

前端智能化实战-WebMCP- WebSkills-WebAgent-01_原文

2026年06月14日 14:20



00:07

各位屏幕前的开发者朋友们，大家晚上好。欢迎来到本次线上技术直播，我是今天的主讲人之一李锦浩，首先非常感谢大家在忙碌的工作和学习中，抽出时间来参与咱们这场直播。无论是刚进入直播间的同学，还是提前预约守候的老朋友，都先给大家道一声感谢。也麻烦大家动手给直播间点个赞，加个关注，后续我们还会持续分享更多实用的技术干货。

00:49

今天要跟大家聊的主题是前端智能最佳实践，重点围绕三个关键词展开。VMCP我have skills和will agent。简单跟大家说一下今天的直播流程。首先由我来简单介绍，为准，然后郑志超同学来讲解VMCPVF skill以及完整的实践操作流程。大家有任何的疑问都可以在弹幕里留言，我会在讲解结束之后，专门留几分钟时间集中解答大家的问题。另外的话直播过程中会有小互动和福利。我们可以针对优秀提问的开发者，抽取三位朋友送出我们的周边礼品。大家全程跟唱，肌肉不要走开，话不多说，让我们开始今天的第一部分人的介绍。

01:51

首先微微枕我们分为四部分来进行介绍。第一部分是讲解微枕是什么。然后第二部分是讲解为准的一个核心工作机制。第三部分的话是主要强调一下它的适用场景，以及说哪些场景它可能不适用。第四部分的话简单讲解一下怎么快速上手。



02:21

好，我们进入第一部分认识为准。这部分可以回答三个问题。Vision是什么？然后它在整个架构中处于什么位置？它和我们熟悉的传统代理有什么区别？



02:39

在开始讲region之前，先快速过一下MCP协议。MCP全称是model context protocol，是and stopped提出的一个开放协议，它定义了AI模型，以我我以A准就是外部交互的一个叫方式，简单来说MCP解决的是AI怎么调用外部能力。然后这个核心问题如图所示，它完整的一个链路是AI或者说LM，然后通过MCP client发起请求，然后我们MCP server接收后调用具体的to或者是resource，开和MCP之间使用的是JSRVC2.0进行双向通信。

MCP 协议概览

Model Context Protocol (MCP) 是 Anthropic 提出的开放协议，定义了 AI 模型 / Agent 与外部工具、资源的标准化交互方式。



MCP 解决了 "AI 怎么调用外部能力" 的问题

03:30

理解了MCP之后，我们就能更好的理解为准在其中扮演的角色。那标准是什么呢？我们先用排除法来理解，它不是一个AI模型，它本身不做推理以及角色。然后它也不是一个HTTP反向代理，它不是简单的转发一个网络流量，它同时也不是一个消息队列，它不做消息的持久化以及削峰。它是什么呢？它是一个MCP协议的一个中中间枢纽服务，它的核心职责是在AI以及业务应用之间架起一座桥梁，然后通过规划管理以及消息转发，让AI能够通过标准的MCP协议来驱动和操控这个web应用。这是一个开源项目，大家可以在get go上面去查看。

定义

WebAgent 是一个开源的 **MCP 代理转发枢纽服务**，它让 AI 智能体 (Agent) 能够通过 **MCP 协议** 驱动和操控 Web 应用。

不是什么

- 不是一个 AI 模型
- 不是一个 HTTP 反向代理
- 不是一个消息队列

是什么

- MCP 协议的中间枢纽
- Agent ↔ App 的桥梁
- 会话管理 + 消息转发



04:37

这张图展示了一个reagent的一个核心定位，它是一个三层架构的中间层。然后最上层是我们的AI控制端，我们称为称它为remote，它可以是这些cover GPT，或者说我们制定的age，或者说我们我们demo从未展示的一个tiny remote。然后最下面是一个业务的应用端，也就是集成了open tiny next GSDK的web端，或者说桌面，或者说我们的手机端web agent，它就在我们的中间。负责规划管理、消息转发以及协议的翻译。注意到上下这两条连线它都标注了一个MCP协议，这也就意味着VA准对上以及对下它都是通过VNCP来进行交流通性的。它不是简单的把详细传递过去，我们后面可以详细的进行讲解。



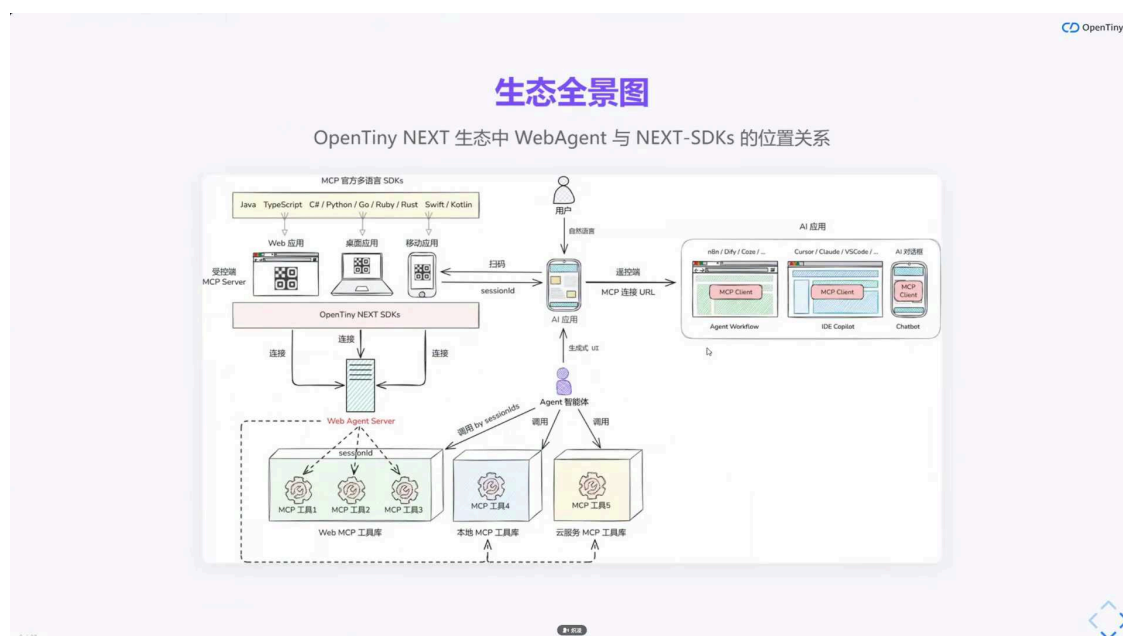
05:47

然后我们讲reagon它提供四大核心能力。第一个就是MCP代理转发，它支持stream OHTP以及SSE的一些传输模式，完整的代理MCP的一个协议消息。第二个就是它的规划管理，每个连接的业务应用以及控制端，它都有独立的规划，支持远程控制以及调试。第三个就是多协议的支持，还支持虚拟网

络TPSSCE以及web service 3种传输层，然后适配不同的部署场景以及网络环境。第四个的话就是生产就绪，我们开源的版本，给大家内置很很好的一个后面安全头健康检查以及一些温室预制PMR doctor部署的一些知识。它不是一个demo项目，它是可以直接上生产。

06:53

然后我们来看看open reagent在整个open tiny next生态中的一个位置。左边是open time SDK，然后它是集成到业务应用中，然后他负责去注册MMCP工具，暴露可以操控的能力。然后中间就是我们的一个vision，然后它作为一个代理的枢纽服务，然后他会去管理，去规划以及转发消息。右边是我们的open tiny next remote，提供操控端的一个UI体验。连接到reagon后，他去向业务应用发送指令。这三个组件它协同工作，构成了一个完整的AI驱动的我运用的一个基础设施。然后注意箭头方向SDK主动连接为准，然后remote它也是主动去连接region version是被动的一个枢纽的一个角色。然后这是我们over生态的一个完整的全景图，可以看到我们v agent它充当是一个桥梁的角色。



08:07

上边这边是我们的一些生态，它一些受控端就是它可以是微应用，可以是桌面应用，也可以是移动应用。然后它通过连接我们的LH注册，可以操控的能力。然后右边是我们的一个AI应用，也就是它可以是MRA，然后可以是A它可以是格式cloud，可以是一些对话框。然后它通过连接我们的一个region，拿到可以操控的一个工具能力。当然他需要提前去扫码，去拿到筛选ID参与能力，拿到他们的操控能力。

08:54

好，这张表它对比了我们v agent以及传统HTTP反向代理的一个差异。在协议层面engines处理的是HTTP以及TCP，我们的region用的是MCP协议，是基于至少RPC2.0。在功能层面，传统那里做的是流量转发以及负载均衡。然后我们为准做的是规划管理，消息路由以及协议翻译。在连接模式上，存储那里是请求响应模式，微vision支持SS异常连接以及string CTTPI以及微缩体等连接方式。最关键的一个区别能力是感知能力，传统的HTP反向代理他不感知业务的逻辑，我们的version它是感知MCP的方法，比如说突破resource read，然后最后是一个角色定位。传统代理是透明的，而我们的v region采用一种独特的孪生客户端的模式，我们后面会进行接送。然后下面是查看我们的一个技术站，它的底层是1 express 5.0，然后语言是我们是tree，核心是依赖的是我们的open tiny agent，它是MCP代理的核心库，它封装了一个我们的软件客户端，消息路由等等的核心逻辑，安全方面的话是我们还美，然后一直用了weston部署支持PM2以及dog。值得一提的是我们的no node环境它依赖是22以上，因为项目用到了一些比较新的no GS特性，整个整个依赖的话非常精简。

