# Chrome 插件（Manifest V3）Service Worker：API 调研报告

## Executive Summary

Chrome 扩展（Manifest V3）将后台上下文从“长期驻留的 background page/event page”迁移为“扩展专用的 Service Worker（extension service worker）”，其本质是**事件驱动**的中心调度器：按需唤醒、空闲后终止，要求你用“可恢复、可持久化、可重入”的方式设计后台逻辑。[1][2]

对开发影响最大的点是：**生命周期更短（默认 30s 空闲终止）、无 DOM、全局变量不可靠、部分旧 API/模式不再适用**；与之对应的解决方案是：用 `chrome.storage*` 持久化状态、用 Offscreen Document 承接 DOM 需求、用声明式网络规则（DNR）替代阻塞式 `webRequest`，并把监听器注册放在顶层、把业务拆成“事件 → 短任务”的形态。[1][4][6][9]

## Key Findings

- **生命周期约束是架构核心**：SW 通常在空闲 30 秒后被终止；单次任务最长 5 分钟；`fetch()` 等待响应超过 30 秒也可能触发终止，设计必须容忍随时“被杀”。[1]
- **扩展 SW 不能访问 DOM，也没有 Web Storage（`localStorage`）**：需要 DOM 能力时，Chrome 推荐使用 Offscreen Document；需要持久化时使用 `chrome.storage`（`local/session/sync/managed`）。[1][2][6]
- **MV3 的 SW 注册方式与 Web SW 不同**：不能用 `navigator.serviceWorker.register()`；必须在 `manifest.json` 的 `background.service_worker` 中声明。[3][7]
- **模块化与导入有特定规则**：支持静态 `import`（需 `background.type: "module"`）或 `importScripts()`；不支持动态 `import()`。[3]
- **异步风格正在从回调转向 Promise**：MV3 中许多 `chrome.*` 方法支持返回 Promise（不传 callback 时），但事件监听仍依赖回调；迁移时要避免“一次调用同时用 callback + promise”。[5][15]
- **网络改写从“拦截式”转向“声明式”**：MV3 建议用 `chrome.declarativeNetRequest` 的规则系统替代阻塞式 `webRequest` 监听器（政策安装扩展有例外）。[9][10]

# Detailed Analysis

## 1) “扩展 Service Worker”到底是什么（与 Web SW 的差异）

Chrome 将 MV3 的后台上下文定位为 **extension service worker（扩展服务工作线程）**：它不仅能处理标准 SW 事件，也能响应扩展事件（例如标签页变化、通知点击等），并且注册/更新机制与 Web SW 不同。[2]

关键结论：

- **它是扩展的中心事件处理器**：所有“需要后台做的事”都要围绕事件编排。[2]
- **它不是常驻进程**：默认会反复被终止/重启，这决定了你必须把“状态”从内存搬到持久化层。[1][2]
- **无 DOM**：所有依赖 `window/document` 的库和逻辑需要迁移到其它上下文（如 Offscreen / extension pages / content scripts）。[2][6]

## 2) 生命周期与超时规则（决定你怎么写代码）

官方文档给出的终止条件非常“硬”：[1]

- **30 秒空闲**：30 秒内没有事件触发或扩展 API 调用，SW 会被终止；收到事件或调用扩展 API 会重置计时器。[1]
- **单次处理 5 分钟上限**：某个事件/调用处理超过 5 分钟可能被杀。[1]
- `fetch()` **响应 30 秒**：`fetch()` 等待响应超过 30 秒可能触发终止。[1]

这意味着：

- 不要把 SW 当作“常驻任务/轮询器”；应改为“事件驱动 + 定时点触发（alarms）+ 可恢复”。[1]
- 后台执行长耗时流程要拆分为可中断步骤：每一步写入 checkpoint（如 `chrome.storage`），下一次唤醒继续跑。[1][7]
- 需要持续连接（例如 WebSocket）的场景要注意版本差异：Chrome 116 起活跃 WebSocket 会延长 SW 生命周期，但你仍应容忍连接中断并可重连。[1]

## 3) `manifest.json`：如何声明与模块化组织

### 注册方式

扩展 SW 必须在 manifest 中声明，Chrome 明确指出“不能用网页那套 `navigator.serviceWorker.register()`”。[3][7]

```
{
  "manifest_version": 3,
  "background": {
    "service_worker": "service-worker.js"
  }
}
```

**模块化导入**

- 若用 ES Modules：在 `background` 中设置 `"type": "module"`，然后用静态 `import`。[3][7]
- 或者继续用 `importScripts()`。[3]
- **不支持动态 `import()`**（这一点和很多 Web/Node 心智不同，容易踩坑）。[3]

