Chrome 插件开发指南

基于 Manifest V3 的 Chrome 扩展开发中文指南——从 Hello World 到上架 Chrome Web Store

全 16 篇 · 作者 David

Chrome 插件开发指南

Chrome 扩展（Extension）是在浏览器里运行的小型程序：可以改网页、拦请求、加侧边栏、接快捷键，而不必 fork Chromium。本专栏面向有前端/JS 基础、没写过插件的开发者，全程使用 Manifest V3（MV3）——2024 年起新上架扩展的唯一标准。

> 本指南以 Chrome 128+、Chromium 系浏览器（Edge、Brave 等）为准。API 名称与官方 Chrome Extensions 文档 对齐。10 课实战完整源码已开源：github.com/davidapp/chrome-extension-demos，每个 Demo 目录均可直接在 `chrome://extensions` 加载验证。

学习路径

```
入门（扩展是什么、环境、第一个插件）
    ↓
Manifest V3（清单、Service Worker、Content Script、Popup）
    ↓
通信与存储（消息、chrome.storage、网络与 CORS）
    ↓
权限与安全（最小权限、CSP、host_permissions）
    ↓
进阶 API（Side Panel、Scripting、declarativeNetRequest）
    ↓
工程化（调试、测试、项目结构、上架商店）
```

章节目录

入门

2. 开发环境准备 — Chrome、加载方式、目录结构
3. Hello World — 最小可运行 MV3 扩展

Manifest V3

4. manifest.json 概览 — 字段、版本、action、icons
5. Service Worker — 后台脚本、生命周期、事件
6. Content Scripts — 注入页面、隔离世界、run_at
7. Popup 与选项页 — action popup、options_page、React 集成

通信与存储

8. 消息传递 — runtime.sendMessage、长连接、与页面通信
9. 存储 API — local / sync / session，配额与迁移
10. 网络与 CORS — fetch、host_permissions、跨域限制

权限与安全

11. 权限模型 — permissions、optional_permissions、最小权限
12. 内容安全策略 — extension_pages CSP、eval 禁令

进阶 API

13. Side Panel — 侧边栏 UI、与 Tab 联动
14. Scripting API — 动态注入、executeScript、registerContentScripts
15. declarativeNetRequest — 规则化拦截/重定向（替代 webRequest 阻塞）

工程化

16. 调试 — Service Worker 调试、Content Script 断点
17. 测试策略 — 单元测试、Puppeteer、CI
18. 项目结构 — 构建、TypeScript、WXT/Plasmo 选型
19. 上架 Chrome Web Store — 打包、开发者账号、审核要点

附录

20. PWA 开发指南 — Web Manifest、离线缓存、与扩展的选型边界
21. 油猴脚本开发指南 — Tampermonkey 元数据、GM API、迁移到扩展

适合谁

- 会写 HTML/CSS/JS 或 TypeScript，想给浏览器加能力的开发者
- 需要把现有 Web 能力「钉」在特定站点上的产品/工具作者
- 想理解 MV3 与 MV2 差异、避免审核踩坑的维护者

写作原则

1. 可运行优先——开源 Demo 仓库 提供 10 个完整可加载项目；专栏文章讲解原理与关键片段，完整 HTML/CSS/JS 以仓库为准。
2. MV3 唯一——不教已废弃的持久 background page、`webRequest` 阻塞模式。
3. 权限即设计——先问「最少需要哪些 host_permissions」，再写功能。

从 扩展是什么 开始，或已有基础直接上 实战第 1 课。

实战课程（配套开源仓库）

完整源码：davidapp/chrome-extension-demos（Chrome 116+，纯 HTML/CSS/JS，无需构建）。建议 clone 后按编号顺序学习：

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos
```

在 `chrome://extensions` 开启开发者模式 → 加载已解压的扩展程序 → 选择对应课次文件夹（如 `01-hello-popup`）。每课目录内的 README.md 含逐步操作与课后练习。

课	文章	源码目录	主题
01	Hello Popup	01-hello-popup	MV3、Popup、CSP
02	Content Script	02-content-script	隔离世界、`scripting`
03	Service Worker	03-service-worker	后台事件、Badge
04	消息传递	04-message-passing	划词翻译
05	Storage API	05-storage-api	Todo、`onChanged`
06	Options Page	06-options-page	选项页、sync
07	Context Menus	07-context-menus	右键菜单、Toast
08	Side Panel	08-side-panel	侧边栏笔记本
09	DNR	09-declarative-net-request	广告拦截规则
10	Offscreen	10-offscreen-document	后台 TTS

也可在 GitHub 上 Download ZIP 后解压使用。专栏文章与仓库 README 互补：文章讲「为什么」与架构，仓库给「完整能跑」的实现。

扩展是什么

扩展不是「在 Chrome 里打开的一个网站」，而是安装在浏览器配置文件里、跨标签页存活的程序包。理解它和网页、PWA、油猴脚本的区别，能避免把插件写成「带 `manifest.json` 的网页」。

直觉：四个角色

形态	谁运行它	典型能力	例子
普通网页	单个标签页	只能访问自己源、用户打开才存在	`https://example.com`
PWA	标签页 + 可安装	离线缓存、推送（受平台限制）	记事本 PWA（见 附录：PWA 开发指南）
Chrome 扩展	浏览器进程 + 可选注入页面	读改任意允许站点 DOM、拦网络、改工具栏	广告拦截、翻译、密码管理
用户脚本（Tampermonkey）	注入脚本	改 DOM，能力取决于管理器	站点美化脚本（见 附录：油猴脚本开发指南）

扩展的核心是 manifest 声明能力 + 多种 JS 运行上下文（Service Worker、Content Script、Popup 等），由 Chrome 按权限调度。

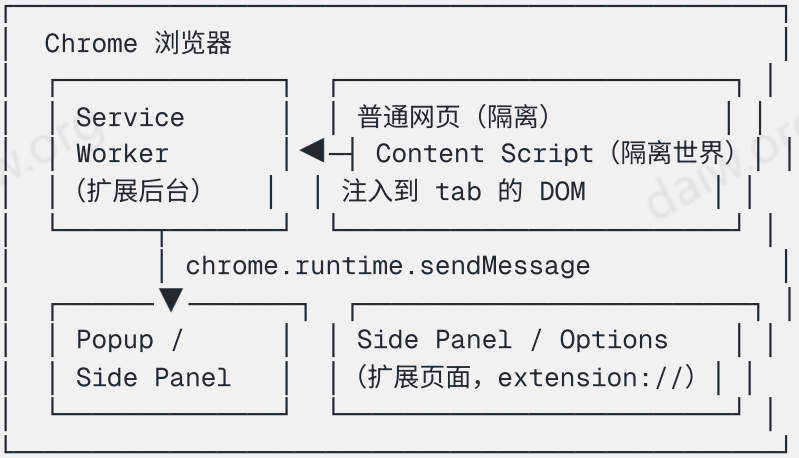
定义

Chrome 扩展（Extension） 是一个包含 `manifest.json` 的目录（或 zip 包），声明：

- 元数据：名称、版本、图标、描述
- 入口：`background` Service Worker、`action` 弹出页、`content_scripts` 注入规则等
- 权限：`permissions`、`host_permissions`、可选的 `optional_permissions`

用户从 `chrome://extensions` 加载未打包目录，或从 Chrome Web Store 安装后，扩展在浏览器会话间持久存在（直到禁用/卸载）。

MV3 下的三种常见上下文



上下文	能否直接访问网页 DOM	能否用 `chrome.*` API	生命周期
Service Worker	否	是（按 manifest 权限）	事件驱动，空闲可被回收
Content Script	是（隔离世界）	部分（如 `runtime.sendMessage`）	随匹配页面加载/卸载
Popup / Options	否（除非 iframe 嵌页面）	是	打开时存在，关闭即销毁

隔离世界（Isolated World）：Content Script 与页面自己的 JS 共享 DOM，但不共享 `window` 上的变量；两边要靠 `postMessage` 或注入 `<script>` 桥接（见后文 消息传递）。

例子 1：翻译插件在做什么

1. `content_scripts` 在 `*://*/*` 匹配页注入 `translate.js`，监听用户选中文本。
2. 用户点击扩展图标，Popup 显示设置。
3. Service Worker 收到消息，带 `host_permissions` 请求翻译 API，把结果 `sendMessage` 回 Content Script 显示浮层。

没有扩展权限，纯网页无法在读 `bank.com` 的同时读 `news.com` 的 DOM。

例子 2：与 MV2 的关键差异（只需记住三条）

点	**MV2**（旧）	**MV3**（本专栏）
后台	持久 `background.page`	事件型 Service Worker
拦截网络	`webRequest` 阻塞	declarativeNetRequest 声明式规则
远程代码	较松	禁止远程托管可执行代码；逻辑须打包进扩展

新提交 Chrome Web Store 必须 MV3。

量化：规模直觉

指标	典型量级
最小扩展文件	`manifest.json` + 1 个 JS，约 1 KB
商店热门扩展包体	1–5 MB（含图标、WASM、模型等可达数十 MB）
Service Worker 单次运行时限	无固定「5 分钟被杀」计时器，但空闲会休眠；长任务需 `chrome.alarms` 或拆片
`chrome.storage.local` 配额	约 10 MB（可申请 `unlimitedStorage`）

常见误区

1. 「扩展 = 内嵌网页」——Popup 是扩展页，但能力来自 manifest 声明的全局权限，不是 `<iframe>` 能替代的。
2. 「Content Script 能读页面 JS 变量」——默认不能；这是安全设计。
3. 「Service Worker 一直在后台跑」——MV3 下会被回收；需要持久状态请用 存储 API 或 `chrome.alarms`。

小结

Chrome 扩展是跨标签、带特权 API 的浏览器程序；MV3 用 Service Worker + 声明式网络规则，Content Script 负责「碰网页」，Popup/Side Panel 负责「碰用户」。下一篇准备开发环境，再动手写 Hello World。

→ 下一篇：开发环境准备

开发环境准备

写扩展不需要特殊 SDK：一个文本编辑器 + Chrome 即可。本篇说明最低环境、如何加载未打包扩展做热调试，以及团队项目常用的目录与构建工具选型。

> 10 课实战完整源码已开源：github.com/davidapp/chrome-extension-demos。clone 后每个课次目录均可直接在 `chrome://extensions` 加载验证。

直觉：像本地跑静态站，但入口是 manifest

网页开发用 `npm run dev` 起服务器；扩展开发是告诉 Chrome：这个文件夹是一个扩展，Chrome 按 `manifest.json` 注册 Service Worker 和注入规则。改代码后 mostly 在 `chrome://extensions` 点刷新。

环境要求

项	建议
浏览器	Chrome 128+ 或 Chromium 系（Edge 137+ 等）；开发时打开 `chrome://version` 确认
编辑器	VS Code / Cursor；推荐扩展 Chrome Extension Manifest File 做 JSON 校验
Node.js	可选；用 TypeScript、打包、ESLint 时需要 Node 22+（与本站一致）
账号	仅上架商店需要 $5 一次性 Chrome Web Store 开发者注册费

获取开源 Demo 仓库

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos
```