OpenTiny

与传统代理的区别

	HTTP 反向代理 (Nginx / Kong)	WebAgent
协议	HTTP / TCP	MCP 协议 (基于 JSON-RPC 2.0)
功能	流量转发 / 负载均衡	会话管理 + 消息路由 + 协议翻译
连接模式	请求-响应	SSE 长连接 / Streamable HTTP / WebSocket
感知能力	不感知业务逻辑	感知 MCP 方法 (tools/call 等)
角色	透明代理	孪生客户端 (Twin Client) 模式

© 2024 开放原子

» 回到顶部

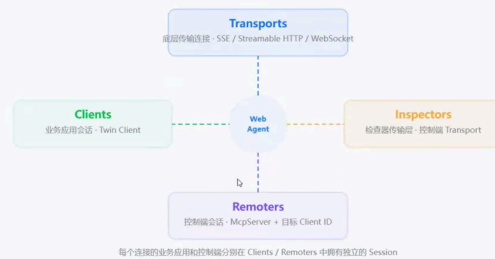


10:59

接下来就我们第二部分，也就是今天相对硬核的部分，然后是它的vision的核心工作机制。我会逐步拆解完整内部的引擎原理，包括四层规划模型，软件客户端模式、双向消息转发，小等关键时机。理解了这些机制，你就能去明白为什么它不是一个简单的消息转化器，而是一个精心设计世纪的协议代理。

四层会话模型

内部维护四组会话注册表



11:34

微软内部它维护了四组会话注册表，构成了它的四层会话模型。底层是它的transport，它负责底层一个传输连接SSE/HTTP或者是websocket，他负责实际的一个消息流的传输，就是我们字节传输。第二层是我们的client，是我们的业务应用规划。每个接入的业务应用在这里，它都有一个孪生clients例。第三层是我们的remote，就是我们的控制端。每个连入的AI仓库控制端在这里都有一个MCP是我实例。第四层是我们的inspect，就是它的检查机构传输层用于控制端的一个transport管理，它这四层的话各司其职，transport负责管连接client管被控端、remote管控制端、inspect管控制端的一个底层传输。每个GU的应用它都有独立的筛选，互不干扰。

12:48

然后我们讲到我们这是vision的核心创新部分，就是它的孪生客户端。这个设计解决了一个关键问题，就是完全不能像engines那样简单做HTTP请求转发。因为他的MCP是有状态的，我们需要维护完整的协议握手以及能力声明。他的做法就是说。



13:14

当我们一个业务应用通过SIC或者是able HTTP连入v agent的时候，v agent它不是简单的记录这个连接，而是为这个应用创建一完整的一个MCP开恩实例。这个看实例就叫做孪生客户端。它和业务应用端是一个对称的，也就是他们是镜像的应用端。因为他自己连接的是一个真正的MCP server。而实际上vision内部的软件客户端在代理它的所有的交互，关键点在于软件客户端声明了所有的MCP的能力to resource promote sampling。同样应用认为自己连上了一个全功能MCP是我，但实际上tim client他并没有真正处理的这些请求，而是把他们转发给了远端的一个控制方，这就是协助传而本而非本地代理。



14:21

然后均开的它的初始化分为三部分，我们把它归叫做为注入初始化。第一部分的话是清空原有的处理器MCP SDK会默认会注册一些请求以及通知感的处理器。软件客户端首先会把这些全部给移除，为自定义的一个逻辑腾出一些空间。第二部分的话注入自定义响应这一部分的话非常关键，孪生客户端可以反馈一个全能力声明，告诉连接的业务应用。我支持to resource, promotes sampling education，它的等等

所有的能力自建应用。因为自己脸上的一个功能完整的MCP事务，第三部分就是挂载监听器系统，未请求响应通知分别注册监听系列。这里有个重要的事情就是说我们支持多个remote同时挂载到同一个的软件客户端上。也就是说多个AIA准，它可以同时观察和操控同一个业务应用。

15:34

三步完成之后，孪生客户端就准备好了，等待remote连入之后建立一个小技巧。当我们mote连入之后，微微信还会建立一个双向消息转发桥。正向的流程很直观，就是说我们remote去发起请求，有to cos不list，比如说的等等，请求代理NCP server去拦截之后进行一个路由分发，然后最终由我们孪生客户端转发给业务应用。然后业务应用处理完毕之后，沿着反方向，这里是反反方向进行传回，然后这个桥是双向的，就是说有个反向的通道，业务应用也可以也主动向remote进行发起请求。典型的比如说我们NCP的一个录的例子，就是说AI请求跟资源列表。还有比如说我们的sampling career message，让AI生成一些消息。这种双向能力意味着为准它不仅是一个AI控制应用的单向管道，而是一个真正的一个协议级的双向桥梁。





16:59

我们认为是完整的突破调用来串联整个流程。首先是第一步，就是我们remote，就是AI agent发起一个出口请求。然后第二步的话是我们的MCP server，进行一个内部代理拦截这个请求。然后第三步它会根据我们的方法名称进行一个路由分发，它识别出这是一个突破。然后第四步是我们路由对应到对应的一个软件客户端进行一个真实的一个突破请求。

17:41

然后我们孪生客户端去把这个请求转发给实际的业务应用。然后我们业务应用去进行一个，工具的一个逻辑的执行。执行完成之后，它的执行结果也是通过transport去回传到孪生客户端。孪生客户端会把这个，标准的一个反馈结果封装一个封装成一个追踪RPC响应。然后最后的话我们的remote就收到一个突破的一个请求。

18:14

整个过程来说整个过程对于moto来说它是一个完全透明的。他以为自己直接和直接在和业务进行业务通信，但实际上VAR在中间做了完整的一个协议代理。这里的关键设计就是MCP的请求处理器被替换了，不再进行一个本地的处理，而是路由给软件客户端的一个对应方法。



18:43

我们agent支持两种连接模式，分别是对应被控端和控制端。左边是我们的应用接入模式，它不会带系统ID接入为准，这个时候region会创建一个transport，然后创建一个ration客户端，train clients，然后注入处理器以及完成链接，最后存入到开一个注册表。然后这个流程就是就是说我是一个被控制的应用。然后右侧是我们的远程控制，也就是控制端的一个接入，然后AI准或者是控制端，它带这个session ID，也就是说他想要操控的一个应用。然后还会进行查找对应的一个train kinds，为这个remote建立一个MCP，so建立一个双向消息桥，最后把它存入到一个remote一个注册表。区分这两种模式的关键就是说筛选ID你不带筛选ID连入，你就是一个作为一个应用，就是一个被控端进行记录。你带的筛选ID你就是远程控制这个设计，它非常优雅，就同一个APA端点，通过有没有筛选ID来区分这种个角色。

20:10

我们为准支持三种传输层。我们来看它的差异，一个是sse模式，通过get的请求来建立长连接，通过post请求来发送消息，规划用122的center ID来进行标识，然后浏览器兼容性最好，但是它现在已经被废弃了。然后我们现在推荐的是我们的新微博TTTP方式，然后我们每次post请求都会带一个MCP筛选ID的一个请求头。然后这就是我们的收集连接方式，它是全双工的一个持久连接，延迟坠机，它适合需要频繁双向通信的一个场景。最重要的一点是这三种传输层它其实是共享同一套郡开一家消息转发机制，也就是说你不管用哪一种方式，连入上层的业务逻辑它是完全一样的。传输层的选择只是影响它连接的建立方式，它不影响功能。

CO OpenTiny

三种传输层

	SSE	Streamable HTTP	WebSocket
连接方式	GET 建立长连接 POST 发送消息	每次 POST 携带 mcp-session-id Header	全双工持久连接
会话标识	URL 中的 sessionId 参数	mcp-session-id HTTP Header	URL query 中的 sessionId
初始化触发	GET /sse 时 立即创建	首次 POST 中 检测 initialize 请求	WebSocket 升级 时创建
适用场景	浏览器端 兼容性最好	无状态 HTTP 环境 (Serverless 友好)	低延迟 双向通信

三种传输层共享同一套 Twin Client + 消息转发机制，仅连接建立方式不同

CO OpenTiny

全局流程总览



21:21

这是我们全局流程的一个总览图，就是把前面我们讲的所有概念串在一起。左边是三个业务应用，分别用sse然后able HTTP以及will socket连入我连入到我们的v agent，中间的version内部，它每一个应用对应分别对应一个train kins，然后每一个控制端对应一个MCP server还有一个消息路由层，进行一个消息的分发。右边是三个控制端，2个AIA准。它分别去操控应用A以及应用B然后一个NCP in Better，它是作为一个调试工具核心的匹配，它的机制就是session ID每个twin can，它有唯一的一个session ID就是这twin开remote通过指定section ID来连接到待测定的一个应用双向消息桥。它确保所有的请求以及响应都能准确路由到目标上。这张图就是我们没准在运行时它完整的一个流程，一个总览图。

22:46

好，了解了核心机制之后，我们来聊聊它的适用场景。任何技术工具都有它的那个铁皮区，也有它不擅长的一个区域，正确使用为准。它的前提是啊我们理解它适合做什么，而不适合做什么。



23:03

也就是说它适合场景，它有这么几类。第一种就是AI驱动外应用，这是最典型的场景。用户通过自然语言让AI来操控前端界面，填写表单，展示数据等等。然后第二种就是企业agent基础设施之类，一个多租户的环境下它可以作为集中式的MCP代理枢纽，统一管理所有应用以及AI的连接。第三种就是我们远程控制与监控，运维人员与AI可以远程操控应用状态，读取运行时数据，非常适合运营以及监控的场景。第四种就是我们的低代码无代码的一种平台，一种驱动的自动化流程编排，将AI替代人进行完成拖出来配置等等操作。第五种就是我的开发调试，在开发结论，通过控制端实时调试MCP工具的注册以及调用，大大提升我们的开发效率。



