果扩展需要在隐身浏览器中加载一个标签，则使用 split 隐身行为。如果扩展需要登录到一个远程服务器，则使用 spanning 隐身行为。

26. input_components

input_components 是一个可选的 Manifest 键，用于启用 input.ime API(输入法编辑器)以供 ChromeOS 使用。这允许扩展处理按键，设置组合，并打开辅助窗口。开发者还必须在扩展的 permissions 数组中声明 input 权限。该键接受一个对象数组：name、id、language、layouts、input_view 和 options_page，代码如下：

```
//第 5 章,5.3.26

{
    ...
    "permissions": [
        "input"
    ],
```

```
    "input_components": [
        {
            "name": "ToUpperIME",
            "id": "ToUpperIME",
            "language": "en",
            "layouts": ["us::eng"]
        }
    ],
    ...
}
```

在这个示例中,定义了一个名为 ToUpperIME 的输入组件,它的语言是英语,布局是美国英语。

注意 在定义 input_components 时,需要注意以下几点。

(1) name:输入组件对象的名称,这是必需的。

(2) id:组件对象的 id,这是可选的。

(3) language:指定的语言或适用的语言列表,这是可选的,例如"en",["en", "pt"]。

(4) layouts:输入方法的列表,这是可选的。注意,ChromeOS 只支持每种输入方法的一个布局。如果指定了多个布局,则选择顺序是未定义的,因此,强烈建议扩展只为每种输入方法指定一个布局。对于键盘布局,xkb:前缀表示这是一个键盘布局扩展,例如["us::eng"]。

(5) input_view:指定一个扩展资源的字符串,这是可选的。

(6) options_page:指定一个扩展资源的字符串,这是可选的。如果未提供,则将使用默认的扩展选项页面。

27. key

key 属性用于在开发过程中保持扩展的唯一 ID。这个值有一些常见的用途,例如配置服务器只接受来自 Chrome 扩展源的请求,让其他扩展或网站可以向扩展发送消息,以及让一个网站可以访问扩展的 web_accessible_resources,代码如下:

```
{
    ...
    "manifest_version": 3,
    ...
    "key":
"ThisKeyIsGoingToBeVeryLong/go8GGC2u3UD9WI3MkmBgyiDPP2OreImEQhPvwpliioUMJmERZK3zPAx72z8MD
vGp7Fx7ZlzuZpL4yyp4zXBI + MUhFGoqEh32oYnm4qkS4JpjWva5Ktn4YpAWxd4pSCVs8I4MZms20 + yx5OlnlmWQ
EwQiiIwPPwG1e1jRw0Ak5duPpE3uysVGZXkGhC5FyOFM + oVXwc1kMqrrKnQiMJ3lgh59LjkX4z1cDNX3MomyUMJ +
I + DaWC2VdHggB74BNANSd + zkPQeNKg3o7FetlDJya1bk8ofdNBARxHFMBtMXu/ONfCT3Q2kCY9gZDRktmNRiHG/
1cXhkIcN1RWrbsCkwIDAQAB",
    ...
}
```

在这个示例中，将扩展的 key 设置为一个长字符串。

注意 在使用 key 属性时，需要注意以下几点：

(1) 需要将扩展打包成.zip 文件并上传到 Chrome 开发者仪表板。

(2) 在开发者仪表板中，单击添加新项目，然后选择并上传扩展的.zip 文件。

(3) 转到包裹选项卡，单击查看公钥。

(4) 当弹出窗口打开时，复制-----BEGIN PUBLIC KEY-----到-----END PUBLIC KEY-----之间的代码。

(5) 删除换行符，使其成为一行文本。

(6) 将代码添加到 manifest.json 的 key 字段下。这样，扩展将使用相同的 ID。

28. minimum_chrome_version

minimum_chrome_version 属性用于定义扩展所需的 Chrome 的最低版本。这个值是一个字符串，必须是现有的 Chrome 浏览器版本字符串的子字符串。可以使用完整的版本号来指定 Chrome 的特定更新，或者可以使用字符串中的第 1 个数字来指定特定的主要版本，代码如下：

```
{
    ...
    "minimum_chrome_version": "107.0.5304.87",
    ...
}
```