# 4) “可用 API”与“不可用能力”：把坑一次性列清

## 4.1 DOM / UI 相关能力：用 Offscreen Document 承接

SW 里没有 DOM，这是设计层面的限制。[2] 若你确实需要 DOM API（例如 `DOMParser`、`localStorage`、剪贴板、WebRTC、播放音频等），Chrome 提供 **Offscreen API**：在隐藏页面里运行 DOM 代码，并通过 `chrome.runtime` 消息与 SW 协作。[6]

要点：[6]

- Offscreen Document 必须是扩展包内的静态 HTML。
- 同一 profile 下通常只能开一个（常规 + incognito 分别可一个）。
- Offscreen Document 支持的扩展 API 很少，文档明确提到 **只有 `chrome.runtime`**（所以通信要用它）。[6]

Chrome 迁移 FAQ 也把“需要 DOM 怎么办”作为已知问题，官方答案就是 Offscreen Document。[8]

## 4.2 状态与持久化：不要依赖全局变量/ `localStorage`

- 全局变量随终止而丢失。[1]
- 扩展 SW **没有 Web Storage API（`localStorage`）**。[1]
- 推荐用 `chrome.storage` 存储状态（教程示例也强调用 `chrome.storage.local` 持久化）。[1][7]

实践建议：

- 把“状态”分成：配置（sync/managed）、缓存（local/session）、短期会话态（session）等；并设计版本迁移/回滚策略（onInstalled/update）。[1][7]

### 4.3 事件监听注册：必须顶层同步注册

在 MV3 中，如果你把 `addListener()` 写在异步回调/Promise 里，可能会因为 SW 重启导致监听器没来得及注册，从而错过事件。Chrome 迁移文档明确要求 **同步注册监听器（放到脚本顶层）**。[4]

### 4.4 异步调用：从 callback 迁移到 Promise

Chrome 指出：MV3 里“许多 API 方法会返回 Promise”，并且仍有大量方法同时支持 callback（兼容旧代码），但**一次调用不能既传 callback 又指望返回 Promise**；事件监听仍需 callback。[5]

“What’s new” 也记录了 promise 支持扩展到了更多 API（例如 `chrome.permissions`、`chrome.downloads`、`chrome.debugger` 等）。[15]

对你的架构影响：

- 建议统一采用 `async/await` + Promise 风格（更易在 SW 里组织“短任务 + checkpoint”）。
- 对不支持 Promise 的 API，保留 callback 或自行封装为 Promise（注意 `chrome.runtime.lastError`）。[5]

### 4.5 网络相关：从 `webRequest`（阻塞）到 DNR（声明式）

如果你的扩展在 MV2 用 `chrome.webRequest` 的 **blocking** 模式改写请求，MV3 推荐迁移到 **Declarative Net Request（DNR）**：用规则描述匹配条件与动作（block/redirect/modifyHeaders 等），浏览器在网络栈层执行。[9][10]

关键点：[9][10]

- 迁移不是“换个函数名”，而是按用例重写为规则系统。[9]
- DNR 有配额/规则集限制，并且规则启用状态的持久化语义（跨 session 保留、跨更新不一定保留）。[10]

## 5) 调试与验证：为什么“开 DevTools 会影响行为”

官方教程与调试文档都提醒：在 `chrome://extensions` 里 Inspect/打开 SW DevTools **会唤醒并保持 SW 活跃**，这会掩盖“真实用户环境下 SW 频繁终止”的问题；要测试健壮性，需要关闭 DevTools 再验证。[7][11]

调试入口：[11]

- `chrome://extensions` → 对应扩展 → `Inspect views`（service worker）。
- 也可以在 Application 面板看 SW 状态并手动 start/stop。[11]

## 6) 跨浏览器（WebExtensions）兼容性：不能只看 Chrome

如果你要做跨浏览器扩展（Chrome/Firefox/Safari），MDN 的 manifest `background` 文档提示了现实差异：[12]

- Chrome：MV3 背景只支持 `background.service_worker`。[12]
- Firefox：`background.service_worker` 支持情况不同（MDN 标注长期存在的 bug 追踪），常见做法仍是 `background.scripts/page`。[12]
- Safari：可支持 `service_worker` 与 `scripts/page`，并可用 `preferred_environment` 做选择。[12]

这会影响你的"架构分层"：建议把业务逻辑做成可复用模块，把"背景上下文适配层"单独封装（service worker vs background page）。[12]

