24min

CHAPTER 3

第 3 章 Manifest 新特性介绍

3.1 浏览器插件的发展愿景

插件平台愿景是对 Chrome 插件平台未来发展方向的设想。它包括了对扩展功能、安全性、隐私性、性能和用户体验等方面的期望和目标。扩展平台愿景的提出是为了指导扩展平台的发展，并帮助开发者理解和接受扩展平台的未来方向。通过明确的愿景，可以使扩展平台的发展更加符合用户需求，同时也能够鼓励开发者创新和优化他们的扩展。扩展平台愿景主要体现在以下几方面。

(1) 隐私：提供了使扩展能够良好运行的方法，而无须持续访问用户数据。通过通知用户扩展正在做什么，让他们在运行时和上下文中授予权限，改善用户对权限的控制。

(2) 安全性：对扩展访问扩展上下文之外的资源采取更严格的协议和要求。

(3) 性能：确保扩展程序在所有设备上都能正常运行，这意味着：性能问题不会影响浏览器体验，即使安装了许多扩展程序，Chrome 也能顺利运行。

(4) 提高用户可见性和控制力：扩展平台将提供更大的用户可见性和控制力，以便用户可以更轻松地管理扩展如何访问其数据和其他资源。该平台已经开始通过以下方式解决这个问题：让用户修改授予扩展的主机权限。扩展菜单显示哪些项目可以或想要访问当前页面。将继续改善这一用户体验。寻找越来越重视临时的、上下文风格的权限授予，限制对用户数据的被动访问。ActiveTab 的引入是朝这个方向迈出的第 1 步。用户就如何处理其数据做出明智的决定也很重要。将引入新的方法来帮助用户了解每个扩展程序访问哪些数据及它如何使用这些数据，以便用户可以控制自己的数据。

3.1.1 Webby 模型

在最早的 Chrome 插件模型中，插件主要通过 C/C++这样的编译语言来扩展浏览器的功能，这种模型主要用于扩展浏览器本身的功能，而不是增强网页的功能。例如，Flash 插件就是一个典型的例子，它是通过 C/C++编写的，用来在浏览器中播放 Flash 动画。

随着 Web 技术的发展，Chrome 开始引入 Webby 模型，让插件的开发和使用都基于 Web 技术，这使 Web 开发者能够快速上手插件的开发，并且插件能够利用 Web 技术来增强浏览器的功能。

在 Webby 模型中，插件运行在一个隔离的沙盒环境中，这意味着插件的代码不能直接访问用户的本地文件系统或其他插件的代码，这样可以提高插件的安全性。插件可以通过 Chrome 提供的 API 来修改浏览器的行为和访问 Web 内容。

Webby 模型主要有以下优点。

(1) 快速上手：插件的开发和使用都基于 Web 技术，例如 HTML、JavaScript 和 CSS，这使 Web 开发者能够快速上手插件的开发。

(2) 安全性：插件运行在一个隔离的沙盒环境中，这意味着插件的代码不能直接访问用户的本地文件系统或其他插件的代码，这样可以提高插件的安全性。

(3) 灵活性：插件可以通过 Chrome 提供的 API 来修改浏览器的行为和访问 Web 内容，这给插件提供了很大的灵活性。

Webby 模型的主要缺点如下。

(1) 安全问题：随着越来越多的应用发布，有部分扩展程序会利用这个平台来非法获得对用户数据和元数据，这对用户的隐私和安全构成了威胁。

(2) 性能问题：插件可能会影响浏览器的性能，特别是如果插件的代码写得不好，或者插件使用了大量的资源，则可能会导致浏览器运行缓慢。

(3) 兼容性问题：由于插件的代码需要与多种浏览器版本兼容，因此插件的开发者需要花费大量的时间和精力来处理兼容性问题。

随着 Chrome 插件技术的不断发展，也在着重从隐私、安全、性能等多个角度解决 Webby 模型使用中的缺陷。

3.1.2 权限模型

权限模型是 Chrome 扩展平台用来控制扩展可以访问和修改的信息和资源的一种机制。在 Manifest V3 中，插件只需声明其需要的高级权限，如文件系统和网络服务。权限模型的出现是为了提高扩展的安全性和隐私性。通过权限模型，用户可以更精细地控制他们安装的任何扩展程序被允许访问哪些信息和资源。这样，用户就可以更好地保护自己的数据，同时也能够防止恶意扩展滥用权限。权限模型可以防止恶意扩展滥用权限，从而提高了扩展的安全性。通过权限模型，用户可以更好地控制他们的数据，从而提高了扩展的隐私性。浏览器插件的权限模型主要包括以下几部分。

