AG-UI JavaScript/TypeScript Core SDK（@ag-ui/core）详解

你提供的链接 **https://docs.ag-ui.com/sdk/js/core/overview** 是 AG-UI 协议（Agent-User Interaction Protocol，代理-用户交互协议）的官方文档页面，专门介绍其 **JavaScript/TypeScript Core SDK**。

这是什么？

- **包名**：@ag-ui/core
- **类型**：纯 TypeScript SDK（提供强类型支持）
- **所属项目**：AG-UI 协议的官方 JavaScript/TypeScript 实现部分
- **GitHub 仓库**：https://github.com/ag-ui-protocol/ag-ui （AG-UI 协议主仓库）

主要用途

这个 SDK 是 AG-UI 协议在前端（JavaScript/TypeScript）环境的**核心基础库**，用于帮助开发者在浏览器或 Node.js 中实现与后端 AI Agent 的标准化、实时交互。

具体来说，它提供：

- **流式事件驱动架构**：代理后端通过 HTTP/SSE（Server-Sent Events）或可选二进制通道发送一系列 JSON 事件，前端实时监听和处理这些事件，实现真正的“Agentic”体验（如实时显示思考过程、状态更新、工具调用进度）。
- **强类型数据结构**：所有事件、消息、状态、工具等都用 TypeScript 类型定义，确保开发时类型安全、减少运行时错误。
- **标准化通信**：桥接后端 Agent（如 LangGraph、OpenAI、Ollama、自定义 Agent）和前端 UI，避免自定义 WebSocket 或乱七八糟的 JSON 解析。

它不是一个“开箱即用”的 UI 组件库（如 CopilotKit 的 @copilotkit/react-ui），而是**更底层、更通用**的核心工具，适合：

- 需要深度自定义 Agent 交互的前端开发者。
- 非 React 框架（如 Vue、Angular、Svelte 或纯 JS）。
- 构建自己的 Agent 客户端，或与其他 AG-UI 子包组合使用。

关键特性

- **完整事件体系**：支持生命周期事件、文本消息流、工具调用、状态管理（STATE_DELTA Patch）、自定义事件等（总共 17+ 种事件类型）。
- **模块化设计**：@ag-ui/core 只负责核心类型和事件定义，其他功能通过子包扩展：
  - @ag-ui/client：实际客户端连接（HTTP/SSE 支持、中间件、运行 Agent）。
  - @ag-ui/encoder：事件编码/解码。
  - @ag-ui/proto：Protobuf 二进制支持（可选高性能模式）。
- **与 CopilotKit 的关系**：CopilotKit 的 React UI 包（如 useAgent Hook）底层就是基于 AG-UI 协议和这些 SDK 实现的，但 @ag-ui/core 更独立、更底层，不依赖 React。

安装与基本使用

安装：

Bash

```
npm install @ag-ui/core
#               client
```

```
npm install @ag-ui/client
```

基本使用（结合 @ag-ui/client 示例，核心类型来自 @ag-ui/core）：

TypeScript

```
import { HttpAgent } from "@ag-ui/client";  // client

const agent = new HttpAgent({
  url: "https://your-ag-ui-endpoint.com/api/agent",  //         AG-UI
});

const result = await agent.runAgent({
  messages: [{ role: "user", content: "          " }],
  //     @ag-ui/core           Message  Context  Tool
});

//
for await (const event of agent.stream()) {
  switch (event.type) {
    case "TEXT_MESSAGE_CONTENT":
      console.log("       :", event.delta);
      break;
    case "STATE_DELTA":
      //     JSON Patch
      applyPatch(localState, event.patch);
      break;
    // ...
  }
}
```