24:09

同样重要的是我们了解它有哪些不适合的场景。如果你只是想要简单的一些HTTP代理，它不是不涉及到MCP协议，那你用engines它就够了，没必要去引入aviation。如果你的场景是高频消息的队列，你需要一些吞吐量。那为准的优化方向就是说MCP绘画管理它不是吞吐量，所以说它也不是很，你可能要

考虑用的是the former cos是啊rubi MQ，如果你是纯同步的一些批处理，它也不合适就是为准以流式以及sse连接为核心试剂，它不是批量的一些同步应用处理器。最后的话，如果你的A准不支持，MMCP协议就没办法直接使用，为准，需要让A准先接入一个MCP。

CO OpenTiny

不适合的场景

✗简单 HTTP 代理

— 如果不需要 MCP 协议，用 Nginx 即可

✗高频消息队列

— 优化方向是 MCP 会话管理，不是高吞吐量

✗纯同步批处理

— 以流式 / SSE 长连接为核心设计

✗无 MCP 的 Agent

— 依赖 MCP 协议，非 MCP 的 Agent 无法直接使用

25:22

我们我们简单举例一个场景，就是AI操控电商后台。就是假设我们运营人员对AI说，帮我把sku 1234的价格调整为二级救援。AI理解意图之后发起一个突破请求，调用个update Price的工具传入sku以及Price参数这个请求到达region version，查找目标应用的train clients，把请求发过去把请求发过去之后，电商后台他收到一个请求，执行一个after Price的工具逻辑，修改数据库中的价格，然后反馈操作结果。整个流程中，运营人他只需要说一句话，就是AI通过MCP协议和微卷完成了原本需要手动操作的任务，这就是前端智能化的价值所在。

CO OpenTiny

场景举例：AI 操控电商后台

AI Agent
「帮我把 SKU-1234 的价格调整为 299 元」

↓ tools/call: updatePrice({ sku, price })

WebAgent
查找目标应用的 Twin Client → 转发请求

↓ callTool({ name: "updatePrice", args })

电商后台 (Vue + @opentiny/next-sdk)
执行 updatePrice 工具 → 修改数据库 → 返回结果

26:15

最后我们简单看一下快速上手，就是啊第一步你可能是需要先安装一个类似SDK。然后第二步的话就是这是前端应用接入的部分，你可能是需要去创建一个VMCP clients，在这里创建一个VMCP clients，然后去连接到MYV agent后端，支持后v agent后端的一个URL。然后第三步就是在子页面为MCP server中进行注册，MCP工具，每个页面都可以定义自己的VMCP。So页面切换的时候，它会也会进行一个切换。然后这里的例子是注了注册了一个demo two的一个工具，使用了js chema进行定义输入参数。当调用这个工具的时候，它会执行回调函数，然后反馈结果，最后可能是我签署。



STEP 1: 安装 NEXT-SDKS

```
npm i @opentiny/next-sdk
```

NEXT-SDKS 是什么

前端智能应用开发工具包，简化 WebAgent 的集成与使用











STEP 2: 创建 WEBMCPCLIENT

```
// App.vue — 在主入口定义 WebMcpClient
import { onMounted, provide } from 'vue'
import {
  WebMcpClient,
  createMessageChannelPairTransport
} from '@opentiny/next-sdk'

onMounted(async () => {
  const [serverTransport, clientTransport] =
    createMessageChannelPairTransport()
  provide('serverTransport', serverTransport)

  const client = new WebMcpClient()
  await client.connect(clientTransport)

  const { sessionId } = await client.connect({
    agent: true,
    url: 'https://agent.opentiny.design/api/v1/webmcp-trial/mcp'
  })
})
```





OpenTiny

STEP 3: 创建 WEBMCPSERVER

```
// views/page1.vue — 在子页面定义 WebMcpServer
import { onMounted, inject } from 'vue'
import { WebMcpServer, z } from '@opentiny/text-sdk'

onMounted(async () => {
  const serverTransport = inject('serverTransport')
  const server = new WebMcpServer({
    name: 'mcp-server-page1',
    version: '1.0.0'
  })

  server.registerTool({
    name: 'demo-tool',
    {
      title: '演示工具',
      description: '一个简单工具',
      inputSchema: { foo: z.string() }
    },
    async (params) => {
      return {
        content: [{ type: 'text', text: `收到: ${params.foo}` }]
      }
    }
  })

  await server.connect(serverTransport)
})
```

OpenTiny

27:24

然后第四步就是我们遥控器的这里去引入我们的remote。然后它会点击右上角一个图标，它就是遥控器的入口，你就可以弹出一个AI对话框进行直接对话。也可以说弹出二维码让手机进行扫码，无论哪种方式，它都可以通过自然语言让AI替你操作前端应用，到这里四步就可以完成前端应用的一个智能化改造。然后具体的一个操作示例，待会有郑志浩同学来替我们完成这个演示。

OpenTiny

核心 API 速查

端点	方法	功能
/api/v1/webmcp/mcp	ALL	Streamable HTTP 端点 (推荐)
/api/v1/webmcp/sse	GET	SSE 连接
/api/v1/webmcp/messages	POST	SSE 消息转发
/api/v1/webmcp/list	GET	所有客户端会话
/api/v1/webmcp/remoter	GET	所有控制端会话
/api/v1/webmcp/tools	GET	客户端注册的工具列表
/api/v1/webmcp/ping	GET	健康检查 + 会话清理
/api/v1/webmcp/reset	GET	重置所有连接

OpenTiny

28:00

然后我这里就是给出一些核心API的一个素材，这是MCP，API跟V一跟MY，MCP跟MCP就是我们新锐部HTTP的端点，推荐去优先使用SIC以及message，就是SAC模式的连接以及消息端点。如果是历史以及remote，就可以分别查看我们的一个客户端以及控制端的一个规划，这在第八的时候它会非常有用。Ping的话它可以做一些健康检查以及管理。Reset的话它会它可以重置所有的连接，非常之前一般都是restful风格的，非常直观。



28:48

都是部署，然后我们提供三种方式的部署。一个是PM2部署，然后构建后你可以使用PM2来启动，支持自动重启，内存预是111GB的告警，有氧停机。然后第二种就是多可部署，它适合一些容器化的环境。第三种是engines加PM2。然后这种需要有一些特别注意的，就是说因为SAC需要长连接，所以engine必须要关闭一个post buffering，以及设计一个足够长的process read汤帽。因为SAC需要长连接，不然的话它会被engine超时断开，造成无法连接的一个问题。

29:39

然后是我们这是reagent部分的一个总结。首先我们介绍了region是什么，还是一个MCP代理协议的枢纽，让AI agent进行驱动。微应用它的核心机制就是我们的twin Carries以及双向消息转发加一个多层组成，实现了透明的协议代理。我能不能直接用呢？是可以的，我们是开箱即用的。然后项目当前已经在D扣上面开源仓库地址是这个。然后欢迎是欢迎大家在机构上面进行提出一些issue，或者是我以及共建代码，谢谢大家。然后。我们来看一下，对于针对vient这一部分有没有一些疑问，看看有没有同学发言。



31:00

我们来看一下后台有没有人留言提问。好，我们这里看到一条疑问，就是说对时延极高的场景它合适吗？其实这个不太合适，因为它本身对于这些消息转发，它本身是需要对一个大模型进行一个处理。然后大模型它处理进行推理，它本身它就是需要一个时间比较长的，所以他不太合适对这种实验高的要求进行去接入。他可能要他适合的场景是一些比较复杂或者重复的一些流程，他他的逻辑比较复杂。

32:01

然后我们来看，第二个问题是这个智能前端工程本身改造，可以把这个MCP服务给别人调用吗？这个是可以的。就是说他注册之后，就是我们前端进行智能化改造之后，他可以进行一个连接，连接到我们为准。或者说待会郑志涛同学会介绍那个VNCP，它也可以不连接到为准。如果我连接到vision的话，它会有一个唯一的一个session ID，然后你拿到一个筛选ID就可以给别人去进行一个连接，去遥控。所以说这个类似一个授权码，你拿着这个授权码，你就可以别人给别人遥控当前这个应用。

33:11

然后我们来再看一个问题，我们的portal部署在OPS上，这个适合使用MCP吗？这个问题是没有没有太理解，就说你不出在我不太清楚这个OPS是什么环节。就是说你如果说前端接入MMCP的一个工具，它可能适合的场景，待会正式商务同学应该也会做一些强调以及做一些进一步的澄清。就是说他你你适合的场景就是智能化的一些改造，你你你可能试试去接入一个AI center，待会这正超郑超同学应该也会有介绍到。然后以及说被操控，就是说我们可以通过自然语言去完成。原本我们在页面上点点点完成很多步骤才能进行一个操控的步骤。现在我们只需要通过一句话就可以完成这一系列的逻辑。这个在新的一些AI交互体验上是一个非常效率提高非常明显的，是一个非常好的一个体验。

34:42

好，我们再再看最后一个，就是一个JUSDK有什么关系？GUSDK是我们也是我们open tiny的一个生态产品，它的主打的就是一个流式渲染，就是流式渲染它的一个应用以及回荡。他他这个跟跟我们是跟我们的vision以及他是不冲突的。好啊，这这里这里我们先暂时回答到这里。