在这个示例中，将扩展所需的 Chrome 的最低版本设置为"107.0.5304.87"。这也可以缩写为"107"、"107.0"或"107.0.5304"。

注意 在使用 minimum_chrome_version 属性时，需要注意以下几点：

(1) 在 Chrome 版本比最低版本老的情况下，Chrome Web Store 将在安装按钮的位置显示一个不兼容的消息。在这些版本上的用户将无法安装此扩展。

(2) 当 minimum_chrome_version 高于当前的浏览器版本时，扩展的现有用户将不会收到更新。由于这是无声的，所以应该小心并考虑让现有的用户知道他们不再接收更新的方式。

29. oauth2

oauth2 是一个可选的 Manifest 键，用于在扩展上启用 OAuth 2.0 安全 ID。这个键接受一个对象，该对象有两个必需的子属性：client_id 和 scopes。当开发一个使用 oauth2 键的扩展时，也应该考虑设置扩展的 key 以保持一致的扩展 ID，代码如下：

```
//第 5 章,5.3.29
{
    ...
```

```
    "oauth2": {
        "client_id": "YOUR_EXTENSION_OAUTH_CLIENT_ID.apps.googleusercontent.com",
        "scopes": ["https://www.googleapis.com/auth/contacts.readonly"]
    },
    "key": "EXTENSION_PUBLIC_KEY",
    ...
}
```

在这个示例中,将扩展的 oauth2 设置为一个对象,该对象包含 client_id 和 scopes。

注意 在使用 oauth2 属性时,需要注意以下几点:

(1) 当开发一个使用 oauth2 键的扩展时,应该考虑设置扩展的 key 以保持一致的扩展 ID。

(2) 应该使用 Chrome 的 identity API 来处理 OAuth2 授权。

(3) 需要在 Google Cloud Platform 上创建 OAuth2 客户端 ID 和令牌。

30. omnibox

omnibox 属性允许扩展在谷歌 Chrome 的网址栏(也称为 omnibox)中注册一个关键字。当用户输入扩展的关键字时,用户开始与扩展进行交互。每个按键都会发送到扩展,可以提供相应的建议。当用户接受一个建议时,扩展会被通知并可以采取行动,代码如下:

```
{
    ...
    "name": "My Extension",
    "version": "1.0",
    "omnibox": {
        "keyword" : "myExtension"
    },
    "icons": {
        "16": "icon.png"
    },
    ...
}
```

在这个示例中,将扩展的 omnibox 设置为一个对象,该对象包含 keyword。当用户在网址栏中输入 myExtension 时,用户将开始与扩展进行交互。

注意 在使用 omnibox 属性时,需要注意以下几点:

(1) 必须在 Manifest 中声明一个 omnibox 关键字字段,以此来使用 omnibox API。

(2) 应该指定一个 16×16 像素的图标,当建议用户输入关键字模式时,它将在网址栏中显示。

31. optional_host_permissions

optional_host_permissions 属性用于在运行时请求主机权限，而不是在安装时。这个键是一个字符串数组，每个字符串都是一个权限请求。这些权限允许扩展与匹配模式的 URL 进行交互，代码如下：

```
{
    ...
    "optional_host_permissions": [
        "https://*/*",
        "http://*/*"
    ],
    ...
}
```

在这个示例中，将扩展的 optional_host_permissions 设置为一个数组，该数组包含两个字符串 https://和 http://。

注意 在使用 optional_host_permissions 属性时，需要注意以下几点：

(1) 必须在 Manifest 中声明一个 optional_host_permissions 关键字字段，以此来使用这个属性。

(2) 应该使用 Chrome 的 permissions API 来处理权限请求。

(3) 当请求权限时，可能会向用户显示一个对话框，请求他们向扩展授予权限。为了最大限度地增加用户授予权限的可能性，应该考虑在请求运行权限时提供一个解释。

---

32. optional_permissions

optional_permissions 属性用于在运行时请求 API 权限，而不是在安装时。这个键是一个字符串数组，每个字符串都是一个权限请求。这些权限允许扩展使用特定的 API，代码如下：

```
{
    ...
    "optional_permissions": [
        "tabs",
        "bookmarks"
    ],
    ...
}
```