也可 Download ZIP 解压。仓库内 `01-hello-popup` … `10-offscreen-document` 十个目录与 实战课程 一一对应，每目录 README 含逐步操作说明。

加载未打包扩展

1. 打开 `chrome://extensions`
2. 右上角开启 开发者模式

3. 点击 加载已解压的扩展程序，选择含 `manifest.json` 的目录
4. 修改文件后，在该扩展卡片上点击 刷新（或勾选「重新加载扩展」相关选项）

```
my-extension/
├── manifest.json      # 必需
├── service-worker.js  # 示例：后台
├── popup.html
├── popup.js
└── icons/
    ├── icon16.png
    ├── icon48.png
    └── icon128.png
```

> ⚠ 加载目录后，不要删除或移动该文件夹；Chrome 记录的是路径。换机器请复制整个目录或改用 zip 打包加载。

推荐目录结构（中小型项目）

```
chrome-extension/
├── manifest.json
├── src/
│   ├── background/        # Service Worker
│   ├── content/           # Content Scripts
│   ├── popup/             # Popup React/Vanilla
│   └── lib/               # 共享工具
├── public/
│   └── icons/
├── package.json           # 若用构建工具
└── dist/                  # 构建产物（加载此目录）
```

原则：开发时加载 `dist/` 或根目录二选一，manifest 里引用的路径与加载目录一致。

工具链选型（可选）

工具	适合	说明
纯手写	教程、极小扩展	零依赖，本章 Hello World 即此路线
WXT	现代 TS 项目	类 Vite 体验，热重载、多浏览器目标
Plasmo	React 优先	框架感强，上手快
`esbuild` / `rollup` 自配	已有 monorepo	扩展作为子包，完全自控

本专栏示例以手写 MV3 为主，避免框架掩盖 manifest 细节；工程化篇再展开 项目结构。

调试入口速查

要调试的对象	入口
Service Worker	`chrome://extensions` → 扩展卡片 → Service Worker 链接
Popup	右键扩展图标 → 检查弹出内容
Content Script	普通网页 DevTools → Sources → Content scripts
扩展页（options 等）	右键页面 → 检查

常见误区

1. 改 manifest 却不点刷新——多数 manifest 变更必须重载扩展；仅改 Content Script 有时刷新页面即可。
2. 路径写错——`"service_worker": "src/bg.js"` 但加载的是 `dist/`，会报 Could not load background script。
3. 用 file:// 测 Content Script——部分匹配规则不对本地文件生效；用 `https://example.com` 或本地静态服务器。

动手试一试

1. Clone 开源 Demo 仓库（若仅跟 Hello World 手写，可跳过此步自建空目录）：

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos
```

2. 打开 `chrome://extensions`，开启开发者模式，确认能看到 加载已解压的扩展程序
3. 加载 `01-hello-popup` 体验第一课，或下一篇在空目录手写最小 manifest

小结

Chrome 128+、开发者模式、加载含 `manifest.json` 的目录，即构成最小开发环境。建议 clone 开源 Demo 仓库 按课次实战。记住 改代码 → 刷新扩展/页面 的循环。下一篇写出第一个可点击的 Hello World。

→ 下一篇：Hello World

Hello World

本篇从零写一个能加载、能点击、能在后台打日志的 MV3 扩展。完成后你应能在工具栏看到图标，点开 Popup 显示文字，并在 Service Worker 控制台看到启动记录。

> 本篇为最小可运行示例。带精美 UI 的完整 Popup 实战见 第 1 课：Hello Popup，源码在 01-hello-popup。

目标效果

- 工具栏显示扩展图标
- 点击图标弹出 Hello, Chrome Extension!
- Service Worker 启动时 `console.log('SW started')`

完整文件

在空目录创建以下 4 个文件（图标可先用任意 PNG，或暂时省略 `icons` 字段）。

manifest.json

```
{
  "manifest_version": 3,
  "name": "Hello MV3",
  "version": "1.0.0",
  "description": "最小 Chrome 扩展示例",
  "action": {
    "default_title": "Hello",
    "default_popup": "popup.html"
  },
  "background": {
    "service_worker": "service-worker.js"
  }
}
```

service-worker.js

```
console.log('SW started', new Date().toISOString());

chrome.runtime.onInstalled.addListener(({ reason }) => {
  console.log('onInstalled', reason);
});
```

popup.html

```html
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="UTF-8" />
    <style>
      body {
        margin: 16px;
        font: 14px system-ui, sans-serif;
        min-width: 200px;
      }
    </style>
  </head>
  <body>
    <p>Hello, Chrome Extension!</p>
  </body>
</html>
```

（Popup 无需单独 `popup.js` 即可显示静态文案。）

加载与验证

1. `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选该目录
2. 点击工具栏拼图图标，固定 Hello MV3
3. 点击图标，应出现 Popup 文案
4. 在扩展卡片点击 Service Worker → Inspect，Console 应有 `SW started` 与 `onInstalled install`

字段说明

字段	作用
`manifest_version: 3`	声明 MV3；缺省或写 2 将无法上架新包
`action`	替代 MV2 的 `browser_action`；`default_popup` 为点击图标弹出的 HTML
`background.service_worker`	后台单例脚本路径（相对扩展根）

加图标（推荐）

```
  "icons": {
    "16": "icons/icon16.png",
    "48": "icons/icon48.png",
    "128": "icons/icon128.png"
  },
```

商店与 `chrome://extensions` 列表会用到 128；工具栏常用 16/32（Chrome 会自动缩放）。

常见报错

错误	原因
`Service worker registration failed`	`service-worker.js` 路径错误或语法错误
`default_popup` 空白	`popup.html` 路径错或 HTML 解析失败
看不到图标	未 pin 到工具栏；或缺少 `action` / `icons`

延伸：点击 Popup 发消息给 Service Worker

在 `popup.html` 底部增加：

```
    <button id="ping">Ping SW</button>
    <script src="popup.js"></script>
```

popup.js

```
document.getElementById('ping').addEventListener('click', async () => {
  const res = await chrome.runtime.sendMessage({ type: 'PING' });
  console.log('reply', res);
});
```

service-worker.js 追加：