1. 权限声明

插件需要在扩展的清单文件中声明其使用的 API 权限。这些权限可以是 tabs、bookmarks、storage 等，也可以是特定的 URL 模式，如 http://www.blogger.com/，http://*.google.com/等。这些权限可以在安装插件时或运行时向用户请求。

2. 可选权限

插件可以声明一些可选的权限,这些权限在安装时并不会被请求,而是在运行时根据需要请求。这些权限需要在清单文件的 optional_permissions 字段中列出。可选权限的主要优点是,插件可以在不需要这些权限时运行,从而减少了插件的权限数量,提高了安全性。

3. 主机权限

主机权限允许插件与 URL 的匹配模式进行交互。有些 Chrome API 需要主机权限,除了它们自己的 API 权限外。主机权限需要在清单文件的 host_permissions 字段中列出。

4. 请求和检查权限

插件可以使用 chrome.permissions API 来请求和检查权限。例如,插件可以使用 chrome. permissions. request()方法来请求权限,使用 chrome.permissions. contains()方法来检查权限是否已经被授予。

5. 移除权限

插件可以使用 chrome. permissions. remove()方法来移除不再需要的权限。

下面是一个清单文件中权限部分的例子,代码如下:

```
{
"name": "Permissions Extension",
...
"permissions": [
   "activeTab",
   "contextMenus",
   "storage"
],
"optional_permissions": [
   "topSites",
],
"host_permissions": [
   "https://www.developer.chrome.com/*"
],
"optional_host_permissions":[
   "https://*/*",
   "http://*/*"
],
...
"manifest_version": 3
}
```

3.1.3 隐私

在 Manifest V3 中,隐私是指扩展如何处理和访问用户数据的方式。Manifest V3 引入了一些新的特性和功能,以提供更高的隐私保护。隐私的重视是为了保护用户的个人信息,

防止被恶意扩展滥用。通过改进隐私保护，用户可以更好地控制他们的数据，同时也能够提高扩展的安全性。提供了使扩展能够良好运行的方法，而无须持续访问用户数据。通过通知用户扩展正在做什么，让他们在运行时和上下文中授予权限，改善用户对权限的控制。Chrome Extensions v3 相比 v2 在隐私性方面的改进主要体现在以下几方面。

1. 消息传递与 Service Worker 交互

在 Manifest V3 中，不同的插件上下文只能通过消息传递与 Service Worker 进行交互。这意味着扩展不能直接获取后台页的引用，而需要改为通过消息传递的形式来与 Service Worker 进行交互，这样可以避免直接获取后台页的引用，从而提高了隐私性。

2. 禁用 eval

在 Manifest V3 中，已经没有任何办法使用 eval 了。这意味着扩展不能再使用 eval 函数来执行动态代码，这样可以避免执行未经检查的代码，从而提高了隐私性和安全性。

3. 禁用 localStorage

在 Manifest V3 的 background.js 文件中 localStorage 被完全禁止使用，这意味着扩展不能再使用 localStorage 来存储数据，这样可以避免扩展直接访问用户的本地存储，从而提高了隐私性。

4. 广告拦截

在 Manifest V3 中，谷歌使用 DeclarativeNetRequest 取代了 V2 中的广告拦截 API。这一变化使广告拦截器在 Manifest V3 下将不得不扮演一个旁观者的角色，而不是网络流量的看门人，这样可以避免扩展直接拦截用户的网络流量，从而提高了隐私性。

