

navigator.modelContext (WebMCP / Web Model Context API) 完整规范深度研究报告

- 生成时间：2026-02-13
- 研究目标：把 Chrome / WebML 社区正在推进的 `navigator.modelContext` (WebMCP) 规范与 API surface 完整过一遍，明确“标准里有什么/没有什么”，并标注与常见 demo/实现 (polyfill) 之间的差异，避免我们 TS 适配层遗漏或误配。

Executive Summary

`navigator.modelContext` 是 Web Machine Learning Community Group (WebML CG) 发布的 WebMCP 规范草案所定义的 Web API：网站可将自身能力以“工具 (tool)”形式注册给浏览器，使浏览器侧的 agent (或辅助技术) 能够在用户监督下发起工具调用。[1]

截至 2026-02-12 发布的 WebMCP Draft CG Report，规范在 IDL 层面定义了 `ModelContext` 的最小方法集合：`provideContext()`、`clearContext()`、`registerTool()`、`unregisterTool()`；工具描述体 `ModelContextTool` 由 `name`/`description`/`inputSchema`/`execute`/`annotations` 构成，其中 `execute(input, client)` 返回 `Promise<any>`，并通过 `ModelContextClient.requestUserInteraction()` 在工具执行期间请求用户交互。[1]

需要特别注意：在该版本规范中不存在 `tools()` / `invoke()` 这类“列出工具/主动调用工具”的页面侧 API。若某些 demo 或第三方实现 (polyfill) 暴露了 `listTools()` / `executeTool()` / `toolsChanged` 等接口，它们属于实现扩展或调试接口，不应与标准 API 混淆（适配层需清楚区分“标准 API”与“实现扩展 API”）。[1][2][3]

Key Findings

- **规范状态与稳定性：**WebMCP 是 WebML CG 发布的 Draft Community Group Report，明确“不是 W3C 标准，也不在 W3C 标准轨道上”，仍在演进；当前多个方法的“算法步骤 (method steps)”仍是 TODO，意味着实现/互操作细节可能变化。[1]
- **Navigator 扩展点：**规范以 Web IDL 形式扩展 `Navigator`，新增
[`SecureContext`, `SameObject`] `readonly attribute ModelContext modelContext`，因此它只在安全上下文可用，并要求多次访问返回同一对象引用 (`SameObject` 语义)。[1]
- **核心 API surface (2026-02-12 版本)：**
 - `provideContext(options?: { tools: ModelContextTool[] }) : void`：注册一批工具，并清空任何已有上下文/工具；
 - `clearContext() : void`：清空所有上下文/工具；
 - `registerTool(tool: ModelContextTool) : void`：增量注册单个工具，不清空已有工具；若重名或 `inputSchema` 无效应抛错；
 - `unregisterTool(name: string) : void`：移除已注册工具。[1]
- **工具描述体与回调签名：**

- `ModelContextTool.inputSchema` 是 JSON Schema (规范引用 JSON Schema 2020-12 Core)；
- `execute` 的签名为 `ToolExecuteCallback = Promise<any>(object input, ModelContextClient client)`；
- `ToolAnnotations` 目前只定义了 `readOnlyHint: boolean` (只读提示)。[1][4]
- **与 Prompt API 的关系 (“工具形状”一致性来源)**: WebMCP API proposal 明确将工具 descriptor 设计为与 Prompt API 的 tool use (`{name, description, inputSchema, execute}`) 形态对齐；因此你在 demo 里看到的字段组合大概率来自这套“工具形状”的共识，而不是某个单一实现私货。[2][5]

Detailed Analysis

1) 规范定位：网站“提供工具”，agent“调用工具”

WebMCP 的核心是：网页作为 model context provider，在页面脚本中把工具注册给浏览器；agent 作为 client，通过浏览器介入来触发工具调用，网页在 `execute()` 中实现实际动作并返回结构化结果（可异步）。[2]

在 [proposal.md](#) 中，工具回调可返回 Promise；agent 会在 promise resolve 后收到结果，这为“工具内部可做异步工作（例如等待用户确认或 worker 计算）”提供了规范方向。[2]

2) 标准 API surface：以 2026-02-12 Draft CG Report 为准

在 WebMCP 规范 (IDL) 中，最关键的是两块：

2.1 `Navigator.modelContext`

- 定义：

```
partial interface Navigator { [SecureContext, SameObject] readonly attribute ModelContext modelContext; }
```

- [1]
- 含义：

- `SecureContext`：仅 HTTPS/localhost 等安全上下文；
- `SameObject`：重复访问返回同一实例（适配层不应每次 new）。[1]

2.2 `ModelContext interface`

IDL (简化摘录)：

- `provideContext(optional ModelContextOptions options = {}): void`
- `clearContext(): void`
- `registerTool(ModelContextTool tool): void`
- `unregisterTool(DOMString name): void`。[1]

语义要点：

- `provideContext()` 会“清空并替换”已有工具集（适合 SPA 状态切换）；
- `registerTool()` 是增量注册（不清空）；

- `registerTool()` 明确要求重名与无效 `inputSchema` 抛错（适配层应在本地或 bridge 层对 schema 做校验/归一化，保证与浏览器侧预期一致）。[1]

3) 工具 descriptor 的完整字段： `ModelContextTool` + `ToolAnnotations`

规范定义：

- `name: DOMString` （必填、唯一标识）
- `description: DOMString` （必填、自然语言描述）
- `inputSchema: object` （JSON Schema）
- `execute: ToolExecuteCallback` （必填）
- `annotations?: ToolAnnotations` （可选）
- `readOnlyHint: boolean` （目前唯一 annotation）。[1]

`execute` 回调签名（规范）：

- `ToolExecuteCallback = Promise<any> (object input, ModelContextClient client)`。[1]