在这个示例中，将扩展的 optional_permissions 设置为一个数组，该数组包含两个字符串 tabs 和 bookmarks。

注意 在使用 optional_permissions 属性时，需要注意以下几点：

(1) 必须在 Manifest 中声明一个 optional_permissions 关键字字段，以此来使用这个属性。

(2) 应该使用 Chrome 的 permissions API 来处理权限请求。

(3) 当请求权限时,可能会向用户显示一个对话框,请求他们向扩展授予权限。为了最大限度地增加用户授予权限的可能性,应该考虑在请求运行权限时提供一个解释。

33. options_page

options_page 属性用于指定 options 页面文件。扩展程序可以使用 options_page 来提供更多、更详细的交互功能,例如配置扩展程序可以在哪些网站上运行,代码如下:

```
{
    "options_page": "options.html"
}
```

在这个示例中,当用户打开扩展程序的选项页面时,浏览器将会显示 options.html 这个 HTML 文件的内容。

注意 如果扩展程序需要用户进行一些配置,例如选择在哪些网站上运行,则需要使用 options_page 属性来指定一个选项页面。如果没有指定 options_page,则用户将无法对扩展程序进行配置。

34. options_ui

options_ui 属性用于指定选项页面的用户界面。与 options_page 不同的是,options_ui 会弹出一个小窗口来展示选项页面,代码如下:

```
{
    "options_ui": {
        "page": "options.html",
        "open_in_tab": false
    }
}
```

在这个示例中,当用户打开扩展程序的选项页面时,浏览器会弹出一个小窗口,窗口的内容来自 options.html 这个 HTML 文件。

注意 如果扩展程序需要用户进行一些配置,例如选择在哪些网站上运行,则需要使用 options_ui 属性来指定一个选项页面。如果没有指定 options_ui,则用户将无法对扩展程序进行配置。

35. permissions

permissions 字段用于声明插件需要访问的特定权限,以确保用户数据和隐私的安全。这些权限可以包括访问浏览器的某些 API,例如自定义创建右键菜单 API(contextMenus),

tab选项卡API(tabs),缓存API(storage),监听浏览器请求API(webRequest)等,代码如下:

```
//第5章,5.3.35
{
    "permissions": [
        "contextMenus",
        "tabs",
        "storage",
        "webRequest"
    ]
}
```

在这个示例中,声明了插件需要访问的权限,包括contextMenus、tabs、storage、webRequest。

注意 在开发插件时,需要根据插件的功能需求来声明所需的权限。如果没有声明某个权限,则插件将无法使用对应的API。同时,为了用户的数据和隐私安全,应该尽量减少所需的权限,只声明必要的权限。

36. requirements

requirements属性用于指定插件运行所需的硬件或Chrome浏览器的特定功能。例如,如果插件需要使用三维图形,则可以在requirements中指定"3D",代码如下:

```
{
    "requirements": {
        "3D": true,
        "plugins": true
    }
}
```

在这个示例中,声明了插件需要使用三维图形和插件功能。

注意 在开发插件时,需要根据插件的功能需求来声明所需的硬件或Chrome浏览器的特定功能。如果没有声明某个需求,则插件将无法使用对应的硬件或功能。

37. sandbox

sandbox属性用于指定一个或多个HTML页面,这些页面将在沙箱环境中运行,与扩展程序的其他部分隔离。这样可以提高扩展程序的安全性,防止恶意代码被执行,代码如下:

```
{
    "sandbox": {
```

```
        "pages": ["sandbox.html"]
    }
}
```

在这个示例中,声明了 sandbox.html 这个 HTML 文件将在沙箱环境中运行。

注意 在开发插件时,如果插件需要执行一些可能存在安全风险的操作,则需要使用 sandbox 属性来指定一个沙箱环境。如果没有指定 sandbox,则所有的 HTML 页面都将在非沙箱环境中运行,这可能会增加安全风险。

38. short_name

short_name 属性用于指定插件的简短名称。这个名称是一个字符串,可以包含任何想要的内容,例如“我的插件”等。这个名称将显示在 Chrome 浏览器的插件管理页面,帮助用户了解插件的信息,代码如下:

```
{
"short_name": "我的插件"
}
```

在这个示例中,将插件的简短名称声明为“我的插件”。

注意 在使用 short_name 属性时,需要注意以下几点:

(1) short_name 属性的值不会影响插件的功能,只是用于显示给用户看的。

(2) 如果没有指定 short_name,则 Chrome 浏览器将使用 name 属性的值作为简短名称。

39. side_panel

side_panel 属性用于指定一个 HTML 页面,这个页面将作为侧边栏显示在浏览器窗口的侧边。这样可以提供一个便捷的用户界面,用户可以在浏览网页的同时查看和操作插件的功能,代码如下:

```
//第 5 章,5.3.39
{
    "action": {
        "default_popup": "popup.html",
        "default_icon": {
            "16": "images/icon16.png",
            "48": "images/icon48.png",
            "128": "images/icon128.png"
        }
    },
    "action": {
```

```
        "default_popup": "popup.html",
        "default_icon": {
            "16": "images/icon16.png",
            "48": "images/icon48.png",
            "128": "images/icon128.png"
        }
    },
    "background": {
        "service_worker": "background.js"
    },
    "permissions": ["declarativeContent"],
    "action": {},
    "icons": {
        "16": "images/icon16.png",
        "48": "images/icon48.png",
        "128": "images/icon128.png"
    },
    "manifest_version": 3,
    "name": "My Extension",
    "version": "1.0",
    "side_panel": {
        "open_at_install": true,
        "page": "sidepanel.html"
    }
}
```

在这个示例中，声明了 sidepanel.html 这个 HTML 文件将作为侧边栏显示在浏览器窗口的侧边。

注意 在开发插件时，如果插件需要提供一个便捷的用户界面，则需要使用 side_panel 属性来指定一个侧边栏。如果没有指定 side_panel，则插件将无法显示侧边栏。

40. storage

storage 属性用于声明插件需要使用 chrome.storage API 来存储和检索数据。这个 API 允许插件在用户的浏览器中存储数据，这些数据可以在插件的不同部分之间共享，代码如下：

```
//第 5 章,5.3.40

//存储数据
chrome.storage.sync.set({key: 'value'}, function() {
  console.log('Value is set to ' + value);
});

//读取数据
chrome.storage.sync.get(['key'], function(result) {
  console.log('Value currently is ' + result.key);
});
```

在这个示例中,首先使用 chrome.storage.sync.set 方法存储了一个键-值对,然后使用 chrome.storage.sync.get 方法读取了这个键-值对的值。

注意 在使用 storage 属性时,需要注意以下几点。

(1) storage 属性有两种存储方式:sync 和 local。sync 存储方式会将数据同步到用户的谷歌账户,这样用户在不同的设备上使用插件时,可以共享这些数据。local 存储方式则只在本地存储数据。

(2) storage 属性有存储限制。对于 sync 存储方式,每个插件最多可以存储 100KB 数据;对于 local 存储方式,每个插件最多可以存储 5MB 数据。

(3) 在使用 storage 属性时,需要在清单文件的 permissions 字段中声明 storage 权限。

41. tts_engine

tts_engine 属性用于声明插件需要使用 chrome.ttsEngine API 来实现从文本到语音的转换。这个 API 允许插件自定义文本到语音的转换过程,例如改变语音的速度、音调等,代码如下:

```
//第 5 章,5.3.41
//注册一个语音合成器
chrome.ttsEngine.onSpeak.addListener(function(utterance, options, sendTtsEvent) {
  console.log('Received speech request: ' + utterance);
  sendTtsEvent({'type': 'start', 'charIndex': 0});
  sendTtsEvent({'type': 'end', 'charIndex': utterance.length});
});

//设置默认语音合成器
chrome.ttsEngine.setDefaultVoice('myVoice');
```

在这个示例中,首先使用 chrome.ttsEngine.onSpeak.addListener 方法注册一个语音合成器,然后使用 chrome.ttsEngine.setDefaultVoice 方法设置了默认的语音合成器。

注意 在使用 tts_engine 属性时,需要注意以下几点:

(1) tts_engine 属性需要在清单文件的 permissions 字段中声明 tts 和 ttsEngine 权限。

(2) 在使用 chrome.ttsEngine.onSpeak.addListener 方法注册语音合成器时,需要处理各种类型的 TTS 事件,例如 start、end、error 等。