3.1.4 安全性

对安全性的重视是为了保护用户的个人信息，防止被恶意扩展滥用。通过改进安全保护，用户可以更好地控制他们的数据，同时也能够提高扩展的安全性。它的安全性主要体现在以下几方面。

1. 更安全的后台服务模型

Manifest V3 引入了一个新的后台服务工作线程模型，这个模型运行在一个与扩展分离的进程中，这样可以降低内存使用并提升性能。此外，服务工作线程对扩展的数据的访问受到限制，这有助于提升安全性。

2. 更精细的权限系统

Manifest V3 引入了一个新的权限系统，这个系统更加精细，限制了扩展对敏感用户数据（如浏览历史、书签和网络活动）的访问。扩展需要明确声明它们需要的权限，用户对他们与扩展共享的数据有更多的控制权。

在 Manifest V2 中，所有的主机权限在安装时都会被授予，因此没有必要有一个面板来与它们进行交互，因为唯一的方式来撤销权限是卸载扩展，而在 Manifest V3 中，所有

的主机权限都变成了可选的,这为用户提供了一种更精细的方式来授予权限:他们可以允许扩展在单击时运行,或者总是在给定的域名上运行,或者总是在所有网站上运行。这也意味着需要能够与没有浏览器动作的 Manifest V3 扩展进行交互,以授予它们相应的权限。

3. 限制网络请求的能力

Manifest V3 通过替换旧的 WebRequest API,引入了一个更加受限的 DeclarativeNetRequest API,这个 API 让扩展可以请求 Chrome 阻止网络请求,但是对规则的数量和效果有限制,这有助于提升安全性。

4. 禁止远程托管代码

在 Manifest V2 中允许浏览器插件执行远程的 JavaScript 代码或者用户提供的自定义 JavaScript 脚本,这个机制是有问题的,它为恶意脚本的执行提供了机制,恶意脚本可以通过该机制利用 WebRequest API 来窃取用户的敏感信息,因为它可以读取、修改甚至阻止网络请求,这可能导致个人隐私泄露和存在安全漏洞。

Manifest V3 禁止扩展使用远程托管的代码,这有助于 Chrome Web Store 的审查者更好地理解扩展带来的风险。

3.1.5 性能

浏览器插件对性能的重视是为了确保扩展程序在所有设备上都能正常运行,这意味着性能问题不会影响浏览器体验,即使安装了许多扩展程序,Chrome 也能顺利运行。

1. CPU 使用率

许多 Chrome 插件有能力在用户打开的每页都运行额外的代码,尽管它们只在必要时运行代码。例如,Evernote Web Clipper 在每页都会花费 368ms 的时间运行代码,如果在这段时间内尝试与页面进行交互,则响应会感觉有些滞后。如果安装了多个扩展,则可能会对用户体验产生显著影响。

Chrome 扩展 V3 通过优化代码和加载策略,减少了在用户访问页面时对 CPU 的使用。例如,Grammarly 在 V3 中只在用户聚焦在文本区域时加载完整的 Grammarly.js 文件,而在大多数网站上只加载 112KB 的 Grammarly-check.js 脚本。这样的优化可以显著地降低对 CPU 的使用,提升扩展的性能。

2. 页面渲染时间

CPU 活动可以导致页面卡住和变得无响应,同时也会增加电能消耗,但是,如果处理发生在页面的初始加载之后,则对用户体验的影响可能不会太大。例如,一些扩展(如 Loom 和 Ghostery)在页面开始渲染之前运行了大量的代码,而其他扩展(如 Clever、Lastpass 和 DuckDuckGo Privacy Essentials)在页面开始加载时就运行了代码,这会延迟用户首次能看到页面内容的时间。

Chrome扩展V3通过优化代码执行时机，提升了页面渲染时间。例如，某些扩展在页面开始渲染之前运行了大量的代码，而在Manifest V3中，这些扩展可以在页面开始加载时就运行代码，从而减少了用户首次能看到页面内容的时间。

3. 后台CPU使用效率

Chrome扩展不仅可以在访问的页面上运行代码，还可以在属于Chrome扩展的后台页面上运行代码。例如，某段代码可以包含阻止对某些域的请求的逻辑，即使在访问简单页面时，Avira Safe Shopping也会让CPU忙于工作超过2s。