这里对 TS 适配层的直接影响：

- `inputSchema` 必须能落到“标准 JSON Schema 对象”语义（至少是 JSON-SCHEMA 2020-12 Core 的兼容子集）；
- `execute` 必须允许 `async/Promise`；
- `client` 参数需被保留（即使你短期不用），因为它是 `requestUserInteraction()` 的入口。[1]

4) 用户交互： `ModelContextClient.requestUserInteraction()`

规范定义：

- `interface ModelContextClient { Promise<any> requestUserInteraction(UserInteractionCallback callback); }`
- `callback UserInteractionCallback = Promise<any> ()`。[1]

解读：

- 这是“工具执行期间”的受控用户交互通道：工具可以在执行中暂停并请求用户确认/输入。
- `callback` 由页面实现（例如弹 `confirm`、打开对话框），浏览器/agent 侧通过这个 API 将“需要用户介入”的阶段显式化；这也符合 WebMCP 的 `human-in-the-loop` 设计目标（proposal 强调用户监督）。[2]

5) 标准里没有什么： `tools()` / `invoke()` / “列出工具”

在 2026-02-12 的规范 IDL 中：

- 没有 `tools()` / `listTools()` 这类“页面侧列出工具”的方法；
- 没有 `invoke()` / `callTool()` 这类“页面侧主动调用工具”的方法；
- 规范聚焦在“注册工具 + 被调用时执行”。[1]

因此，如果你们的 PRD 或 demo 代码里出现：

- `navigator.modelContext.tools()`
- `navigator.modelContext.invoke(...)`

它们更可能来自：

1. 某个 polyfill/SDK 的便捷层（例如为了调试提供 `list/execute`）；或
2. 某个非公开/实验实现的扩展接口。

适配层建议：

- **标准层**只承诺上述 4 个方法 + tool/client 类型；
- **扩展层**（如果需要）用 feature detection (`if ('modelContextTesting' in navigator)` 之类) 按实现分别适配，避免把扩展当成标准能力硬编码。

6) 对“适配层不遗漏”的检查清单 (Phase 0 直接可用)

以规范为硬基线（必须覆盖）：

- `modelContext` 属性：SecureContext + SameObject 语义
- `provideContext({tools})`
- `clearContext()`
- `registerTool(tool)`
- `unregisterTool(name)`
- `ModelContextTool` 字段：`name/description/inputSchema/execute/annotations.readOnlyHint`
- `execute(input, client)` 的 Promise 返回与 `client.requestUserInteraction()`

以 proposal/Prompt API tool use 形状为软基线（建议对齐）：

- `tool descriptor` 结构（至少 `{name, description, inputSchema, execute}`）一致性
- `execute` 返回值形态（规范允许 `any`；proposal 示例中常返回结构化 `content` 数组）[2][5]

Areas of Consensus

- WebMCP 的“标准最小 API”就是围绕 `navigator.modelContext` 注册工具，并在被调用时执行 `execute()`，且支持异步与用户交互请求。[1][2]
- 工具 descriptor 的字段集合与 Prompt API 的 tool use 形态高度一致：`name/description/inputSchema/execute` 是共同核心。[2][5]

Areas of Debate

- **返回值与内容格式的规范化程度：**当前规范让 `execute` 返回 `Promise<any>`，而 proposal 示例倾向返回 `{ content: [...] }` 这种结构化响应；未来是否会把“响应格式”标准化（并与 MCP 更强对齐）仍不确定。[1][2]

- **工具发现/列举 API 是否会进入标准：**规范当前缺少“页面侧列举已注册工具”的标准化能力；实际工程中调试/桥接常需要该能力，未来可能通过新的接口或测试/调试面板补足，但现阶段不宜假设存在。[1][2]

Sources

[1] Web Machine Learning Community Group — *WebMCP Draft Community Group Report* (2026-02-12). (权威：规范草案；明确 IDL/API surface 与 CG 状态) <https://webmachinelearning.github.io/webmcp/>

[2] webmachinelearning/webmcp — *WebMCP API Proposal* ([proposal.md](#)) (2025-08-13). (高可信：规范仓库内的 API proposal，包含动机、示例与异步/交互语义)

<https://raw.githubusercontent.com/webmachinelearning/webmcp/main/docs/proposal.md>

[3] WebMCP-org/npm-packages — `@mcp-b/global` polyfill 源码 (global.ts) . (中可信：实现参考；用于识别“实现扩展/调试接口”的存在及其与标准差异) <https://raw.githubusercontent.com/WebMCP-org/npm-packages/main/packages/global/src/global.ts>

[4] JSON Schema — *JSON Schema Core* 2020-12 (Draft 2020-12). (权威：规范引用的 inputSchema 语义基础)
<https://json-schema.org/draft/2020-12/json-schema-core.html>

[5] webmachinelearning/prompt-api — *Explainer for the Prompt API* (Tool use section) . (高可信：Chrome built-in AI team 的 explainer；说明 tool descriptor 形状与 execute 回调)
<https://raw.githubusercontent.com/webmachinelearning/prompt-api/main/README.md>

Gaps and Further Research

- **原生 Chrome 实现的“非标准调试接口”清单：**规范未定义列举/执行工具的调试 API，但 polyfill 与部分实现可能提供（例如 `modelContextTesting`）。建议用真实 Chrome 版本（你们目标版本）在 `MAIN` world 中枚举 `navigator` 上的相关属性并截图留档，形成“适配层 feature matrix”。[1][3]
- **响应格式与错误模型：** `execute` 返回 `any` 太宽松，适配层应先决定内部标准（比如统一成 MCP `CallToolResult` 样式），并验证在各类实现（原生/polyfill）下的兼容性与失败行为。[1][2]