```
chrome.runtime.onMessage.addListener((msg, _sender, sendResponse) => {
  if (msg.type === 'PING') {
    sendResponse({ ok: true, ts: Date.now() });
  }
  return true;
});
```

右键 Popup → 检查，点按钮应在 Popup 控制台看到 `reply`。消息机制详见 消息传递。

小结

最小 MV3 扩展 = `manifest.json` + `background.service_worker` + `action.default_popup`。加载目录、刷新、查 Service Worker 控制台，构成日常开发三角。建议 clone 开源 Demo 仓库 进入实战课程，或继续系统学习 manifest 各字段。

→ 下一篇：第 1 课：Hello Popup · 参考：manifest.json 概览

第 1 课：Hello Popup

本课配套开源 Demo 01-hello-popup：创建第一个带精美 UI 的 Chrome 插件，理解 MV3 清单、Popup 生命周期与 CSP 约束。

▲ *在真实Chrome 中加载01-hello-popup 后，点击工具栏图标弹出的Popup*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `01-hello-popup` 目录内的 README。

本课目标

- 理解 Chrome 扩展 = Web 技术 + 浏览器特权 API
- 写出最小 `manifest.json`（`manifest_version: 3` + `action.default_popup`）
- 实现 Popup 内时钟、问候卡片、明暗主题切换
- 掌握 Popup 调试技巧（DevTools 保持窗口不关闭）

核心概念

Chrome 插件是什么

插件由 HTML、CSS、JavaScript 和图片组成，但可调用 `chrome.*` API（标签页、存储、脚本注入等），能力远超普通网页。

Manifest V3 三条变化

点	MV3
安全	禁止远程托管可执行 JS，代码必须打包进扩展
后台	用 Service Worker 替代常驻 Background Page（本课暂不涉及）
API	异步接口支持 Promise / async-await

Popup 生命周期

Popup 是点击工具栏图标弹出的临时窗口。点击外部或切换标签页会立即销毁——每次打开 HTML/JS 重新执行，内存变量清零。持久化请用 Storage API（第 5 课）。

文件结构

```
01-hello-popup/
├── manifest.json
├── popup.html
├── popup.css
└── popup.js
```

> ⚠ CSP 禁止内联脚本：不能写 `onclick="..."` 或 `<script>console.log(1)</script>`，必须用 `<script src="popup.js"></script>` 外部引入。

manifest.json

```
{
  "manifest_version": 3,
  "name": "Hello Chrome Extension",
  "version": "1.0.0",
  "description": "我的第一个 Chrome 插件 - 体验 Manifest V3 和 Popup",
  "action": {
    "default_popup": "popup.html"
  }
}
```

本课不声明 icons——Chrome 使用默认占位图。若声明 `icons`，对应 PNG 必须存在，否则加载失败。

popup.js 要点

```
document.addEventListener('DOMContentLoaded', () => {
  const timeEl = document.getElementById('current-time');
  const updateTime = () => {
    timeEl.textContent = new Date().toLocaleTimeString('zh-CN', { hour12: false
  };
  updateTime();
  const timerId = setInterval(updateTime, 1000);

  const btn = document.getElementById('action-btn');
  const nameInput = document.getElementById('name-input');
  btn.addEventListener('click', () => {
    const name = nameInput.value.trim() || '开发者';
    // 切换问候卡片显示…
  });

  window.addEventListener('unload', () => clearInterval(timerId));

  // 主题：本课用 localStorage（Popup 专用）；跨上下文请用 chrome.storage
  const savedTheme = localStorage.getItem('theme') || 'dark';
  // …
});
```

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/01-hello-popup
```

也可 Download ZIP 解压后进入 `01-hello-popup` 目录。

加载与调试

1. 打开 `chrome://extensions`，开启开发者模式
2. 点击 加载已解压的扩展程序 → 选择 `01-hello-popup` 目录
3. 固定工具栏图标，点击体验 Popup
4. 调试：在 Popup 内右键 → 检查；DevTools 打开期间 Popup 不会自动关闭

课后练习

1. 输入框定制问候：`Hello, [名字]! 欢迎来到 Chrome 插件世界!`
2. 明暗主题切换并持久化（本课 `localStorage` 即可）

延伸阅读

- 完整源码：`01-hello-popup` — 可运行项目与 README 逐步说明
- Hello World — 更简版最小示例
- manifest.json 概览
- Popup 与选项页
- 调试

小结

第 1 课完成 Popup-only 扩展：清单 + 外部 JS + CSP。下一课用 Content Script 操作当前网页 DOM。

→ 下一课：第 2 课：Content Script

第 2 课：Content Script

本课配套开源 Demo 02-content-script：开发「网页高亮与字数统计助手」，理解 Content Script、隔离世界（Isolated World）与 `chrome.scripting.executeScript` 的三种用法。

▲ *「网页助手」的 Popup：一键高亮段落 / 清除高亮 / 统计网页字数*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `02-content-script` 目录内的 README。

本课目标

- 用 Content Script 读取/修改网页 DOM
- 区分声明式与编程式注入
- 申请 `activeTab` + `scripting`，避免宽泛 host 权限
- 处理 `chrome://` 等受保护页面注入失败

隔离世界

	Content Script	页面自身 JS
DOM	共享	共享
JS 变量 / 函数	独立作用域	独立作用域

网页里的 `$`、React 状态 Content Script 读不到；插件函数也不会污染页面全局——这是防劫持的设计。

manifest.json

```
{
  "manifest_version": 3,
  "name": "网页高亮与统计助手",
  "version": "1.0.0",
  "permissions": ["activeTab", "scripting"],
  "action": {
    "default_popup": "popup.html"
  }
}
```

- `activeTab`：用户点击扩展图标后，临时授权当前标签页——比 `<all_urls>` 友好
- `scripting`：使用 `chrome.scripting.executeScript` 所必需

content.js（文件注入）

```
(function () {
  const paragraphs = document.querySelectorAll('p');
  if (paragraphs.length === 0) {
    alert('当前页面未找到任何段落 (<p> 标签)! ');
    return;
  }
  paragraphs.forEach((p) => {
    p.style.backgroundColor = 'rgba(253, 224, 71, 0.4)';
    p.style.outline = '2px dashed #f59e0b';
  });
})();
```

popup.js：三种 executeScript 模式

1. 注入文件

```
await chrome.scripting.executeScript({
  target: { tabId: tab.id },
  files: ['content.js'],
});
```

2. 注入函数（清除高亮）

```
function clearPageHighlights() {
  document.querySelectorAll('p').forEach((p) => {
    p.style.backgroundColor = '';
    p.style.outline = '';
  });
}
await chrome.scripting.executeScript({
  target: { tabId: tab.id },
  func: clearPageHighlights,
});
```

3. 注入函数并读返回值（字数统计）

```
function analyzeWordCount() {
  const text = document.body.innerText || '';
  const chinese = (text.match(/[\u4e00-\u9fa5]/g) || []).length;
  const english = text.replace(/[^a-zA-Z]/g, ' ').split(/\s+/).filter(Boolean).
  return { chinese, english };
}
const results = await chrome.scripting.executeScript({
  target: { tabId: tab.id },
  func: analyzeWordCount,
});
const { chinese, english } = results[0].result;
```

注入函数在网页环境执行，不能闭包引用 Popup 变量；传参用 `args: [...]`。

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/02-content-script
```

也可 Download ZIP 解压后进入 `02-content-script` 目录。

体验步骤

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `02-content-script` 目录
2. 打开百科/博客等含 `<p>` 的 https 页面（勿用 `chrome://`）

3. 点击扩展 → 高亮所有段落 / 统计网页字数

课后练习

1. 颜色选择器 + `args` 传递高亮颜色
2. 统计 `h1` / `h2` 列表，点击 `scrollIntoView({ behavior: 'smooth' })`

延伸阅读

- 完整源码：`02-content-script` — 可运行项目与 README 逐步说明
- Content Scripts
- Scripting API
- 权限模型

小结

Content Script 改 DOM；Popup 通过 `activeTab` + `scripting` 按需注入。下一课在后台用 Service Worker 监听浏览器事件。

→ 下一课：第 3 课：Service Worker

第 3 课：Service Worker

本课配套开源 Demo 03-service-worker：「标签页浏览计数器」——在 MV3 后台监听标签事件、更新工具栏角标，并理解 Service Worker 无全局状态 原则。

◢ *Service Worker 在后台记录访问与标签切换次数，Popup 实时展示统计*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `03-service-worker` 目录内的 README。

本课目标

- 注册 `background.service_worker`
- 监听 `onInstalled`、`tabs.onActivated`、`tabs.onUpdated`
- 用 `chrome.storage.local` 持久化计数（禁止 `let count = 0` 全局累加）
- 用 `chrome.action.setBadgeText` 显示角标

Service Worker 特点

> ⚠
>
> 1. 事件驱动：空闲约 30s 可被终止，不能假设常驻内存。
> 2. 无 DOM：没有 `window` / `document`。

3. 状态进 storage：重启后全局变量归零。

manifest.json

```
{
  "manifest_version": 3,
  "name": "标签页浏览计数器",
  "version": "1.0.0",
  "permissions": ["storage"],
  "background": {
    "service_worker": "background.js"
  },
  "action": {
    "default_popup": "popup.html"
  }
}
```

background.js 核心逻辑