Chrome扩展V3通过优化后台代码执行策略，提升了后台CPU的使用效率。例如，某些扩展在后台页面上运行了大量的代码，而在V3中，这些插件可以更有效地管理和执行后台代码，从而减少了CPU使用时间。

4. 减少内存消耗

Chrome插件可以增加每个被访问页面的内存使用量，以及扩展本身所使用的内存。这可能会影响性能，特别是在低规格设备上。例如，广告拦截器和隐私工具通常会存储大量网站信息，需要大量的内存来存储这些数据。

Chrome扩展V3通过优化内存管理和数据存储策略，减少了每个被访问页面的内存使用量，以及扩展本身所使用的内存。例如，广告拦截器和隐私工具在V3中可以更有效地存储和管理大量网站信息，从而减少内存消耗。

5. API的升级

在Manifest V2中，WebRequest API允许开发者在插件的生命周期的任意时间点执行网络请求，因此，在极端情况下使用时，WebRequest API的使用可能会影响浏览器性能，特别是当扩展程序同时拦截大量请求时会增加系统资源的消耗，导致浏览器变慢或卡顿，同时会引入额外的延迟，特别是在处理大量请求时，可能会影响用户的网络体验。

Manifest V3最有争议的变化是用DeclarativeNetRequest API取代了WebRequest API，允许设置模式匹配规则探测所请求流量并采取行动。DeclarativeNetRequest API是一个声明式的网络请求修改系统。它允许扩展程序提前将一组规则提交到浏览器，告诉浏览器在特定条件下如何处理网络请求，而不需要实时地请求拦截。这种方式避免了传统WebRequest API中每个请求都需要获得扩展程序的处理情况，大大减轻了负担，但谷歌对规则的数量设置了上限。

3.1.6 Webbiness

Webbiness是一种基于Web技术的设计模式。它是Chrome扩展平台的基础，旨在使Web开发者能够快速上手。

Webbiness的出现是为了降低开发人员的参与难度，使他们能够更容易地开发和优化他们的扩展。此外，Webbiness还注重安全性，它在设计上比以前的模型更加安全。

3.2 主要新特性详解

3.2.1 Service Worker

Manifest V3 以服务工作线程(Service Worker)取代了后台页面(Background Page),这也意味着 Manifest V3 抛弃了持久性脚本和事件页面的二元性。服务工作者与对应的网页一样,扩展服务工作者也会监听和响应事件,以提升用户体验。对于网页服务工作线程来讲,这通常意味着管理缓存、预加载资源和启用离线网页。虽然服务工作线程仍然可以完成所有这些工作,但扩展包已经包含了大量可以离线访问的资源,因此,扩展服务工作线程往往专注于对扩展 API 公开的浏览器事件做出反应,但是服务工作线程与 Manifest V2 的 Background Page 还是不一样,主要区别有以下几点。

1. DOM 的访问

在 Manifest V2 中,Background Page 可以直接访问和操作 DOM,然而,在 Manifest V3 中,服务工作线程无法直接访问或操作 DOM。这意味着,如果你的扩展需要访问 DOM,则可能需要将这些调用移动到一个不同的 API 或者一个离屏文档。

2. XMLHttpRequest

在 Manifest V2 中,Background Pages 可以使用 XMLHttpRequest 来发送 HTTP 请求,但是,在 Manifest V3 中,服务工作线程不再支持 XMLHttpRequest,因此,需要将所有的 XMLHttpRequest 调用替换为 fetch()调用。

3. TimeAPI

在 Manifest V2 中,Background Pages 可以使用全局变量和定时器来管理状态和时间,然而,在 Manifest V3 中,服务工作线程在不使用时会被终止。这意味着,不能再依赖全局变量来维护应用状态,而需要将应用状态持久化。此外,终止服务工作线程也可能会在完成之前结束定时器,因此,需要用 alarms 来替换它们,代码如下:

```
//这行代码创建了一个新的警报。delayInMinutes: 5 参数指定了警报应在 5min 后触发
chrome.alarms.create({ delayInMinutes: 5 });
//添加了一个事件监听器,当警报触发时,它会执行指定的函数
chrome.alarms.onAlarm.addListener(() => console.log("5 minutes is up!"));
```