35:16

然后下面的话就是我们的第二部分，由我们的曾世涛同学带来YMCPY skill的介绍以及完整的一个操作演示。好，行，那我今天我再介绍一下。大家好，欢迎参与这本次的分享。今天我主要说的是企业级的应用智能化方案。相比前几天我也进行过几期，大概两期分享吧。



35:50

就这期分享给大家带来的一个新的比较大特性的就是我们不再去重新定义YYMC，而是直接从实现这个叫什么浏览器标准到我们去遵循浏览器标准。然后我们直接使用谷歌浏览器，它的最新的版本146版本。但是需要在你打开那个设置，把那两个设置选项打开，你才能用它内置的这个YMMCP。然后我们现在提供的一条路就是你直接使用谷歌浏览器内置的这个外边CP的API。然后我们会提供一个Polly view，一个抹平层给你。然后你就直接使用它的语法去构建，就再加上我们的web q，再加上外表层构建一个企业级的一个智能化的一个方案。Web SQ就不多说，就跟业界的sq一样外部性的。就刚才景浩他分享的一个后台代理服务，这个是可以让你的外部应用可以只要别人拿到你的在线ID就可以操控你的应用。但这个三星ID可不能随便传给人，因为这个可能涉及到一些安全问题。

37:02

好，然后我们接下来本期的整体演讲大概分为七个部分。第一个我们要说明一下，现在的这些应用为什么要智能化。然后它智能化后面有后后会有哪些好很好的一个用途，然后它的体它的价值在哪里？然后第二个是带来一个情景在线，这个情景在线的主要是说明我们这个外部MCP它的一个使用场景是在哪里。第三个是我们罗列一下业界的一些当前的方案的一些痛点。第四个是我们去拥抱这个浏览器内置的这个外文MCP。第五个是介绍一下我们的web q和这个web age。第六个是我给大家去过一下代码，直接实战去看下一个view应用是怎么样去通过几个简单的配置就可以实现这个real应用的一个智能化。

议程概览：WebMCP 智能应用升级之路



开启智能办公新时代 · WebMCP 技术分享会

38:04

第七个是总结与展望。先说一下我们为什么要智能化这个应用智能化，先看一下我们现在有现在应用的一些重点，比如说有一些工作，他需要全程24小时去高度的精力去关注这件事。然后这些其实都是一些比较重复的操作。那这样的话我们可以交给AI然后去提供一个说明书一样的东西，让AI帮我们这么做。这个说明书其实就是后面我要说到的这个web q然后就是降低成本，就是流程自动化智能辅助智能助手的一个辅助。

我们为什么要应用智能化？



38:47

比如说你的整个后台管理系统是比较复杂的一个。比如说ERP系统也好，一个CIM系统也好。但是这里面你比如说招来一个新的一个员工，那这个员工就可能需要阅读你大量的文档，你的说明书，你整个应用的说明文档。那他才会还有一些就是培训的一些时间，培训的一些时间，然后加上一些金钱的，加上一些时间的一些成本，所以这块我们就可以帮你去降低这个成本。让AI去看说明书，然后AI看着说明书再去操作你的页面。

39:22

第三个是优化一下体验。比如说用户他使用你的外部应用的时候，他可能对你的应用他肯定不会接受你的这么频繁的一个培训，对吧？他拿过来用，他就想着我直接通过一些简单的一些对话，我就可以去完成我自己的一个诉求。然后在同一个场景来出发。

WebMCP + WebSkills: 企业级智能化页面操控方案，兼顾极致安全与高效落地

✎ 情景再现：小明的“职场救赎”

🔴 困境：新人遇急单

入职两小时的小明，被老板要求立即为客户创建1000元价保申请单。面对陌生的复杂后台和财务逻辑，他焦虑万分。

🤖 转机：AI 助手出手

同事指点：“直接跟系统说话！” 小明尝试输入指令：“帮我给用户王五创建1000元价保单。”

😊 结果：一句话搞定

系统自动识别身份、调取信息并预填表单。几秒钟后，申请单生成，点击确认即完成。效率革命就此发生！

⚡ 核心价值：化繁为简，让复杂的业务操作变得像聊天一样简单

39:44

比如说刚才我说的一个例子，就是一个新员工来到一个公司入职，然后他老板第一天对他没有过多的培训的情况下，就让他去完成一件他从来没有做过的事情。比如说他要为某一个给客户创建一个价保申请单，这个价保申请单需要哪些字段，需要哪些，比如说一些订单管理，订单的一些ID或者商品的一些ID刚入职的小明他肯定是不知道这个事情的。所以当这种情况下，就需要我们这种AI助手去部署。只要他跟这个对话框去说明一下老板那个老板给他原封不动的诉求，告诉这个AI然后AI去匹配这web SQ，然后去读说明书就可以知道老板这句话，他需要哪些字段，可能需要哪些字段。然后就搜寻这个客户的一些必要的信息，然后给给这个给这个小明，然后让小明去去看他可能需要哪些字段。然后如果有这些字段，它直接自动的去输入输出给AI然后AI去自动帮它创建这个单。然后再看一下我们在这个现有方案的一些痛点。

业界痛点：现有方案处理复杂页面能力有限



基于无障碍信息的方案

核心局限：局限于简单结构，依赖页面无障碍信息，对复杂布局和动态内容处理能力差。

具体痛点：

- **操作不可控：**AI行为难以预测，可能执行危险操作。
- **信息局限：**只能获取页面上已有的信息，无法调用后端数据。
- **稳定性差：**依赖页面结构，结构变化易导致失败。



基于视觉模型的方案

核心局限：对复杂交互力不从心，虽然能理解布局，但难以处理复杂的业务逻辑和交互。

具体痛点：

- **执行不稳定：**对页面微小变化敏感，易失败。
- **安全性低：**可能误操作，权限控制困难。
- **成本高昂：**需要强大的视觉模型支持，推理成本高。



基于DOM解析的方案

核心局限：强依赖页面静态DOM结构，对动态渲染和复杂交互无能为力。

具体痛点：

- **脆弱性高：**DOM结构稍有变化（如class名、标签层级），定位就会失效。
- **交互理解难：**无法理解需要复杂逻辑判断的交互，如“点击按钮后，等待弹窗出现再操作”。
- **维护成本高：**需要为每个页面维护大量的定位规则，开发和维护成本巨大。

41:01

我们现有的方案比如说基于无障碍信息的，基于视觉模型的，还是基于动物解析的这块。其实下面几个例子可能这个视频可能就能交代清楚，就是现在的这些这些无障碍视觉模型动物体系，它其实都能解决一些简单的一些中等或者中等简单的一些操作。比如说点个按钮，然后点开菜单，打开一个页面，填写一些简单的表单。但是对于那些复杂页面，复杂操作，如果你不给AI这些背景知识，比如说你创建一个采购订单或者创建一个价保单，你需要哪些前置操作，这些东西通通不知道。他只能基于现在这个页面自己去做他自己的一个想想想法的一个实践。你比如说他可能想法跟你之前定义的那种操作可能完全不一样。而且这种有也有个他们三个共同那个地方就是安全性比较低。因为他其实按照自己的想法来，他有可能就会操作一些你意想之外的一些高风险的一些操作。

42:08

接下来我就简单看一下。比如说这是通过一个medicines，它是一个业界的一个AI测试框架。他其实就用一个一种纯视觉的一个模式去操作的页面。然后我们简单看一下，就简单看一下，也不花太多时间给你看。

42:24

这里的话我从头拖到尾，就简单的。这样拖过来。就是他可能需要不停的去截图，然后去做思考判断，然后不停的去操作。点击按钮，然后填写表单，然后整个流程下来。你看到这整个流程下来大概有五六分钟，六分钟就是操作。一个比较简单的操作，他可能需要几十分钟或者是几分钟那样的去去操作。

42:59

一个简单的一个其实人为的话大概需要个两三分钟就解决的一个问题。这个是一个纯视觉模型，然后再看一下这个基于动物的，基于动物的话，它这个可能就时间就更长了。而且它这个怎么说呢？他这个东西他这个错误率还挺高的，就是他点着，他有可能自己忘了自己之前点过啥，然后到后面就可能输错了，他也不知道，然后就整个就陷入一个死循环，后面就没法执行下去了。可以看到这里其实用到了大概七分钟才完成这个操作。因为他这里中间的话他做错了，做错了，然后他来来回来回倒腾，最后还

是没有完成这个任务。最后这个就没办法，就直接七分钟之后他就直接没法提交了，然后直接就退出了。

43:52

然后再看一下这个，这个是两个通过演示的就可以看到他对于这些比较复杂的一点操作的，他可能就需要花的时间长，而且他需要获得token。给大家说明一下，就是这个micon他他token大概是操作多做一下，新建一个新建一个新建一个我们team pro的一个一个一个一个国际化，还是什么操作。然后他一顿操作大概花了有40万的top，大概40万的top。

传统方案操作 Web 应用效果



通过 Midscene.js 操作 TinyPro

基于视觉识别
代表选手：Midscene.js

演示地址：<https://www.bilibili.com/video/BV1e3XEBGEt1/>



通过 page-agent.js 操作 TinyPro

基于DOM识别
代表选手：page-agent.js

演示地址：<https://www.bilibili.com/video/BV1x8XEBFEs6/>

44:27

然后我们再看一下，我们基于这个外部MCP，它这个整体操控这个pending pro就简单给大家看一下它的整体流程。因为它是基于外部MCP的，所以它的他是没有任何的一个错觉的，他就是精确的能知道我想要干嘛。他这里其实你看他用了大概一分钟，这一分钟之前这个一分钟其实这里是一个龙虾初始化的时间，花了大概有几十秒。其实它真正操作的话可能就几十秒，大概半分钟就已经搞定了。所有的这个流程就添加这个字条国际化词条，就是给大家。到时候可以把这个PPT分享出来给大家。可以到这个哔哩哔哩网站去看一下这个整体的一个操作流程。