```
chrome.runtime.onInstalled.addListener(async () => {
  await chrome.storage.local.set({ visitCount: 0, switchCount: 0 });
  await chrome.action.setBadgeText({ text: '0' });
  await chrome.action.setBadgeBackgroundColor({ color: '#10b981' });
});

async function incrementCounter(type) {
  const data = await chrome.storage.local.get(['visitCount', 'switchCount']);
  let visitCount = data.visitCount ?? 0;
  let switchCount = data.switchCount ?? 0;
  if (type === 'visit') visitCount++;
  else if (type === 'switch') switchCount++;
  await chrome.storage.local.set({ visitCount, switchCount });
  await chrome.action.setBadgeText({ text: String(visitCount + switchCount) });
}

chrome.tabs.onActivated.addListener(() => incrementCounter('switch'));

chrome.tabs.onUpdated.addListener((_tabId, changeInfo) => {
  if (changeInfo.status === 'complete') incrementCounter('visit');
});
```

`changeInfo.status === 'complete'` 避免 loading 阶段重复计数。

Popup 读 storage + 重置

```
const data = await chrome.storage.local.get(['visitCount', 'switchCount']);
// 渲染到界面…

resetBtn.addEventListener('click', async () => {
  await chrome.storage.local.set({ visitCount: 0, switchCount: 0 });
  await chrome.action.setBadgeText({ text: '0' });
});
```

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/03-service-worker
```

也可 Download ZIP 解压后进入 `03-service-worker` 目录。

加载与体验

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `03-service-worker` 目录
2. 切换标签页、刷新页面，观察工具栏角标数字变化
3. 打开 Popup 查看计数详情，或点击重置

调试 Service Worker

`chrome://extensions` → 扩展卡片 → Service Worker（或「检查视图」）→ 独立 DevTools。切换标签页观察 Console 日志；卡片显示「无活动视图」表示 SW 已休眠。

课后练习

1. 仅 `complete` 时计 visit（代码已示范）
2. Popup 增加「清空历史」按钮（示例已含 `reset-btn`）

延伸阅读

- 完整源码：`03-service-worker` — 可运行项目与 README 逐步说明
- Service Worker
- 存储 API

小结

后台逻辑放 Service Worker；数字角标用 `chrome.action`；持久化用 storage。下一课让 Popup、Content Script、SW 通过消息协作。

→ 下一课：第 4 课：消息传递

第 4 课：消息传递

本课配套开源 Demo 04-message-passing：「划词翻译助手」——Content Script、Popup、Service Worker 通过 消息总线 协作，并掌握异步响应时 `return true` 的要点。

◢ Popup 与 Service Worker 消息往返后渲染出的译文（真实消息通信结果）

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `04-message-passing` 目录内的 README。

本课目标

- 使用 `chrome.runtime.sendMessage` / `onMessage`
- 声明式注入 `content_scripts`（http/https 自动加载）
- 网页划词 → 悬浮卡片 → 后台翻译 → 回显
- Popup 手动输入走同一后台 handler

为什么需要消息

Popup、Content Script、Service Worker 三个隔离世界，不能直接共享变量。数据交换靠：

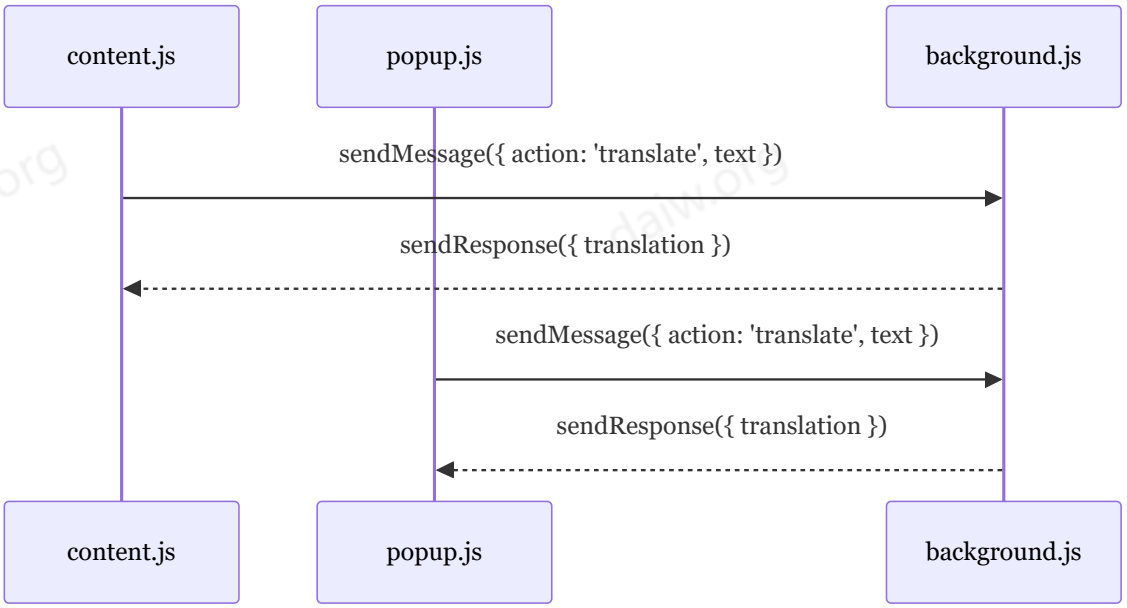
API	方向
`chrome.runtime.sendMessage`	任意上下文 → 扩展内部（通常到 SW）
`chrome.tabs.sendMessage`	扩展 → 指定 tab 的 Content Script
`chrome.runtime.onMessage`	监听入站消息

manifest.json

```
{
  "manifest_version": 3,
  "name": "划词翻译助手",
  "version": "1.0.0",
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
      "matches": ["http://*/*", "https://*/*"],
      "js": ["content.js"]
    }
  ],
  "action": {
    "default_popup": "popup.html"
  }
}
```

本 Demo 零 permissions——声明式 content_scripts + background 即可。

消息流（Mermaid）



background.js：异步必须 return true

```
chrome.runtime.onMessage.addListener((request, _sender, sendResponse) => {
  if (request.action === 'translate') {
    setTimeout(() => {
      sendResponse({
        success: true,
        translation: mockTranslate(request.text),
      });
    }, 500);
    return true; // 保持通道，等待异步 sendResponse
  }
});
```

不写 `return true` 会报错：`The message port closed before a response was received`。

content.js：划词 + 悬浮窗

```
document.addEventListener('mouseup', () => {
  const selectedText = window.getSelection().toString().trim();
  if (!selectedText) return removeTooltip();
  // 创建 position:absolute 悬浮 div，z-index: 2147483647
  chrome.runtime.sendMessage(
    { action: 'translate', text: selectedText },
    (response) => {
      if (chrome.runtime.lastError) { /* … */ return; }
      updateResult(response.translation);
    },
  );
});
```

悬浮层用内联样式 + `!important`，降低被宿主 CSS 覆盖的概率。

popup.js

```
chrome.runtime.sendMessage({ action: 'translate', text }, (response) => {
  translatedTextEl.textContent = response?.translation ?? '失败';
});
```

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/04-message-passing
```

也可 Download ZIP 解压后进入 `04-message-passing` 目录。

体验步骤

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `04-message-passing` 目录
2. 刷新已打开网页（让 content.js 注入）
3. 网页选中英文 → 悬浮译文
4. Popup 输入文字 → 点击翻译

调试：网页 Console 看 content 日志；`chrome://extensions` 看 SW 日志。

课后练习

1. `background.js` 用 `fetch` 接真实翻译 API（见 网络与 CORS）
2. Popup 按钮 `chrome.tabs.sendMessage` 改网页背景色

延伸阅读

- 完整源码：`04-message-passing` — 可运行项目与 README 逐步说明
- 消息传递
- Content Scripts

小结

扩展各上下文靠消息协作；异步 `sendResponse` 必须 `return true`。下一课系统学习 `chrome.storage`。

→ 下一课：第 5 课：Storage API

第 5 课：Storage API

本课配套开源 Demo 05-storage-api：「待办事务清单」——在 Popup 做 Todo CRUD，后台监听 `chrome.storage.onChanged` 自动更新角标。

◢ *基于 chrome.storage 的待办清单，数据持久保存、刷新不丢*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `05-storage-api` 目录内的 README。

本课目标

- 理解为何不用 `localStorage`（SW 不可用、多进程冲突）
- 使用 `chrome.storage.local` 存对象数组
- `onChanged` 实现 Popup ↔ Background 状态同步
- Todo 渲染时 `escapeHtml` 防 XSS

为什么不用 localStorage

问题	说明
Service Worker	同步 API，Worker 中禁用
多上下文	Popup / CS / SW 不同进程，竞态严重
类型	只能存字符串，需手动 JSON

`chrome.storage`：异步、全上下文、直接存对象、可选 sync 云同步。

存储区域

区域	容量	用途
`local`	约 10 MB	Todo、缓存、大列表
`sync`	约 100 KB	用户配置（第 6 课）
`session`	内存	浏览器关闭即失效

manifest.json

```
{
  "manifest_version": 3,
  "name": "待办事务清单",
  "permissions": ["storage"],
  "background": { "service_worker": "background.js" },
  "action": { "default_popup": "popup.html" }
}
```

数据流

```
Popup chrome.storage.local.set({ todos })
        ↓
chrome.storage.onChanged（系统总线）
        ↓
background.js ⟶ 统计未完成数 ⟶ setBadgeText
```

popup.js：CRUD 片段