4. 事件处理器

扩展服务工作线程不仅是网络代理(正如网络服务工作线程通常被描述的那样)。除了标准的服务工作线程事件外,它们还响应扩展事件,如导航到新页面、单击通知或关闭标签页。它们的注册和更新方式也与网络服务工作线程不同。

由于服务程序需要定期启动和停止,因此必须以特定的方式组织后台脚本,以确保行为

正确。在编写 Service Worker 时，应牢记以下行为。

（1）安装过程中触发的第 1 个事件是网络服务工作线程的安装事件。

（2）接下来是扩展的 OnInstalled 事件，它会在扩展（而非服务工作线程）首次安装、扩展更新到新版本及 Chrome 浏览器更新到新版本时触发。使用该事件可以设置状态或进行一次性初始化，例如上下文菜单，代码如下：

```
chrome.runtime.onInstalled.addListener((details) => {
  if(details.reason !== "install" && details.reason !== "update") return;
  chrome.contextMenus.create({
    "id": "sampleContextMenu",
    "title": "Sample Context Menu",
    "contexts": ["selection"]
  });
});
```

（3）服务工作线程的激活事件被触发。需要注意的是，与网络服务工作线程不同的是，该事件会在安装扩展后立即触发，因为在扩展中没有与页面重载类似的功能。

（4）通常情况下，Chrome 会在满足以下条件之一时终止 Service Worker：

①30s 不活跃，接收事件或调用扩展 API 会重置该计时器；②单个请求（如事件或 API 调用）的处理时间超过 5 分钟；③fetch()响应的到达时间超过 30 秒。

总之，为了优化扩展的资源消耗，应尽可能地避免让服务工作线程无限期地存活。应测试扩展，以确保不会无意中这样做。可以使用服务工作线程内部接口工具调试服务工作线程，如图 3-1 所示。

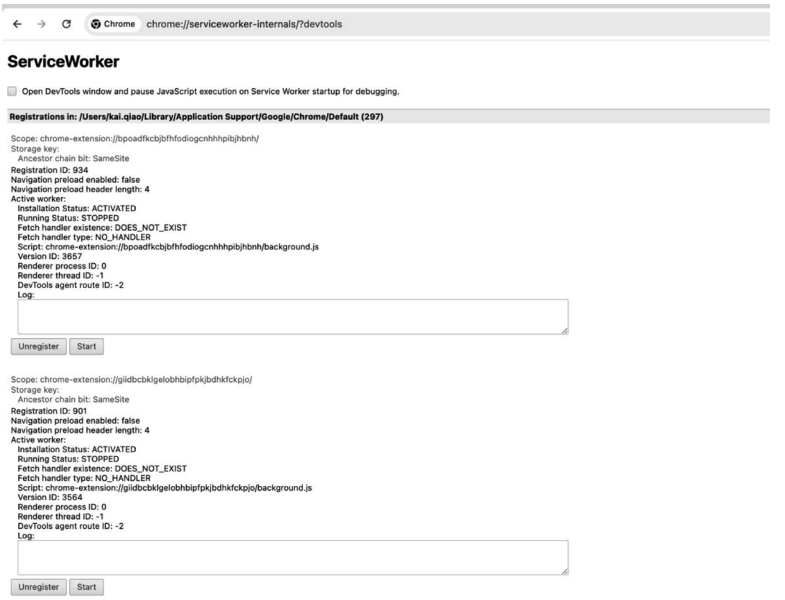

图 3-1　服务工作线程接口工具

注意 在谷歌浏览器中,如果访问 chrome://serviceworker-internals/?devtools,则会发现服务工作线程内部接口,它允许您监控服务工作线程的实时状态及控制台输出。浏览器检查器界面可能会阻止浏览器识别空闲的服务工作线程,而该工具则不同,它允许服务工作线程空闲并停止。