45:20

从上面我们就可以得出个结论，我们这个YMMCP它的作用就是去操控一些复杂的应用。然后把这些复杂的应用变成那种人人可以操控，人人可以去玩转的一个一个智能化的应用场景。接下来我就说一下什么是原生的外部MCP。原生的外部MCP就是谷歌在今年刚推出来的，在146版本。其实你现在如果你升到146版本，如果不打开了那两个设置的设置的选项，你还是没法用。所以说现在它只是一个实验阶段。

WebMCP 方案操作 Web 应用效果

基于 WebMCP

代表选手：
WebMCP-SDK
(OpenTiny)



演示地址: <https://www.bilibili.com/video/BV12tXEBhEr7/>

45:58

但是我们现在是提供一种，怎么说呢？就是我们提供一种point view，让你直接可以在低版本浏览器也可以用。它是一个标准的定义，统一的一个工具注册入口。它使用的API就是从navigate里面去拿到这个model context。拿到这个model contest，你就可以model contest点reject to去注册工具。然后它的调用工具，可能你就需要它是一个navigate点model context的testing。它后面又加了一个testing，可能是一个测试的一个。所以他去去那个excuse to他并跟那个M那个MCP不一样，它是excuse MCP的是两个语法还不太一样。

核心变革：统一推荐原生 navigator.modelContext 接口

告别自定义实现，拥抱标准化的浏览器原生能力时代，构建更稳定的未来。



开发标准化

遵循浏览器官方标准规范，彻底告别自定义实现的兼容性与维护负担。



全端普适性

内置 SDK 提供的 Polyfill 能力，确保在所有新旧浏览器环境下都能无缝运行。



交互智能化

自动完成 AI 工具注册、路由状态同步与智能调用，提升开发与运行效率。

技术升级 · 体验跃升 · 开发提效

46:46

然后核心变革，我们现在这一阶段大概是零点。我们SSDK的0.3.1版本就全面拥抱了这个原生的navigate点model context，我们为什么要这么做？就是因为我们需要去兼容后面。比如说你后面有谷歌浏览器，正式把这个navigate model context引入了正式版本，并且不需要去自己去打开某些设置。那么我们就可以无缝的去切换到，所有的语法都不用变。然后你就可以直接去享受这种无缝低成本零成本

的一个升级。这边我们就可以看到，我们到之前说的，为什么要选择它？因为它是标准的，然后是他我们是兼容他的，然后我们就提供一个垫片 Polly feel去让你在当下就可以使用那些未来谷歌浏览器版本才能使用的API。

拥抱未来：为什么选择原生 WebMCP?



标准化与兼容性

遵循浏览器标准，无需关心底层细节。随着浏览器原生支持增强，应用性能将自动获得提升。



极致性能与效率

原生实现比 Polyfill 更高效。按需注册机制可显著减少传递给 AI 的上下文大小，提升响应速度。



开发体验简化

直接使用浏览器原生 API，无需复杂封装，代码更简洁、更符合直觉，降低维护成本。

47:43

直接使用原生API就有一个好处，就是你不用在你的外部应用里面去引入任何的包。你直接使用name get点那个model context，你就直接可以去定义一个工具。我们SSDK的角色也从提供者到这个赋能者，我们是不推荐大家再用那个web MCP那个API。其实那个API也是可以用，但是我不推荐这么用了。因为你如果是web应用的话，你就直接使用浏览器内置的这个原生的API，其实是更好更兼容的。然后我们只需要一个导入，从那个导入这个initialize build in标题应用这个外部MCP，你就可以在你低版本浏览器使用我们内置的这个浏览器，内置这个外部MCPAPI，提供原生的API能力。

SDK 的角色转变：从“提供者”到“赋能者”

next-sdk 核心职能升级：不再局限于基础支撑，而是全面释放 Web 应用原生潜能



提供 Polyfill 兼容层

通过`initializeBuiltinWebMCP()`函数，为不支持原生 webmcp 的浏览器提供统一兼容实现，确保 API 调用一致性。



增强原生 API 能力

超越基础功能，提供如自动路由同步、WebAgent 远程遥控等高级特性，显著提升应用交互体验与操控能力。

核心价值：打破浏览器限制，释放 Web 应用原生潜能

迁移实战：前置知识 - WebSkills：AI的‘智能说明书’



核心定义

一份用自然语言(Markdown)编写的智能说明书，定义了AI的工具、所需工具及页面间的导航规则。



核心价值

渐进式披露提示词，让AI像人一样逐步理解复杂指令，更精准地执行任务。



实现步骤

1. 在项目根目录创建 skills 文件夹
2. 在 App.vue 中使用 import.meta.glob 批量加载 .md 文件

掌握 WebSkills，让 AI 交互更加智能与高效

48:36

之前把这个YMCP和原生的内置的这个外部MC都介绍了一下，然后紧接着我们再介绍一下这个web q的它是干嘛的。Web sk e就是我也不多说了，因为这个skill大家都最近都比较熟，因为他最近比较火。我们这里的web sq其实跟业界的sq的规范是一模一样的。

48:57

然后我们其实只是在前端去实现了一套文件主体的一个API。就使用这个wit提供的这个import meta global去批量加起所有的MD文件。包括你一些ggs文件或者其他文件都可以。只要是读取的一些文件，都是可以读到我们那个里面去，然后传给remote组件。然后remote组件去去通过这种渐进式披露去完成一件一件比如说一件非常复杂的事情。比如先打开某些页面，然后操作完某人流程，然后再打开模型界面。

生态集成：接入远程遥控 (WebAgent)

通过 WebAgent 实现页面的‘可控性’，真正实现 AI 对应用的‘远程驾驶’。

核心亮点：

- 获取持久化的 Session ID: 与云端 AI 编排器建立长效连接。
- 跨时空协同: 即使用户不在当前聊天窗口前，AI 也可以根据业务指令远程调用页面工具。
- 高阶业务自动化: 将页面上的工具转化为自动化流程中的执行单元。

```
import { WebMcpClient } from '@opentiny/next-sdk'
const client = new WebMcpClient()
// 连接至 WebAgent 服务
const { sessionId } = await client.connect({
  agent: true, builtin: true, url: 'https://agent.opentiny.design/api/v1/webmcp-trial/mcp'
})
console.log('连接成功! 会话 ID:', sessionId)
```

WebAgent 赋能 · 随时随地掌控应用状态

49:33

再介绍一个我们这个亮点，就是我们这里就算你是使用内置的YBMCP浏览器内置的这个工具，你也可以直接接入我们的外表景。然后你这个API可能会有有一点点变动，但是它整体变简单了。它不需要去import一个web MCP server，它只需要import一个web MCP client。然后在这里丢了一个client之后，把这个adding的true，然后build in true这个配置上，这里的话，你这里可以使用这个我们提供的，但是这个最近他在整改，所以现在访问不通。可能你需要自己去部署一下自己的外表进程，然后这里的话就可以去远程遥控。

50:14

接下来后面我会给大家做一个实战，然后给大家演示一下。这里的话实战第一步就是可能你需要在你的money DS里面去去引入这个Polly feel，然后去执行一下。这样的话你就可以在你的页面里面直接使用never get点model context。然后同样的你这里还需要去帮我们设置一下这个set never get。因为这个路由路由就那个路由注入的主要是为了跳转页面。因为有些的业务操作需要去在多个页面之间去去跳转之后再再去完成的。没有说一个很复杂的应用，只有在这个页面做完就完了，他可能还需要再跳到别的页面去。然后这里注册的话，你这里就可以看到我们的注册就变得非常简单。

迁移实战：第一步 - 环境初始化 (main.ts)

src/main.ts 核心配置代码

```
// 1. 注册核心导航器：告诉 SDK 如何跳转页面
setNavigator(async (route) => {
  const failure = await router.push(route);
  if (failure) throw new Error(`跳转失败: ${failure.message}`);
});

// 2. 激活浏览器内置 WebMCP 服务
initializeBuiltinWebMCP();

// 3. 标准 Vue 应用挂载流程
createApp(App).use(router).mount('#app');
```



setNavigator 路由注入

向 SDK 注入路由跳转逻辑，确保页面跳转符合业务预期，捕获并处理跳转异常。



WebMCP 服务激活

初始化内置 WebMCP 服务，自动处理浏览器兼容性，为后续功能调用奠定基础。

迁移实战：方案一 - 页面内按需注册 (中大型应用首选)



核心理念：工具与页面生命周期严格绑定，需要时才注册，不需要时立即销毁。



减少幻觉干扰

工具只在对应页面挂载时存在，上下文纯净。



降低系统负载

工具列表随路由变化自动增减，避免资源浪费。



职责划分清晰

工具与业务页面紧密结合，代码逻辑维护性高。

Vue 3 代码实现示例

```
// src/views/product-list/index.vue
import { onMounted, onUnmounted } from 'vue'
const ctx = navigator.modelContext
onMounted(() => {
  ctx?.registerTool({ /* ... */ })
})
onUnmounted(() => {
  ctx?.unregisterTool('get_detail')
})
```