```
const newTodo = { id: Date.now(), text, completed: false };
todos.push(newTodo);
await chrome.storage.local.set({ todos });

// 渲染时用 escapeHtml 防 XSS
function escapeHtml(s) {
  return s.replace(/[&<>"']/g, (m) => ({ '&':'&amp;','<':'&lt;' }[m]));
}
```

background.js：onChanged

```
chrome.storage.onChanged.addListener(async (changes, areaName) => {
  if (areaName === 'local' && changes.todos) {
    await updateBadge(changes.todos.newValue);
  }
});

async function updateBadge(todosList) {
  const list = todosList ?? (await chrome.storage.local.get('todos')).todos ?? [
  const pending = list.filter((t) => !t.completed).length;
  await chrome.action.setBadgeText({ text: pending > 0 ? String(pending) : '' });
}
```

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/05-storage-api
```

也可 Download ZIP 解压后进入 `05-storage-api` 目录。

体验

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `05-storage-api` 目录
2. 点击扩展图标，添加/完成/删除 Todo → 角标数字实时变化
3. 关闭浏览器再开，数据仍在 `local`

课后练习

1. 读写改为 `chrome.storage.sync`，多设备同步测试
2. 底部 Tab：全部 / 待办 / 已完成

延伸阅读

- 完整源码：`05-storage-api` — 可运行项目与 README 逐步说明
- 存储 API
- Service Worker

小结

业务状态进 `chrome.storage`；`onChanged` 让后台无需 Popup 主动通知。下一课用 Options Page + `sync` 做用户配置。

→ 下一课：第 6 课：Options Page

第 6 课：Options Page

本课配套开源 Demo 06-options-page：「个性化配置中心」——选项页保存昵称与主题色到 `chrome.storage.sync`，Popup 读取并应用样式。

◢ *将昵称与主题色保存到 chrome.storage.sync 的选项页*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `06-options-page` 目录内的 README。

本课目标

- 在 manifest 注册 `options_ui`
- 三种入口打开选项页
- Options 写 sync、Popup 读 sync 实现联动
- 理解嵌入 Modal vs 独立标签页

Options Page 是什么

成熟扩展需要设置页：API Key、快捷键、主题等。

进入方式：

- `chrome://extensions` 卡片 → 扩展程序选项
- 工具栏图标右键 → 选项
- 代码：`chrome.runtime.openOptionsPage()`

manifest.json

```
{
  "manifest_version": 3,
  "name": "个性化配置中心",
  "permissions": ["storage"],
  "options_ui": {
    "page": "options.html",
    "open_in_tab": false
  },
  "action": {
    "default_popup": "popup.html"
  }
}
```

open_in_tab	行为
false	在扩展管理页内嵌入 Modal（MV3 推荐）
true	新标签页打开完整选项页

options.js

```
const data = await chrome.storage.sync.get({
  username: '开发者',
  themeColor: 'blue',
});
usernameInput.value = data.username;
document.querySelector(`input[value="${data.themeColor}"]`).checked = true;

saveBtn.addEventListener('click', async () => {
  await chrome.storage.sync.set({
    username: usernameInput.value.trim() || '开发者',
    themeColor: document.querySelector('input[name="theme-color"]:checked').valu
  });
});
```

popup.js（联动 Popup）

仓库 `06-options-page` 中 `popup.js` 需与 `popup.html` 配套；核心逻辑如下：

```
document.addEventListener('DOMContentLoaded', async () => {
  const data = await chrome.storage.sync.get({
    username: '开发者',
    themeColor: 'blue',
  });

  document.getElementById('welcome-msg').textContent = `你好，${data.username}! `

  const card = document.getElementById('popup-card');
  card.classList.remove('theme-blue', 'theme-emerald', 'theme-purple', 'theme-r
  card.classList.add(`theme-${data.themeColor}`);

  document.getElementById('open-options-btn').addEventListener('click', () => {
    chrome.runtime.openOptionsPage();
  });
});
```

CSS 中为 `.theme-blue` / `.theme-emerald` / `.theme-purple` / `.theme-rose` 定义不同渐变色即可。

> 仓库 `06-options-page` 已补全 `popup.js` 与 `popup.css`，可直接在 `chrome://extensions` 加载。

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/06-options-page
```

也可 Download ZIP 解压后进入 `06-options-page` 目录。

体验流程

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `06-options-page` 目录
2. Popup → 打开配置页面 → 改昵称与主题色 → 保存
3. 再开 Popup：问候语与主题色已更新

课后练习

1. 恢复默认：清除 sync 中 `username` / `themeColor`
2. 选项页右侧实时预览（改 radio 即更新预览卡片，无需点保存）

延伸阅读

- 完整源码：`06-options-page` — 可运行项目与 README 逐步说明
- Popup 与选项页
- 存储 API
- manifest.json 概览

小结

Options + `storage.sync` 是用户配置的标配；Popup 只负责展示与跳转。下一课学习右键 Context Menus。

→ 下一课：第 7 课：Context Menus

第 7 课：Context Menus

本课配套开源 Demo 07-context-menus：「右键快捷收藏与搜索工具」——用 `chrome.contextMenus` 按选中文字/图片/链接显示不同菜单，并通过 Content Script 在网页内弹出 Toast。

◢ *右键菜单触发后，Content Script 在网页右上角弹出的 Toast（真实网页注入效果）*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `07-context-menus` 目录内的 README。

本课目标

- 在 `onInstalled` 中注册菜单（避免重复 `create`）
- 使用 `contexts`：`selection`、`image`、`link`、`page`
- `onClicked` 处理收藏、搜索、开新标签
- `chrome.tabs.sendMessage` 向页面展示操作反馈

核心概念

右键菜单项由 Service Worker 注册，点击后在 `background.js` 处理。操作完成后必须给用户可见反馈（Toast、通知或角标），不能静默执行。

contexts	出现场景
`selection`	用户选中文字
`image`	右键图片
`link`	右键超链接
`page`	页面空白处

manifest.json

```
{
  "manifest_version": 3,
  "name": "右键收藏工具",
  "permissions": ["contextMenus", "scripting", "activeTab", "storage"],
  "host_permissions": ["<all_urls>"],
  "background": { "service_worker": "background.js" },
  "content_scripts": [{
    "matches": ["http://*/*", "https://*/*"],
    "js": ["content.js"]
  }]
}
```

background.js：注册与点击

```
chrome.runtime.onInstalled.addListener(() => {
  chrome.contextMenus.create({
    id: 'save-text',
    title: '📚 收藏选中文字',
    contexts: ['selection'],
  });
  chrome.contextMenus.create({
    id: 'search-text',
    title: '🔍 百度搜索选中文字',
    contexts: ['selection'],
  });
});

chrome.contextMenus.onClicked.addListener(async (info, tab) => {
  if (info.menuItemId === 'save-text') {
    await saveFavorite({ type: 'text', content: info.selectionText });
    await chrome.tabs.sendMessage(tab.id, {
      action: 'showToast',
      message: '📚 已收藏选中文字',
    });
  }
  if (info.menuItemId === 'search-text') {
    const q = encodeURIComponent(info.selectionText);
    await chrome.tabs.create({ url: `https://www.baidu.com/s?wd=${q}` });
  }
});
```

Demo 还支持父子菜单、`image` / `link` 收藏，收藏列表写入 `chrome.storage.local` 的 `favorites` 数组（最多 100 条）。

content.js：Toast

监听 `showToast`，在 `body` 注入固定定位卡片，`setTimeout` 3 秒后移除。`tabs.sendMessage` 在 `chrome://` 页面会失败，需 `try/catch` 静默处理。

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/07-context-menus
```

也可 Download ZIP 解压后进入 `07-context-menus` 目录。

体验步骤

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `07-context-menus` 目录

2. 普通 https 页面：选中文字 → 右键 → 收藏 / 搜索
3. 右键图片 → 收藏图片链接
4. 在 `chrome://extensions` 打开 SW 控制台：`chrome.storage.local.get('favorites')` 查看数据

课后练习

1. 增加 `popup.html` 展示收藏列表并支持删除
2. 搜索引擎前缀存 `storage.local`，支持百度/必应/谷歌切换

延伸阅读

- 完整源码：`07-context-menus` — 可运行项目与 README 逐步说明
- 消息传递 — `tabs.sendMessage`
- 存储 API

小结

右键菜单 = SW 注册 + `contexts` 过滤 + 点击后 storage/导航/页面反馈。下一课学习常驻 Side Panel。

→ 下一课：第 8 课：Side Panel

第 8 课：Side Panel

本课配套开源 Demo 08-side-panel：「边栏网页笔记本」——Side Panel 不会因失焦关闭，适合需要持续操作的工具界面。

▲ *常驻侧边栏的笔记面板，切换标签也不关闭，内容每 2 秒自动保存*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `08-side-panel` 目录内的 README。