(3) 在使用 chrome.ttsEngine.setDefaultVoice 方法设置默认语音合成器时,需要确保语音合成器已经被注册。

42. update_url

update_url 属性用于指定插件的更新服务器的 URL。当插件需要更新时,Chrome 浏览器会访问这个 URL 来获取更新信息,代码如下:

```
//第 5 章,5.3.42

{
    "update_url": "https://clients2.google.com/service/update2/crx"
}
```

在这个示例中,将插件的更新服务器的 URL 声明为 https://clients2.google.com/service/update2/crx。

注意 在使用 update_url 属性时,需要注意以下几点:

(1) update_url 属性只在发布到 Chrome Web Store 之外的插件中使用。如果插件是被发布到 Chrome Web Store 的,则不需要指定 update_url,因为 Chrome Web Store 会自动处理插件的更新。

(2) update_url 属性的值必须是 HTTPS 的 URL。

---

43. version_name

version_name 属性用于指定插件的版本名称。这个名称是一个字符串,可以包含任何想要的内容,例如 v1.0.0、beta 等。这个名称将显示在 Chrome 浏览器的插件管理页面,帮助用户了解插件的版本信息,代码如下:

```
{
    "version_name": "v1.0.0"
}
```

在这个示例中,将插件的版本名称声明为 v1.0.0。

注意 在使用 version_name 属性时,需要注意以下几点:

(1) version_name 属性的值不会影响插件的功能,只是用于显示给用户看的。

(2) 如果没有指定 version_name,则 Chrome 浏览器将使用 version 属性的值作为版本名称。

---

44. web_accessible_resources

web_accessible_resources 属性用于指定插件中可以通过网络访问的资源。这些资源可以包括图片、脚本、HTML 文件等。可以指定允许访问资源的页面,以此来限制资源的访问权限,代码如下:

```
//第 5 章,5.3.44
{
    "web_accessible_resources": [
        {
            "resources": ["*/img/xxx.png", "*/img/xxx2.png"],
```

```
            "matches": ["https://*.csdn.net/*", "https://*.xxx.com/*"]
        }
    ]
}
```

在这个示例中,声明了插件中的/img/xxx.png 和/img/xxx2.png 这两个资源可以被 https://.csdn.net/和 https://.xxx.com/这两个页面访问。

注意 在使用 web_accessible_resources 属性时,需要注意以下几点:

(1) web_accessible_resources 属性的值必须是一个数组,数组中的每个元素都是一个对象,对象中包含 resources 和 matches 两个字段。

(2) resources 字段用于指定可以通过网络访问的资源,matches 字段用于指定允许访问资源的页面。

(3) 在 Manifest V3 中,web_accessible_resources 属性的使用方式有所变化,需要提供数组对象,每个对象可以将一组资源映射到一组 URL 或扩展 ID。

5.4 本章小结

本章深度剖析了 Manifest V3 文件,这是构建 Chrome 扩展的基石。清单文件作为扩展的心脏,定义了扩展的基本信息和行为。详细解读了 manifest.json 文件的结构,它如何搭建起扩展的骨架,并通过一系列示例展示了其在实际开发中的应用。

首先,介绍了清单文件的基本组成和格式要求,解释了它如何指导浏览器加载和管理扩展。通过不同的例子演示了如何设置扩展的名称、版本、图标、权限等,并演示了如何通过声明 Background Scripts、Content Scripts、Popup Pages 和 Options Pages 来增强扩展的功能。

5.2 节探讨了浏览器插件的国际化问题,说明了如何通过_manifest.json_文件支持多语言,使扩展可以跨文化界限,触及全球用户。同时,还讲解了模式匹配(Pattern Matching),这是一个核心概念,用于定义扩展何时及在哪些网站上激活其功能。

5.3 节详细介绍了清单文件中的必填属性和推荐属性,以及编写时的注意事项。必填属性(如 manifest_version、name 和 version)是每个扩展都必须声明的,而推荐属性(如 default_locale、icons、permissions 等)则用于扩展更多的功能和提升用户体验。还提醒开发者注意属性使用中的潜在陷阱,例如权限过多可能会导致用户安全顾虑,以及如何避免这些问题。

通过本章的学习,开发者应能够熟练掌握清单文件的编写和使用,这为开发高质量的 Chrome 扩展奠定了坚实的基础。正确地使用清单文件不仅能确保扩展的顺利运行,也能够提升用户对扩展的信任和满意度。