51:02

我只要在这个mounted里面，就页面挂载的时候注册一下，在m money的里面上去取消注册一下去。这个为什么要取消注册？因为你不取消注册，这里的内存没法释放。所以我为了防止内存泄露，你最好是注册完之后，再按问题的时候把它取消注册一下。这里就是页面按需的，就是你这个页面不加载，你这工具就不会加载了，所以就可以极大的减少这个AI的幻觉。你整个应用可能是二三十个工具，如果你全部给AI那AI的话上下文他就会被撑爆，那他就会出现那种极大的一个幻觉，然后就做着各种匪夷所思的事情。

51:43

然后我们再看一下这个方案2，方案二就是分离式的全量注册。这种全量注册就是我需要在这里全量注册一个工具，然后在这里我就不写回调，然后这里指向一个路由，这个东配置是干嘛的？就是我全集都会加载这个工具，就是你不管跳到哪里我都会加载工。然后我在调用这个工具的时候，我会自动帮你跳到这个路由。然后你在页面里需要调用一下，从next SDK里面去引入这个reject page two，然后去注册这个handle，这个名称和这个名称要对应上。然后我们就可以帮你自动打开这个路由，就不需要你手动打开路由器。

迁移实战：方案二 - 分离式全量注册 (适合轻量/小型应用)



核心理念：一次性注册所有工具，通过配置 routeConfig 让 SDK 自动接管路由跳转与逻辑分发。



实现简单高效

无需编写复杂的 Skills 引导词，配置即接入，开发成本极低。



AI 能力全感知

工具对 AI 始终可见，调用时 SDK 自动处理页面跳转，体验无缝。

// Step 1: 全局注册 (TypeScript)

```
navigator.modelContext.registerTool({
  name: 'finance_summary_query',
  routeConfig: { route: '/finance' } // 关键跳转配置
})
```

// Step 2: 页面绑定 (Vue Setup)

```
onMounted(() => {
  cleanup = registerPageTool({ route: '/finance',
    handlers: { 'finance_summary_query': async () =>
  { /* ... */ } }
  })
})
```

52:21

而remote的话就是呃呃引入组件，然后去配置一下MCB服务器。这里的话我们配置的是谷歌浏览器内置的这个build in web MCP。然后它需要的是type是build in，然后client就是这种客户端，我们直接给它指向navigate model concepts的testing，就是刚才说的那个API。这个API它为什么要做testing？可能是它是用来调试用，他可能还没有正式的对外暴露，但是他目前来说是可以直接用的。再加上我们有玻璃费用，就算没有的话你也可以用。然后这些就是刚才说的，直接把所有的Q的文档直接使用met的emote meta点globe引进来之后，直接传给这个remote，就可以使用sq技能。

生态集成：接入 TinyRemoter 对话面板

基于 OpenTiny 快速构建 AI 原生交互界面，配置即插即用



组件引入 (Import)

直接从 @opentiny/next-remoter 导入 TinyRemoter 核心模块



服务配置 (MCP Server)

将客户端指向 navigator.modelContextTesting，对接原生能力



技能加载 (Skills)

通过 import.meta.glob 自动扫描加载 Skills 目录下的所有指令

核心代码实现 (App.vue)

```
// 1. 引入组件
import { TinyRemoter } from '@opentiny/next-remoter'

// 2. 配置 MCP 服务器
const mcpServers = {
  'builtin-webmcp': { type: 'builtin', client:
    navigator.modelContextTesting }
}

// 3. 加载本地技能文件
const skillMdModules = import.meta.glob('./skills/**/*.md',
  { eager: true })
```

🔑 关键优势：零侵入式集成，直接调用浏览器原生 AI 能力，开箱即用

决策指南：WebMCP 方案选型对比

原生内置 WebMCP (按需)	分离式全量注册 (全局)
 推荐场景： 中大型、业务逻辑复杂的系统	 推荐场景： 小型应用、功能单一工具
 幻觉风险： 极低 (工具随页面动态上线)	 幻觉风险： 中等 (工具全局常驻内存)
 路由依赖： 依赖 WebSkills 配置指引	 路由依赖： 依赖工具自身的 routeConfig
 核心优势： 上下文纯净，运行性能最优	 核心优势： 一站式注册，开发效率极高
 总结： 规模决定方案，按需方案更稳，全量方案更快	

53:10

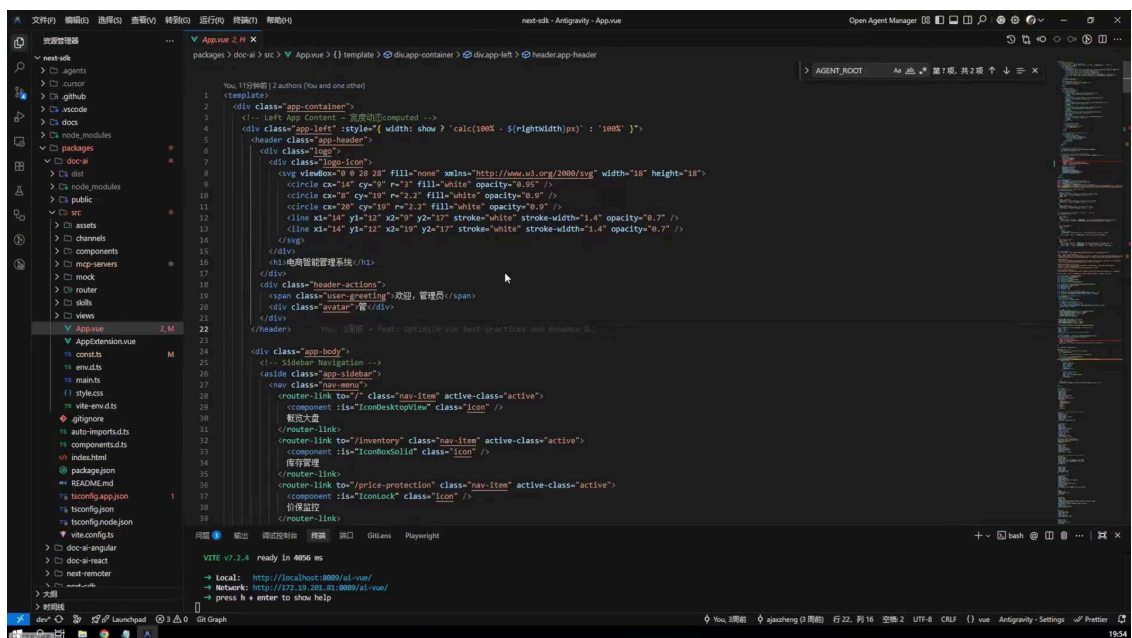
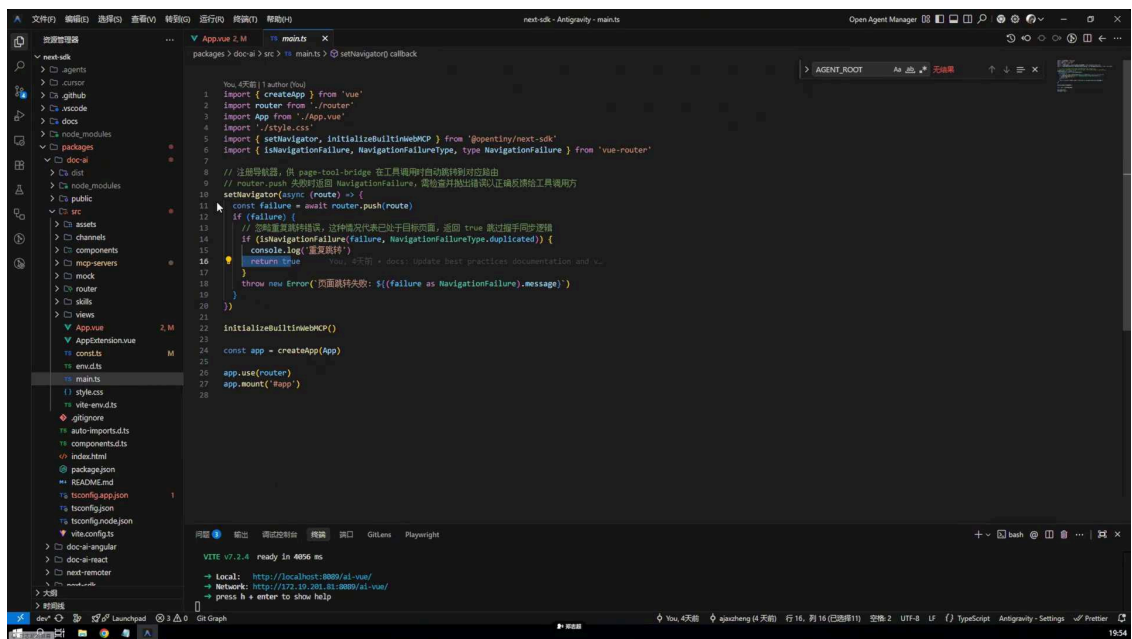
我说一下这个说完之后我就直接开始代码进行演示了。就是我说一下这个原生内置的按需的还是分离式的，是在什么场景下用这种原生内置的，它就是那个页面加载了这个工具才会出现。它就属于那种按需的，就是减少幻觉中大型的系统，然后幻觉的风险是极低的，然后它需要依赖web q因为因为你如果不把所有的工具都给AI，AI不知道你有什么工具，你所以你要在web SQ里面说明你有什么页面，哪些页面里面有什么SQL可以用。比如说我操作一个价保价保申请单，那么我就需要先跳到订单管理，跳到订单管理，找到订单号之后，拿到订单号我去这个价保申请管理页面，然后再再去创建这个用户的一个价保单。这种就适合那种比较大型的，中等复杂的，中等偏上复杂的。