Side Panel vs Popup

特性	Popup	Side Panel
触发	点击图标（`default_popup`）	`chrome.sidePanel.open()`
持久性	失焦即销毁	常驻侧边栏
典型场景	快捷操作	笔记、AI 助手、阅读辅助

> ⚠️ 仅声明 `side_panel.default_path` 不会自动打开侧边栏，必须由代码调用 `chrome.sidePanel.open()`。
> `chrome.action.onClicked` 仅在 没有 `default_popup` 时触发。

> ⚠️ `activeTab` 不适用于侧边栏内触发的 tabs 操作。从 Side Panel 读当前标签 URL 需 `tabs` 权限，不能依赖 `activeTab`。

manifest.json

```
{
  "manifest_version": 3,
  "name": "边栏笔记本",
  "permissions": ["sidePanel", "storage", "tabs"],
  "side_panel": { "default_path": "sidepanel.html" },
  "background": { "service_worker": "background.js" },
  "action": {}
}
```

`"action": {}` 为空对象，无 `default_popup`，图标点击走 `onClicked`。

background.js

```
chrome.action.onClicked.addListener(async (tab) => {
  await chrome.sidePanel.open({ windowId: tab.windowId });
});
```

sidepanel.js 要点

- 笔记存 `chrome.storage.local` 的 `sidebarNote`
- 输入防抖 2 秒自动保存
- 插入当前页 URL：`chrome.tabs.query({ active: true, lastFocusedWindow: true })` 取标题与 URL 追加到笔记

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/08-side-panel
```

也可 Download ZIP 解压后进入 `08-side-panel` 目录。

加载与体验

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `08-side-panel` 目录
2. 点击工具栏扩展图标，侧边栏应打开
3. 在笔记区输入文字，切换标签页后测试「插入当前页 URL」

调试

侧边栏内右键 → 检查；或 `chrome://extensions` → 检查视图：侧边栏。

课后练习

1. 按域名 `hostname` 分桶保存多份笔记，切换标签自动加载
2. 简单 Markdown 预览（`#`、`**`、列表）

延伸阅读

- 完整源码：`08-side-panel` — 可运行项目与 README 逐步说明
- Side Panel
- 权限模型

小结

Side Panel 适合长会话 UI；打开侧栏与读 tabs 的权限设计要早于功能编码。下一课学习 DNR 网络拦截。

→ 下一课：第 9 课：Declarative Net Request

第 9 课：Declarative Net Request

本课配套开源 Demo 09-declarative-net-request：「简易广告资源拦截器」——MV3 用 DNR 替代阻塞式 `webRequest`，规则在浏览器网络层执行，扩展 JS 不接触请求正文。

◢ DNR 预置静态规则 + 自定义拦截域名列表

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `09-declarative-net-request` 目录内的 README。

为什么不用 webRequest

MV2 的 `webRequest` 可阻塞并窥视全部请求（含 Cookie），性能与隐私风险大。DNR 改为声明规则，Chrome 内核匹配 URL 后执行 `block` / `redirect` 等，扩展看不到响应体。

规则结构

```
{
  "id": 1,
  "priority": 1,
  "condition": {
    "urlFilter": "||doubleclick.net^",
    "resourceTypes": ["image", "script", "xmlhttprequest"]
  },
  "action": { "type": "block" }
}
```

- `urlFilter`：见 urlFilter 语法
- `resourceTypes`：`image`、`script`、`stylesheet`、`sub_frame` 等
- `action.type`：`block`、`redirect`、`modifyHeaders`、`allow`

静态规则 vs 动态规则

类型	来源	说明
静态	`rules.json` + `declarative_net_request.rule_resources`	随包发布
动态	`chrome.declarativeNetRequest.updateDynamicRules()`	运行时增删，适合用户黑名单

manifest.json

```
{
  "permissions": ["declarativeNetRequest", "storage"],
  "declarative_net_request": {
    "rule_resources": [{
      "id": "ruleset_1",
      "enabled": true,
      "path": "rules.json"
    }]
  },
  "background": { "service_worker": "background.js" },
  "action": { "default_popup": "popup.html" }
}
```

Demo 的 `rules.json` 预置 doubleclick、googlesyndication、Facebook 像素等 5 条静态规则。

动态规则（background.js）

```
await chrome.declarativeNetRequest.updateDynamicRules({
  addRules: [{
    id: 1001,
    priority: 2,
    condition: {
      urlFilter: `||${domain}^`,
      resourceTypes: ['image', 'script', 'xmlhttprequest'],
    },
    action: { type: 'block' },
  }],
});
```

动态规则 ID 建议从 1000+ 起，避免与静态规则冲突。列表同步到 `chrome.storage.local` 以便重装恢复。

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/09-declarative-net-request
```

也可 Download ZIP 解压后进入 `09-declarative-net-request` 目录。

加载与体验

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `09-declarative-net-request` 目录
2. 访问含广告资源的 https 页面，观察 Network 面板中被拦截的请求

验证拦截

DevTools → Network：被拦请求显示红色，状态 blocked:extension。Popup 可展示累计拦截次数（`onRuleMatchedDebug`，开发调试用）。

课后练习

1. Popup 管理用户黑名单（增删动态规则 + storage 持久化）
2. `redirect` 到 1×1 透明 GIF 替换广告图

延伸阅读

- 完整源码：`09-declarative-net-request` — 可运行项目与 README 逐步说明
- declarativeNetRequest

小结

DNR = 声明规则 + 网络层执行；静态打底、动态可配置。下一课在后台使用 Offscreen Document 调用 TTS 等 DOM API。

→ 下一课：第 10 课：Offscreen Document

第 10 课：Offscreen Document

本课配套开源 Demo 10-offscreen-document：「网页文字转语音播放器」——Service Worker 无 DOM，通过 Offscreen Document 使用 `SpeechSynthesis` 等 Web API。

▲ *输入文字后交给 Offscreen Document 调用 Web Speech API 朗读*

> 本文讲解原理与关键片段；完整 HTML/CSS/JS 及逐步操作见仓库 `10-offscreen-document` 目录内的 README。

为什么需要 Offscreen

Service Worker 没有 `window` / `document`，无法直接调用：

- `SpeechSynthesisUtterance`（TTS）
- `HTMLCanvasElement`
- `Audio`、`DOMParser`

Offscreen 创建用户不可见的 HTML 文档，完整支持上述 API。

> ⚠ Offscreen 内几乎不能用 `chrome.tabs`、`chrome.storage`、`chrome.action` 等——仅 `runtime.sendMessage` / `getURL` 等少数 API。其余操作必须由 Service Worker 代理。

> ℹ 全局同时只能存在 一个 Offscreen Document。`createDocument` 前必须 `chrome.offscreen.hasDocument()`。

通信架构

```
Popup sendMessage('speak')
    ⟶ background.js
        ⟶ ensureOffscreenDocument()
        ⟶ sendMessage({ action: 'doSpeak', target: 'offscreen' })
    ⟶ offscreen.js ⟶ speechSynthesis.speak()
        ⟶ sendMessage('speechEnded')
    ⟶ background.js ⟶ closeDocument() + 清除 Badge
```

manifest.json

```
{
  "permissions": ["offscreen"],
  "background": { "service_worker": "background.js" },
  "action": { "default_popup": "popup.html" }
}
```

background.js 片段

```
async function ensureOffscreenDocument() {
  if (!(await chrome.offscreen.hasDocument())) {
    await chrome.offscreen.createDocument({
      url: chrome.runtime.getURL('offscreen.html'),
      reasons: [chrome.offscreen.Reason.AUDIO_PLAYBACK],
      justification: '使用 Web Speech API 进行 TTS 播放',
    });
  }
}

chrome.runtime.onMessage.addListener((msg, _sender, sendResponse) => {
  if (msg.action === 'speak') {
    handleSpeak(msg.text, sendResponse);
    return true;
  }
});
```

offscreen.js 片段

```
chrome.runtime.onMessage.addListener((message, _sender, sendResponse) => {
  if (message.target !== 'offscreen') return;
  if (message.action === 'doSpeak') {
    const u = new SpeechSynthesisUtterance(message.text);
    u.lang = /[\u4e00-\u9fa5]/.test(message.text) ? 'zh-CN' : 'en-US';
    u.onend = () => chrome.runtime.sendMessage({ action: 'speechEnded' });
    speechSynthesis.speak(u);
    sendResponse({ success: true });
  }
});
```

获取源码

```
git clone https://github.com/davidapp/chrome-extension-demos.git
cd chrome-extension-demos/10-offscreen-document
```

也可 Download ZIP 解压后进入 `10-offscreen-document` 目录。

体验与调试

1. 打开 `chrome://extensions` → 开发者模式 → 加载已解压的扩展程序 → 选择 `10-offscreen-document` 目录
2. Popup 输入文字 → 朗读 / 停止
3. 朗读期间 `chrome://extensions` 出现 检查视图：offscreen；播放结束 SW 关闭文档，链接消失