页面上没有完整代码示例，但强调通过类型（如 RunAgentInput、Message、Tool、State）和事件参考来开发。

与 CopilotKit 的区别

- CopilotKit（@copilotkit/react-*）是**高层封装**：提供现成 React 组件（聊天窗、侧边栏）和 Hook，直接用几行代码就能出效果。
- @ag-ui/core 是**协议级底层 SDK**：更灵活、框架无关，适合自定义实现或非 React 项目，但需要自己处理 UI 渲染和事件逻辑。

如果你想在 React 中用 AG-UI，推荐先用 CopilotKit；如果需要跨框架或极致控制，才直接用这个 Core SDK。

官方文档还在完善中（当前 Overview 页面较简略），建议继续查看：

- Types 参考：https://docs.ag-ui.com/sdk/js/core/types
- Events 参考：https://docs.ag-ui.com/sdk/js/core/events
- Client 子包：https://docs.ag-ui.com/sdk/js/client

有具体想实现的功能（如自定义事件处理），我可以帮你写更详细的代码示例！🚀

是的，你的 **React Native** 项目**可以用 AG-UI 协议**，但目前（2025 年 12 月）**没有官方或现成的 React Native SDK/客户端**，需要手动实现或等待社区贡献。

当前状态

- **JS/TS Core SDK**（@ag-ui/core）是纯类型定义和事件结构，主要针对 Web 环境（浏览器）。React Native 是非 DOM 环境，可能需要 polyfill（如 fetch、EventSource 用于 SSE），但核心逻辑（如事件处理、JSON Patch）可以运行。
- AG-UI 官方仓库明确标记 **React Native 集成** 为 “🛠️ Help Wanted”（社区求助贡献），有一个开放 Issue（#510，2025 年 10 月提出）：请求添加 React Native 客户端，支持原生组件渲染 AG-UI 事件、实时流式连接等。目前尚未实现。
- CopilotKit（AG-UI 的主要推动者）也主要支持 Web React，没有官方 React Native 包。

可行方案（推荐优先级）

1. **手动实现 AG-UI 客户端（最实际，目前可行）**
   - 使用 @ag-ui/core（类型）和 @ag-ui/client（如果兼容）作为基础。
   - 在 React Native 中用 fetch 或第三方库（如 react-native-sse、event-source-polyfill）实现 HTTP/SSE 连接到你的 AG-UI 后端端点。
   - 手动监听事件流（TEXT_MESSAGE_CONTENT、STATE_DELTA 等），用 React Native 组件（如 Gifted Chat、FlatList）渲染消息、状态更新、工具调用进度。
   - 支持 Human-in-the-Loop：事件到来时弹出原生 Alert 或 Modal 确认。
   - 优势：完整保留 AG-UI 所有特性（实时状态同步、生成式 UI 等），性能好。
   - 难度：中等，需要自己写事件解析逻辑（参考 Web 版的 CopilotKit useAgent Hook）。
2. **等待或贡献官方 React Native 客户端**
   - 可以 star/watch GitHub Issue #510，或自己贡献（社区欢迎）。
   - 一旦实现，就能像 Web 版一样几行代码接入。
3. **替代移动方案：Kotlin Multiplatform SDK（纯原生）**
   - AG-UI 已于 2025 年 11 月推出 **Kotlin SDK**（社区贡献），支持 Android、iOS、JVM 原生环境，无 JS 桥接。
   - 如果你的项目允许混合原生代码，可以用 Jetpack Compose (Android) + SwiftUI (iOS) 构建 UI，直接用 Kotlin SDK 连接 AG-UI。
   - 这不是 React Native，但如果是新项目或可重构，体验更好（原生性能）。

总结建议

你的 React Native 项目**完全可以用 AG-UI 协议**（后端不变，前端手动消费事件流），这也是协议设计的初衷——框架无关、跨平台。短期内手动实现是最快的路径，能快速获得实时 Agent 交互（如思考过程、状态更新）。

如果你提供更多细节（如想实现聊天 + 状态同步的具体场景），我可以帮你写一个 React Native 的基础代码模板（基于 fetch + SSE）！🚀