54:05

然后这种分离式的就是刚才它自动他就你这里就不用写skill，你直接把这种全局的全局的这个外部外部C直接抛给AR。AR的话可能你这个应用可能比较小，大概有个十个不到的MMCP工具，然后全部给AIL，那就靠AR了。如果他他觉得他可能他这个AI的智商比较低，那可能他的幻觉就比较高。它如果AR的成本比较高，那那这当然这跟你的应用的复杂程度有关系。你用应用如果很简单，那就没有那么多的一个幻觉的风险，然后它就会自动帮你跳转路由。

54:44

行，那我们就是给大家简单介绍一下我们这个整体怎么配置。就是我用一个标准的我们最佳实践最佳实践的一个V5智能化应用给大家介绍一下。首先我们需要在这个入口文件里面去使用这2个API。这个API就是路由跳转器，就是我们在跳转路由的时候都会经过你去定义，那这里你就可以去控制。比如说你发现是重复路由，你直接回接触，这样的话我们就不会再去等待握手。



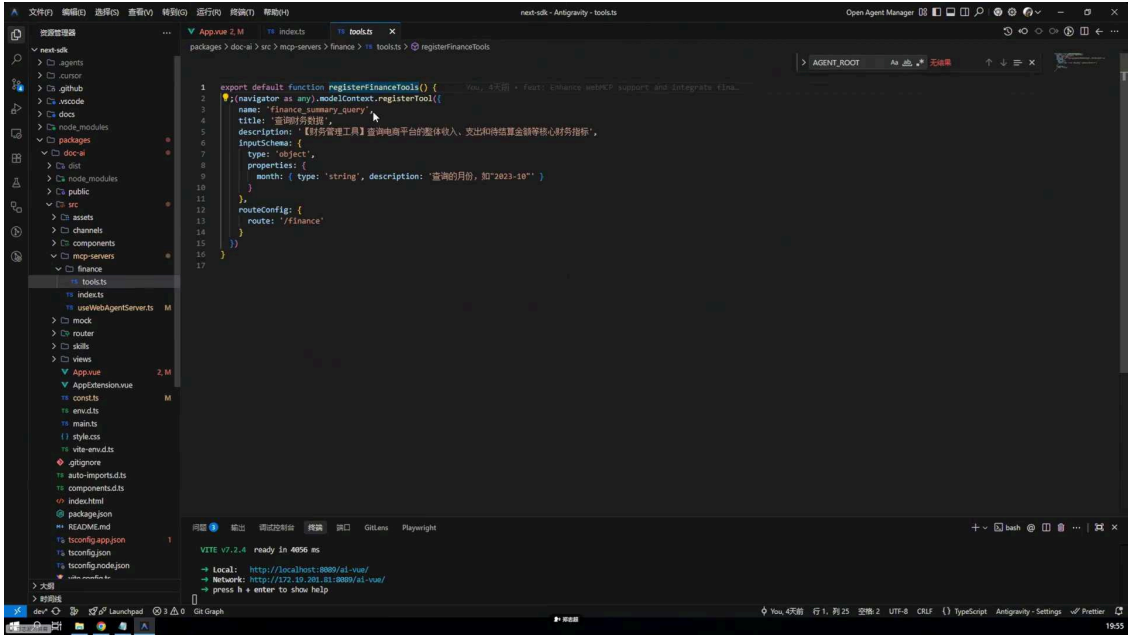
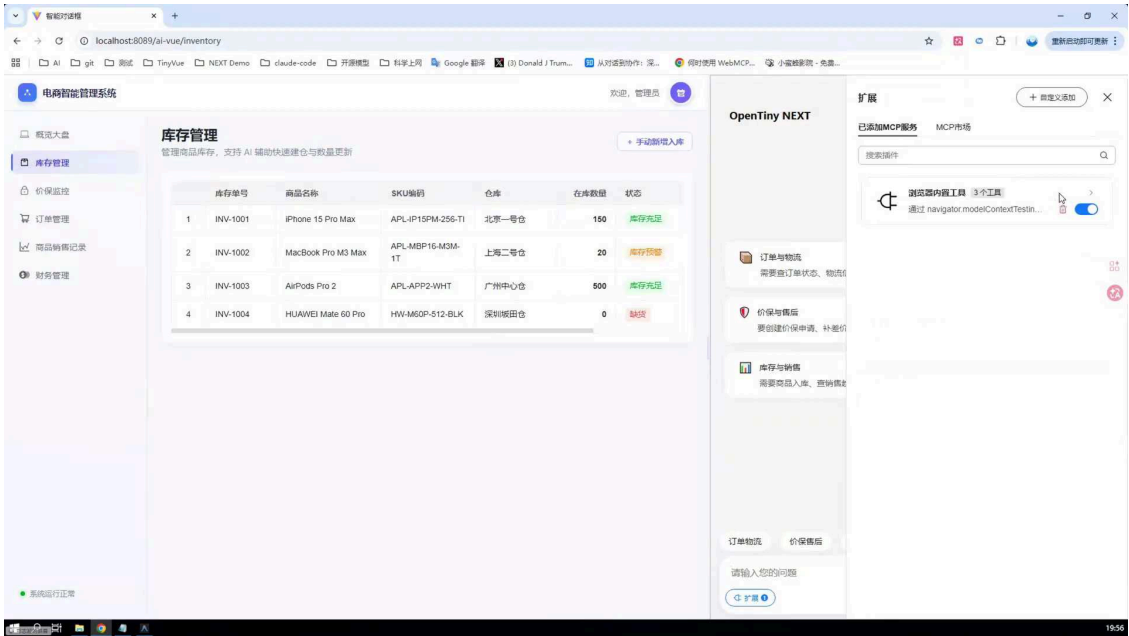
55:16

就是一个页面它如果又跳转了，它并不代表那个页面里面的那个工具就已经注册了。因为你你要知道这个路由跳转过去只代表这个路由是过去了。但是这个页面有没有mounted，有没有挂载，这个谁都不知道。所以我们这个内部会有个握手机制，就保证你跳过去之后，握完握手完成之后，我们才去调用这个工具。所以在这里如果你是重复调用，你就重复的路由，那你就？你就是在当前跳转已经重复了，你就直接ren处告诉我们，就不去不用去握手了。如果是不同的页面，那么你就正常的去返回。然后我们就会进行一个精确的握手，握手之后我们再就会调用对应的工具。

55:57

然后第二个就比较简单了，就这个东西调用一下就完了。然后我们就会帮你去poly fill一下，把你这个低版低版本浏览器也能去支持这个YBMC，这是第一步。我们看一下这个工具怎么注册，工具注册的话分为几部。第一个就是我们内置一个we just never get to我们内置的一个工具跳转这个工具跳转的路，这个工具是会自动握手的。就是你不用担心你跳过去的那个工具还没下载，我们会自动帮你握

手，握手成功之后才会进行对应的一个操作。然后这种方式就是刚才说的这种小应用，就是这种小应用才会小小应用才会去集成的一种这种分离式的。它的话当然它也是直接用这个name get点model，它就是用原生的，然后它会多了一个配置项，这个配置这个工具会常年的存在你的内存里面，就是我给大家看一下。



57:03

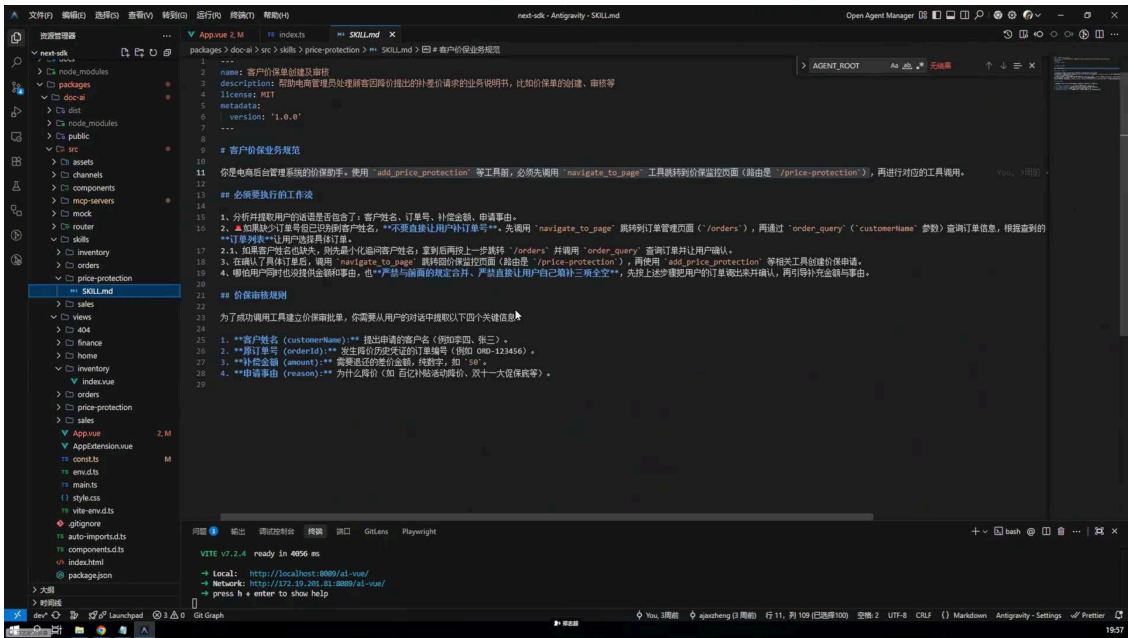
这个工具这个工具就是电商，就是财务管理。财务管理这个就是财务管理工具，它是会常驻在，你不管切换到哪个页面，我给大家看一下这个。就比如说你点到不同的页面，你会有不同的工具，它会在这里不停的刷。但是怎么刷，他这个财务管理的这个工具都存在。就是因为它是常驻的，然后它是一个分离式的这是在小应用才适合去这么用。