10 课能力总览

课	核心能力	源码
01	MV3、Popup、CSP	01-hello-popup
02	Content Script、`scripting`	02-content-script
03	Service Worker、Badge、storage	03-service-worker
04	`sendMessage`、`return true`	04-message-passing
05	`storage.local`、`onChanged`	05-storage-api
06	`options_ui`、`storage.sync`	06-options-page
07	`contextMenus`、Toast 反馈	07-context-menus
08	Side Panel、`tabs` 权限	08-side-panel
09	DNR 静态/动态规则	09-declarative-net-request
10	Offscreen、后台 TTS	10-offscreen-document

完整仓库：davidapp/chrome-extension-demos

课后练习

1. Popup 增加语速滑块、`lang` 下拉，传给 Offscreen 设置 `utterance.rate` / `utterance.lang`
2. 用 Offscreen Canvas 生成文字像素图，`toDataURL` 回 Popup 展示

延伸阅读

- 完整源码：`10-offscreen-document` — 可运行项目与 README 逐步说明
- Service Worker（第 3 课曾提及 Offscreen，见本篇）
- 调试
- 上架 Chrome Web Store

小结

Offscreen 补全 SW 的 DOM 能力缺口；职责分离：Popup 触发、SW 管生命周期、Offscreen 只跑 Web API。十课实战完结，可继续阅读专栏工程化章节。

PWA 开发指南

渐进式 Web 应用（Progressive Web App，PWA） 是让普通网页具备「可安装、可离线、可推送」等类原生体验的一套 Web 标准。它和 Chrome 扩展共享部分技术名词（如 Service Worker），但运行模型、分发渠道、能力边界完全不同。本篇作为附录，帮你在「做扩展」之外判断何时该做 PWA，并给出最小可运行路径。

◢ *可安装、离线可用的记事本 PWA —— 页脚显示「SW: 已注册 ✓」*

> 本附录面向已读完 扩展是什么 的读者。PWA 不依赖 `chrome.*` API，可在任意现代浏览器上渐进增强。

与 Chrome 扩展的边界

维度	PWA	Chrome 扩展（MV3）
入口	用户访问 URL，可选「安装到桌面/开始菜单」	`chrome://extensions` 或 Chrome Web Store
清单文件	`manifest.webmanifest`（Web App Manifest）	`manifest.json`（Extension Manifest）
后台逻辑	页面域下的 Web Service Worker	扩展域下的 Extension Service Worker
改任意网站 DOM	否（仅限自己的源）	是（Content Script + `host_permissions`）
拦截第三方请求	仅缓存自己源资源	是（`declarativeNetRequest` 等）
审核	无商店审核（自托管）	Chrome Web Store 审核（若上架）

选型口诀：能力要「钉」在别人的网站上 → 扩展或油猴；能力只在「自己的站点」且希望可安装、离线 → PWA。

最小 PWA 需要三件套

```
your-pwa/
├── index.html           # 可访问的 HTTPS 页面
├── manifest.webmanifest
├── sw.js                # Web Service Worker
└── icons/
    └── icon-192.png
```

1. HTTPS（`localhost` 开发除外）——Service Worker 只在安全上下文注册。
2. Web App Manifest——声明名称、图标、`start_url`、`display: standalone` 等。
3. Service Worker——负责离线缓存、推送（可选）、后台同步（可选）。

manifest.webmanifest 示例

```
{
  "name": "道雾记事本",
  "short_name": "记事本",
  "description": "离线可用的简易记事 PWA",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#1a1a2e",
  "icons": [
    {
      "src": "/icons/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any maskable"
    }
  ]
}
```

在 HTML 中链接：

```
<link rel="manifest" href="/manifest.webmanifest" />
<meta name="theme-color" content="#1a1a2e" />
```

Web Service Worker：离线缓存

```
// sw.js
const CACHE = 'notes-v1';
const ASSETS = ['/', '/index.html', '/app.js', '/styles.css'];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE).then((cache) => cache.addAll(ASSETS)),
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((cached) => cached ?? fetch(event.request)
  );
});
```

在页面注册（注意与扩展 SW 不是同一个 API 命名空间）：

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js');
}
```

安装提示（beforeinstallprompt）

Chrome 在满足 PWA 安装条件时触发 `beforeinstallprompt`，可自定义安装按钮：

```
let deferredPrompt;

window.addEventListener('beforeinstallprompt', (e) => {
  e.preventDefault();
  deferredPrompt = e;
  document.getElementById('install-btn').hidden = false;
});

document.getElementById('install-btn').addEventListener('click', async () => {
  if (!deferredPrompt) return;
  deferredPrompt.prompt();
  await deferredPrompt.userChoice;
  deferredPrompt = null;
});
```

> ⚠ iOS Safari 对 PWA 安装与推送的支持与 Chrome 不同；若主要面向国内移动端，需单独验证「添加到主屏幕」与离线行为。

例子：同一功能两种实现

需求：给用户一个「随时打开的 Todo 列表」。

方案	做法	优点	缺点
PWA	部署 `https://todo.example.com`，用户安装到桌面	无需商店审核、跨浏览器、迭代快	不能自动注入 GitHub 页面
扩展	Popup + `chrome.storage`	工具栏一键打开、可读写指定站点	需打包、上架要审核

若 Todo 只服务你自己的域名，PWA 更轻；若要在 `github.com` 旁路显示项目 Todo，只能扩展或油猴。

与扩展技术的易混点

1. 两个 Service Worker——PWA 的 SW 挂在 `https://your-domain/sw.js`；扩展 SW 挂在 `chrome-extension://<id>/service-worker.js`，API 面有 `chrome.*` 扩展专用接口。

2. 两个 manifest——`manifest.webmanifest`（W3C）与 `manifest.json`（Chrome Extension）字段名部分重叠（`name`、`icons`），不可混用文件。
3. 存储——PWA 用 IndexedDB / Cache API / localStorage；扩展用 `chrome.storage` 且可跨设备 sync。

常见误区

1. 「PWA 能替代扩展」——不能跨站改 DOM、不能声明 `host_permissions` 级别的网络拦截。
2. 「注册了 SW 就等于 PWA」——还需有效 manifest + 图标 + `start_url`，且站点的 Lighthouse PWA 检查 才会通过安装条件。
3. 「PWA 一定能离线」——取决于 SW 缓存策略；默认 `fetch` 仍可能回源失败。

调试与验收

- Chrome DevTools → Application 面板：查看 Manifest、Service Workers、Cache Storage。
- Lighthouse → PWA 类别：检查可安装性与离线能力。
- `chrome://apps`：查看已安装的 PWA（与扩展分开管理）。

延伸阅读

- MDN：渐进式 Web 应用
- web.dev：PWA 清单
- 本站对比入门：扩展是什么

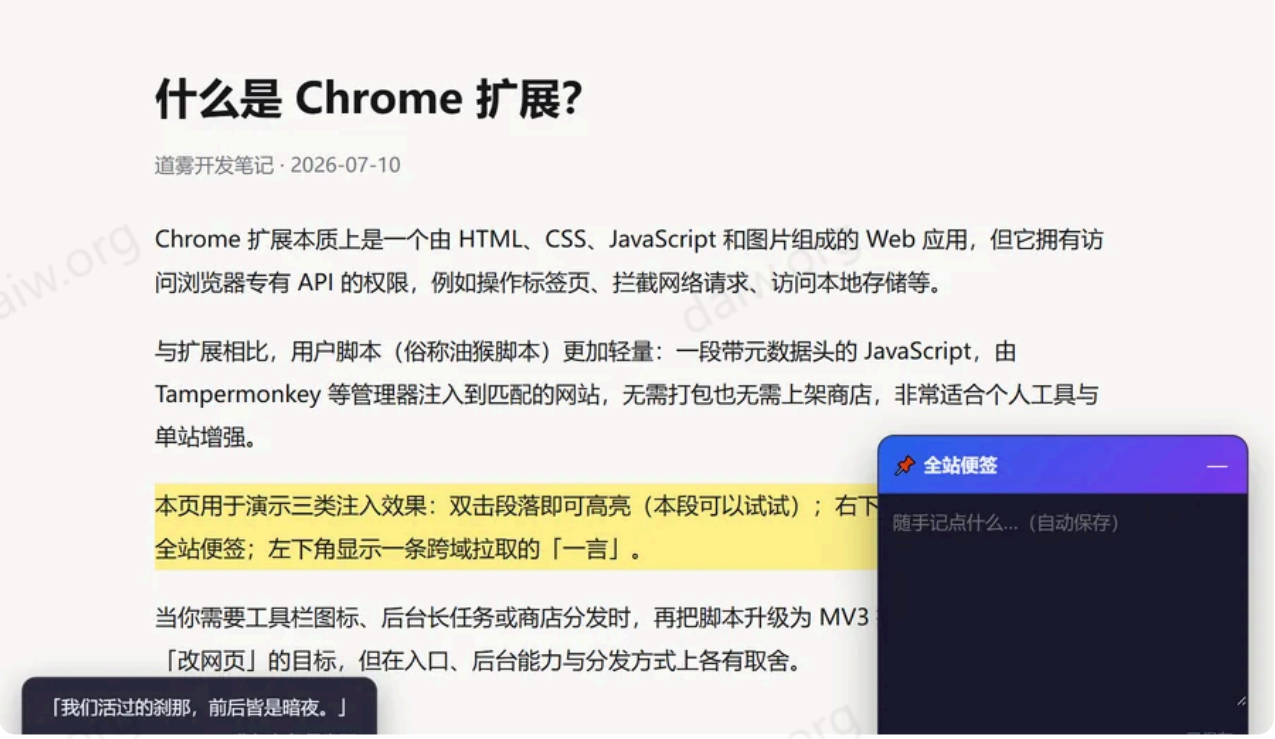
小结

PWA 是自有站点上的可安装 Web 应用，用 Web Manifest + Web Service Worker 实现离线与应用壳体验；Chrome 扩展是浏览器级特权程序，可跨站注入与拦截。先问「功能是否只服务我的域名」——是则优先 PWA，否则回到扩展或 油猴脚本 路线。

← 回到：专栏首页

油猴脚本开发指南

用户脚本（Userscript） 是一段带元数据头的 JavaScript，由 Tampermonkey、Violentmonkey、Greasemonkey 等管理器注入到匹配的网站。它常被称为「油猴脚本」。能力上最接近 Chrome 扩展的 Content Script，但无需打包、无需上架商店，适合个人工具与站点增强。



▲ *同一网页上三个脚本的注入效果：段落高亮、右下角全站便签、左下角跨域「一言」角标*

> 本附录默认以 Tampermonkey（Chrome 安装量最大的用户脚本管理器）为例。Violentmonkey 语法大体兼容，部分 `GM_*` API 名称略有差异。

与 Chrome 扩展的边界

维度	油猴脚本	**Chrome 扩展（MV3）**
分发	粘贴脚本 / 从 Greasy Fork 安装	未打包目录 / Chrome Web Store
运行上下文	页面主世界或隔离世界（取决于管理器与 `@grant`）	Content Script 固定隔离世界
后台常驻	无独立 Service Worker；随页面加载执行	Extension Service Worker
工具栏 / Popup	依赖管理器 UI，无原生 `action`	原生 Popup、Side Panel
改任意站	`@match` 声明 URL 模式	`content_scripts` + `host_permissions`
审核	无 Google 审核（脚本来源自负）	商店扩展需审核

选型口诀：个人用、只改少数站点、快速迭代 → 油猴；要工具栏图标、后台长任务、商店分发、企业策略部署 → 扩展。

脚本文件结构

油猴脚本是一个 `.user.js` 文件，顶部元数据块由管理器解析，下方是业务逻辑：