# Areas of Consensus

- **MV3 的后台 SW 是事件驱动、会被频繁终止**，需要持久化与可恢复设计。[1][2]
- **SW 无 DOM**，DOM 需求需要迁移到 Offscreen Document 或其它页面上下文。[2][6]
- **监听器要顶层注册**，否则存在丢事件风险。[4]
- **网络拦截能力发生范式变化**：阻塞式 `webRequest`（面向处理器）→ DNR（面向规则）。[9][10]

# Areas of Debate

- **哪些扩展用例"不适合 SW 生命周期模型"**：WebExtensions 社区（W3C）专门收集"不被 SW 很好服务的用例/模式"，反映出生态对"短生命周期后台"仍有争议与诉求。[13]
- **MV3 对内容过滤/隐私扩展能力的影响**：EFF 等组织认为 MV3 的变更会削弱隐私/内容拦截类扩展的能力与创新空间，并质疑其收益；Chrome 官方则强调隐私、安全与性能目标（例如限制远程代码、减少阻塞式拦截）。[14][16]

# Sources

[1] The extension service worker lifecycle — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/concepts/service-workers/lifecycle （官方文档，解释终止条件与生命周期改进；抓取于 2026-02-13）

[2] About extension service workers — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/concepts/service-workers （官方概念文档；抓取于 2026-02-13）

[3] Extension service worker basics — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/concepts/service-workers/basics （官方文档，注册/导入/更新与限制；抓取于 2026-02-13）

[4] Migrate to a service worker — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/migrate/to-service-workers （官方迁移指南，强调顶层同步注册监听器；抓取于 2026-02-13）

[5] Update your code (API calls) — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/migrate/api-calls （官方迁移指南：Promise 支持、替代不支持 API；抓取于 2026-02-13）

[6] chrome.offscreen — API — Chrome for Developers. https://developer.chrome.com/docs/extensions/reference/api/offscreen （官方 API 文档，Offscreen Document 的能力与限制；抓取于 2026-02-13）

[7] Handle events with service workers — Chrome for Developers. https://developer.chrome.com/docs/extensions/get-started/tutorial/service-worker-events （官方教程，示例强调用 `chrome.storage` 持久化、调试注意事项；抓取于 2026-02-13）

[8] Known issues when migrating to Manifest V3 — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/migrate/known-issues （官方 FAQ/已知问题，DOM/远程代码等；抓取于 2026-02-13）

[9] Replace blocking web request listeners — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/migrate/blocking-web-requests （官方迁移指南：webRequest blocking → DNR；抓取于 2026-02-13）

[10] chrome.declarativeNetRequest — Chrome for Developers. https://developer.chrome.com/extensions/declarativeNetRequest （官方 API 文档：规则/配额/与 SW 交互；抓取于 2026-02-13）

[11] Debug extensions — Chrome for Developers. https://developer.chrome.com/docs/extensions/get-started/tutorial/debug （官方调试教程，指出 Inspect 会保持 SW 活跃；抓取于 2026-02-13）

[12] `background` (manifest.json) — MDN WebExtensions. https://developer.mozilla.org/en-US/Add-ons/WebExtensions/manifest.json/background （权威跨浏览器文档，背景上下文与兼容性说明；最后修改 2025-10-13）

[13] Use cases that are not well served by service workers — w3c/webextensions issue #72. https://github.com/w3c/webextensions/issues/72 （标准化社区讨论，反映生态争议与未满足用例；抓

取于 2026-02-13）

[14] What is Manifest V3 — Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3 （官方目标与变更概览；抓取于 2026-02-13）

[15] What's new in Chrome extensions — Chrome for Developers. https://developer.chrome.com/docs/extensions/whats-new （官方更新记录，包含 Promise 支持扩展信息；抓取于 2026-02-13）

[16] Chrome Users Beware: Manifest V3 is Deceitful and Threatening — EFF. https://www.eff.org/deeplinks/2021/12/chrome-users-beware-manifest-v3-deceitful-and-threatening （倡议组织观点，用于"争议"部分对照；抓取于 2026-02-13）

# Gaps and Further Research

- **按 API 维度的"SW 可用/不可用清单"**：Chrome 文档更偏概念与迁移实践，若你需要"某个 `chrome.*` API 在 SW/Offscreen/popup/content script 的可用矩阵"，建议下一步按你的业务 API 列表逐个查 reference 页的"Supported contexts/Promise label"。[5][15]
- **长期运行任务的最佳实践**：官方提供生命周期规则与一些延寿改进，但"后台长任务/流式处理/大规模同步"的通用模式仍在演进，建议结合 W3C 用例收集与 Chrome 版本演进持续更新。[1][13]