5. 持久化

在 Manifest V2 中,Background Pages 可以持续运行,即使没有任何活动窗口或标签页,然而,在 Manifest V3 中,服务工作线程在不使用时会被终止。这意味着,不能再依赖全局变量来维护应用状态,而需要将应用状态持久化,代码如下:

```
//在 background.js 文件中
chrome.runtime.onInstalled.addListener(() => {
  //使用 storage API 初始化一个值
  chrome.storage.local.set({key: 'value'}, () => {
    console.log('Value is set to ' + value);
  });
});

chrome.runtime.onMessage.addListener(
  (request, sender, sendResponse) => {
    if (request.message === 'changeValue') {
      //改变存储的值
      chrome.storage.local.set({key: request.newValue}, () => {
        console.log('Value is set to ' + request.newValue);
      });
    }
  }
);

//在 Content Script 或者 popup.js 文件中
chrome.runtime.sendMessage({message: 'changeValue', newValue: 'newValue'});
```

在这个示例中,首先在扩展安装时使用 chrome.storage.local.set 方法初始化一个值,然后监听来自 Content Scripts 或者 Popup 的消息,当收到'changeValue'消息时,就改变存储的值。

3.2.2 网络请求调整

Manifest V3 改变了扩展修改网络请求的方式。新的声明式网络请求 API(DeclarativeNetRequest API)可让扩展以保护隐私和高性能的方式修改和阻止网络请求。该 API 的精髓在于扩展程序不是拦截请求并按程序修改请求,而是要求 Chrome 浏览器代表它评估和修改请求。

扩展程序首先声明了一组规则，以便匹配请求的模式和匹配后要执行的操作，然后浏览器根据这些规则修改网络请求。使用这种声明式方法可以大大减少对持久主机权限的需求。

声明式权限有助于限制您的扩展受到恶意软件的威胁时的损害。某些权限警告会在安装前或运行时显示给用户，以获取他们的同意，如在带有警告的权限中进行详细说明。考虑在您的扩展功能允许的地方使用可选权限，以便为用户提供对资源和数据访问的知情控制，代码如下：

```
//manifest.json
{
  "name": "My extension",
  ...
  "permissions": [
    "declarativeNetRequest",
    "declarativeNetRequestFeedback"
  ],
  "host_permissions": [
    "http://*/*",
    "https://*/*"
  ],
  "declarative_net_request": {
    "rule_resources": [
      {
        "id": "ruleset_1",
        "path": "rules.json"
      }
    ]
  },
  ...
}
```

然后创建一个名为 rules.json 的文件，其中包含需要的规则集，代码如下：

```
//
[
  {
    "id": 1,
    "priority": 1,
    "action": {
      "type": "block"
    },
    "condition": {
      "urlFilter": "example.com",
      "resourceTypes": ["main_frame"],
      "domains": ["example.com"],
      "ExceludedDomains": ["subdomain.example.com"]
    }
  }
]
```

这个规则集包含一个规则,该规则阻止所有指向 example.com 的主框架请求,但不包括指向 subdomain.example.com 的请求,而使用 WebRequest 的代码如下:

```
chrome.webRequest.onBeforeRequest.addListener(
  function(details) {
    return {cancel: details.url.indexOf("example.com") != -1};
  },
  {urls: ["<all_urls>"], types: ["main_frame"]},
  ["blocking"]
);
```

这段代码的效果与上述 DeclarativeNetRequest API 的示例相同,但它使用了 WebRequest API 的 onBeforeRequest 事件监听器,并且需要 webRequestBlocking 权限。

总体来讲,DeclarativeNetRequest API 提供了一种声明式的方法来处理网络请求,而不是在运行时拦截和修改网络请求。这提供了更多的隐私,因为扩展代码无法直接访问请求的详细信息。

3.2.3 远程资源访问限制

Manifest V3 的一项关键安全改进是,扩展程序无法加载 JavaScript 或 Wasm 文件等远程代码。这样,当扩展提交到 Chrome 网上商城时,就能更可靠、更高效地审查扩展的安全行为。具体来讲,所有逻辑都必须包含在扩展包中,但是图片、音视频文件不受影响。