```
// ==UserScript==
// @name         划词高亮
// @namespace    https://daiw.org/
// @version      1.0.0
// @description  双击段落添加黄色高亮
// @author       You
// @match        https://*/*
// @match        http://*/*
// @grant        none
// @run-at       document-idle
// ==/UserScript==

(function () {
  'use strict';

  document.addEventListener('dblclick', (e) => {
    const p = e.target.closest('p');
    if (!p) return;
    p.style.backgroundColor = '#fef08a';
  });
})();
```

在 Tampermonkey 中：添加新脚本 → 粘贴 → 保存。匹配页面刷新后即生效。

常用元数据字段

字段	含义	示例
`@name`	脚本显示名称	`划词高亮`
`@match` / `@include`	注入哪些 URL	`https://github.com/*`
`@exclude`	排除 URL	`https://github.com/settings/*`
`@grant`	申请的特权 API	`GM_setValue`、`GM_xmlhttpRequest`
`@run-at`	注入时机	`document-start` / `document-idle` / `document-end`
`@connect`	`GM_xmlhttpRequest` 允许访问的域名	`api.example.com`
`@require`	注入外部库（CDN）	`https://cdn.jsdelivr.net/npm/jquery@3`
`@icon`	脚本图标 URL	

`@match` 使用 match 模式 与扩展类似：`<scheme>://<host><path>`。

@grant 与 GM API

`@grant none` 表示脚本在页面上下文中运行，只能使用页面已有的 DOM / `fetch`（受页面 CSP 约束）。

需要跨域请求、持久存储、注册菜单时，声明对应 grant：

```
// ==UserScript==
// @name         跨域天气角标
// @match        https://news.example.com/*
// @grant        GM_xmlhttpRequest
// @grant        GM_setValue
// @grant        GM_getValue
// @connect      api.weather.example.com
// ==/UserScript==

(function () {
  'use strict';

  const KEY = 'lastTemp';

  GM_xmlhttpRequest({
    method: 'GET',
    url: 'https://api.weather.example.com/v1/current',
    onload(res) {
      const { temp } = JSON.parse(res.responseText);
      GM_setValue(KEY, temp);
      const badge = document.createElement('div');
      badge.textContent = `${temp}°C`;
      badge.style.cssText =
        'position:fixed;top:8px;right:8px;z-index:99999;padding:4px 8px;backgrou
      document.body.appendChild(badge);
    },
  });
})();
```

API	作用	扩展中的近似能力
`GM_setValue` / `GM_getValue`	键值存储	`chrome.storage.local`
`GM_xmlhttpRequest`	跨域 XHR（需 `@connect`）	`fetch` + `host_permissions`
`GM_addStyle`	注入 CSS	Content Script 注入 `<style>`
`GM_registerMenuCommand`	管理器菜单项	`chrome.contextMenus`（扩展侧）
`GM_notification`	系统通知	`chrome.notifications`

与 Content Script 的对比

扩展 Content Script（见 第 2 课）：

- 运行在隔离世界，默认读不到页面 `window.jQuery`。
- 通过 `chrome.runtime.sendMessage` 与 Service Worker 通信。

油猴脚本（`@grant none` 时）：

- 常运行在页面主世界，可直接调用站点已有库（也更容易被页面 CSP 或反篡改检测）。
- 无内置「扩展后台」；长连接、定时任务需自己用 `setInterval` 或配合管理器能力。

若需要稳定后台 + 最小页面耦合，扩展更合适；若只是「给 Wikipedia 加个深色模式」，油猴更快。

例子：站点阅读模式（最小版）

```
// ==UserScript==
// @name         简易阅读模式
// @match        https://en.wikipedia.org/wiki/*
// @grant        GM_addStyle
// @run-at       document-end
// ==/UserScript==

GM_addStyle(`
  #content { max-width: 42rem; margin: 2rem auto; font-size: 18px; line-height:
  #mw-navigation, .noprint { display: none !important; }
`);
```

等价的 MV3 扩展需要：`manifest.json`、`content_scripts` 匹配规则、可能还有 Popup 开关——文件更多，但可上架商店、统一图标入口。

安全与维护

1. 只安装可信来源——脚本拥有匹配页面的完整访问权，恶意脚本可窃取 Cookie、篡改表单。
2. `@require` 远程库——CDN 被劫持时脚本行为会被替换；生产环境优先内联或锁定版本。
3. 站点改版——选择器失效时脚本静默失败；扩展同样面临 DOM 变更，但可用 `chrome.scripting` 动态调试。
4. 企业环境——部分公司禁止 Tampermonkey；扩展可通过策略强制安装。

从油猴迁移到扩展

当你需要以下能力时，应考虑升级为 MV3 扩展：

- 工具栏图标、Side Panel、快捷键
- `declarativeNetRequest` 广告拦截
- `chrome.storage.sync` 跨设备同步
- Chrome Web Store 公开分发

迁移步骤概要：

1. 将 `@match` 转为 `content_scripts.matches`。
2. 将 `GM_*` 调用映射为 `chrome.storage`、`fetch`（带 `host_permissions`）。
3. 把需后台逻辑的部分迁入 Extension Service Worker。
4. 参考本站 Hello World 与 10 课实战 搭骨架。

常见误区

1. 「油猴能调用 chrome.*」——不能；除非再装一个桥接扩展。
2. 「@grant none 最安全」——仍与页面共享上下文，XSS 可影响脚本；敏感操作应用 `@grant` 隔离 API。
3. 「一个脚本匹配所有 URL」——`@match *://*/*` 会在每个页面注入，浪费性能且增加暴露面；尽量收窄匹配。

调试

- Tampermonkey 面板 → 脚本 → 检查视图（Inspector）查看注入时机与错误。
- 普通页面 DevTools Console 中错误栈指向 `userscript.html` 或脚本名。
- 使用 `console.log` + `@run-at document-idle` 避免 DOM 未就绪。

延伸阅读

- Tampermonkey 文档
- Greasy Fork——公开脚本托管与评论
- 本站：消息传递（扩展侧页面通信）

小结

油猴脚本是轻量级的页面增强方案：元数据头声明匹配与权限，`GM_*` 补齐跨域与存储。它与扩展共享「改网页」的目标，但省去 manifest 打包与商店流程。个人工具、单站增强优先油猴；要全局入口、后台能力与正规分发，用本专栏主线里的 MV3 扩展。

← 回到：专栏首页 · 相关：PWA 开发指南

道雾轩

《Chrome 插件开发指南》