如果插件需要访问远程资源,则主要有以下方案。

(1) 使用服务工作线程:服务工作线程可以向任何网站发送 fetch 请求,只要该网站允许跨源资源共享(CORS)。可以在服务工作线程中发送请求,然后将结果传递给扩展。

(2) 使用 CORS 头:如果需要控制远程资源的服务器,则可以添加 CORS 头来允许扩展访问这些资源。

(3) 使用代理服务器:如果无法控制远程资源的服务器,则可以设置一个代理服务器,将请求从扩展转发到远程服务器,然后将响应返给扩展。

以下是一个使用服务工作线程发送 fetch 请求的示例,代码如下:

```
//在 background.js 文件中
chrome.runtime.onInstalled.addListener(() => {
  //注册服务工作线程
  navigator.serviceWorker.register('service-worker.js');
});
//在 service-worker.js 文件中
self.addEventListener('fetch', event => {
  if (event.request.url.includes('example.com')) {
    event.respondWith(fetch(event.request));
  }
});
```

在这个示例中，首先在安装扩展时注册了一个服务工作线程，然后在服务工作线程中监听 fetch 事件，当请求的 URL 包含'example.com'时，使用 fetch API 发送请求，并将结果作为响应返回。

3.2.4 Promise

Manifest V3 为 Promise 提供了对更多的支持。大多数应用程序接口现在支持 Promise，最终将在所有适当的方法上支持 Promise。Promise 是异步方法返回值的代理或占位符。使用 Promise 主要有以下几个优点。

(1) 更清晰的代码：使用 Promise 可以使代码更清晰，更易于维护。应该考虑在以下情况下使用 Promise：任何时候想通过更同步的调用风格来清理代码。

(2) 错误处理：在使用回调进行错误处理过于困难的情况下，应该考虑使用 Promise。

(3) 并发方法调用：当想要一种更紧凑的方式来调用几个并发方法并将结果收集到一个代码线程中时，应该考虑使用 Promise。

此外还支持 Promise Chains、Async 和 Await。某些 API 功能(如事件监听器)仍然需要回调，代码如下：

```
//使用 Promise
const newPerms = { permissions: ['topSites'] };
chrome.permissions.request(newPerms)
  .then((granted) => {
    if (granted) {
      console.log('granted');
    } else {
      console.log('not granted');
    }
  });
```

该示例是一个使用 Promise 的示例。在这个示例中，使用 chrome.permissions.request 方法请求权限，然后使用.then 方法处理返回的 Promise。如果权限被授予，则在控制台打印 granted，否则打印 not granted。与此相比，使用回调的代码如下：

```
//使用回调
chrome.permissions.request(newPerms, (granted) => {
  if (granted) {
    console.log('granted');
  } else {
    console.log('not granted');
  }
});
```

这段代码的效果与上述 Promise 的示例效果相同，但它使用了回调而不是 Promise。

注意 为了向后兼容,在添加 Promise 支持后,许多方法仍继续支持回调。需要注意的是,不能在同一个函数调用中同时使用这两种方法。如果传递回调,则函数将不会返回 Promise。如果希望返回 Promise,则不要传递回调。

3.3 本章小结

本章首先讨论了浏览器插件的发展愿景,例如在隐私、安全、性能、体验方面,然后讨论了 Chrome 扩展 Manifest V3 的一些关键特性和变化。首先介绍了 Manifest V3 的服务工作线程与 Manifest V2 的 Background Page 的区别,包括 DOM 访问、XMLHttpRequest、Time API,以及持久性、全局状态和存储。接着介绍了一些使用 Manifest V3 中的新 API,如 declarativeNetRequest 和 storage,然后介绍了 Manifest V3 对远程资源访问的限制,以及如何在有这种需求时进行处理。最后,介绍了为什么 Manifest V3 要引入 Promise,并提供了一个使用 Promise 的示例。

总之,Manifest V3 引入了一些重要的改变,这些改变旨在提高扩展的性能、安全性和隐私保护。虽然这可能需要开发者对现有的扩展进行一些修改,但最终的结果将是值得的。