57:29

我们再看一下大型应用这种按需加载的是怎么用的，比如说这个页面，然后你就可以在这里直接使用浏览器原生的这个\$2。然后在这里你可以去定义你自己的这个。这里你可以拿到你当前页面的所有的内存的变量，然后你就可以直接在你的这里执行你自己的业务逻辑。然后这个页面就比如说这个是库存管理，这个是库存管理。如果你你你在这概览这个页面首页，你是没有那个文件。但是如果你切换到库存管理，你看到这里就会有一个入库管理工具，就说明这个工具它并不是一直存在你的那个AI对话框里面的，就不存在你的不一直存在你的那个上下文里面的对AI上下文里面的需要打开它的时候它才会出现。

58:24

然后这是外部MCP的用法，然后我们再看一下这个QQ的话也是单独一个文件夹在SSRC下面。这里的话它跟业界的sq是完全一模一样，你按照业界的这么去去去用就行了。它的定义方式也跟业界一模一样，所以我们这是完全遵循业界的这个sq e的规范的。

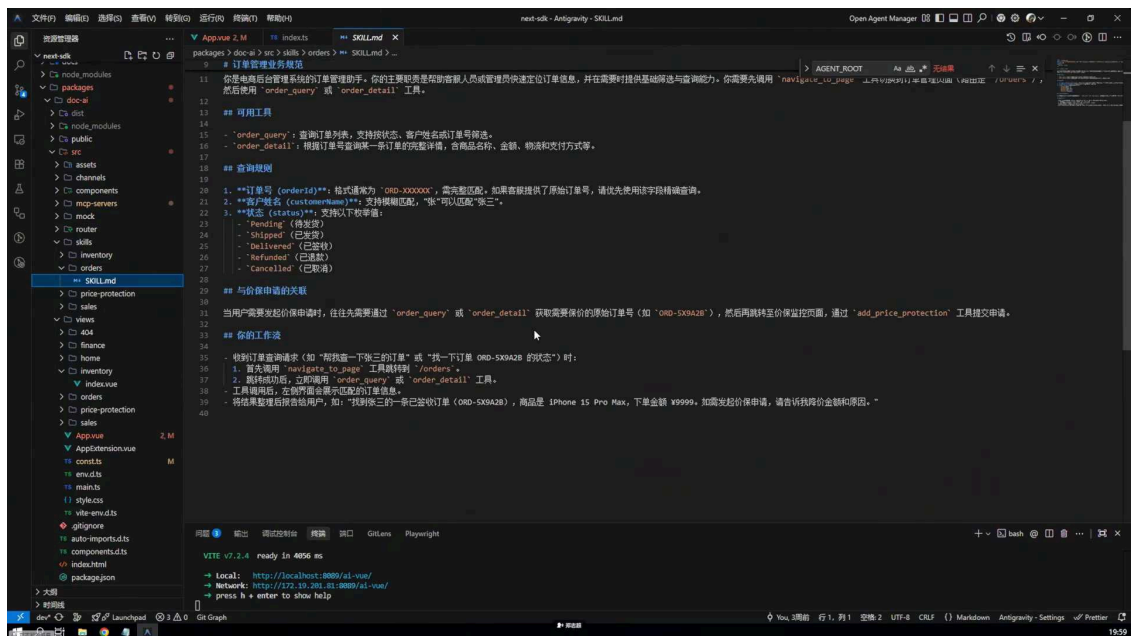


58:44

然后在skill里面，我就刚才说了，如果你是中大型应用，按需加载那种，那你就需要在sq里面去把你的这个工具是怎么调用。然后哪个页面干了，就是哪个路由干某件事情，你就给他交代清楚了。就比如说我们以这个一个价保单申请的这个例子为准，就是他可能先需要跳到这个价保监控页面，然后这里就必须执行的工作流就要告诉他。就比如说他需要哪些字段，你要跟他说。

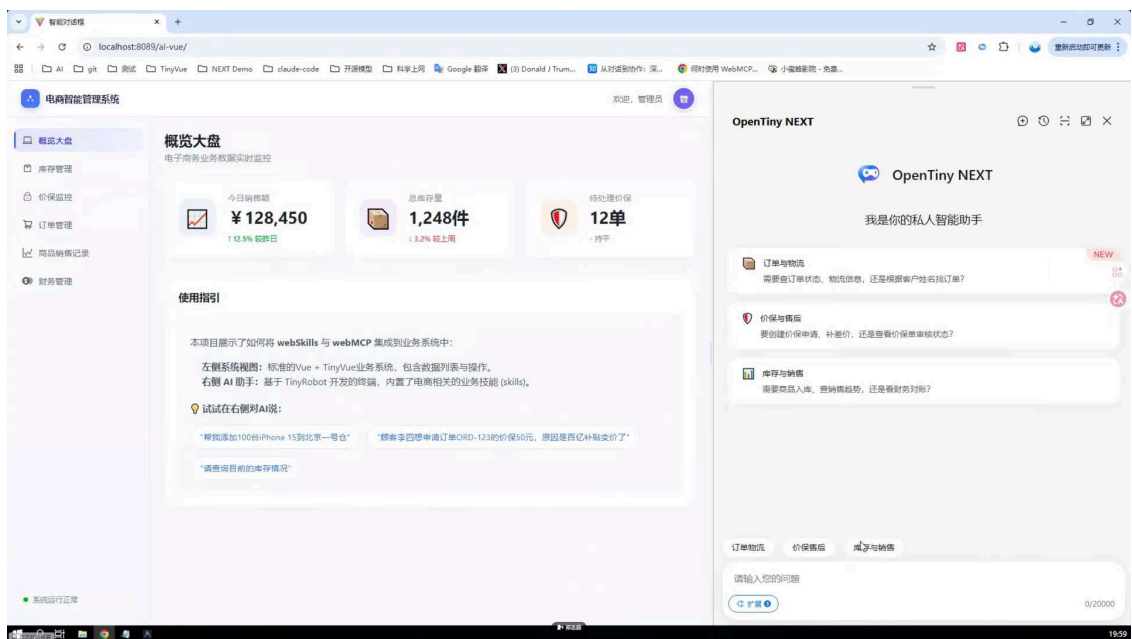
59:15

然后这个如果缺少订单号，不要直接让用户补订单号。因为用户补订单号他要自己去查，那这个东西没有减轻用户的一个负担，所以你要先去使用路由跳转，先调到这个订单管理，然后就去找到这个用户姓名对应的订单号。找到订单的号之后，你直接去去跳转到那个价保管理页面，然后直接去创建那个价保的整体的。你只要告诉清楚了，他就可以去干一些也很复杂，流程很长的一些东西。包括这里面所有的都是可以自己去定义，定义好了之后直接告诉他就行了。



59:50

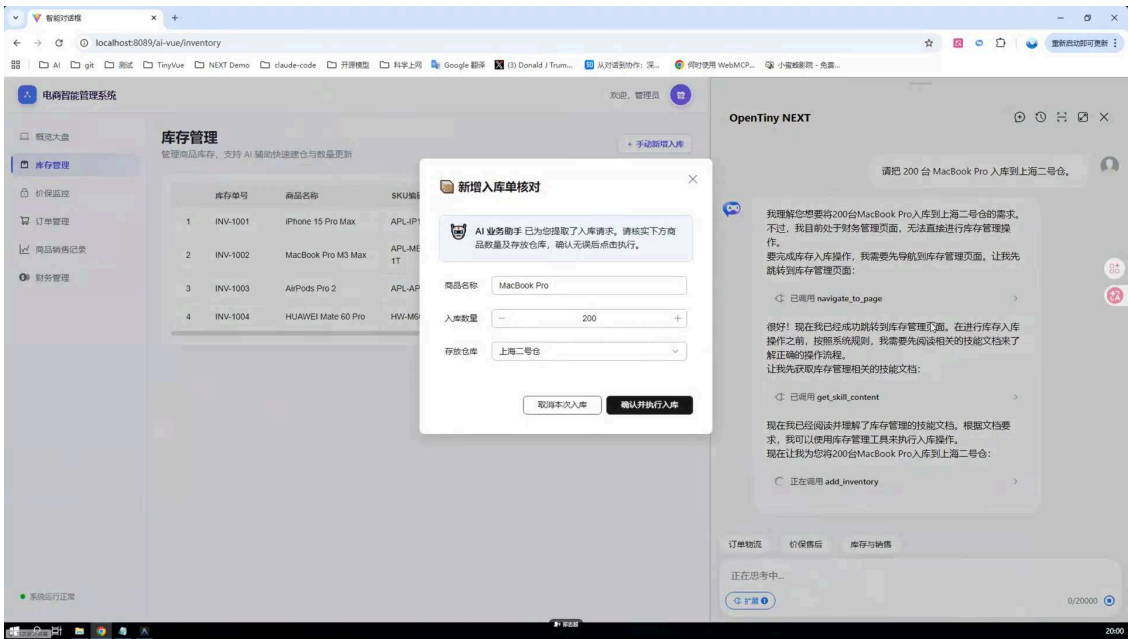
然后船的话就是刚才有两个，一个是MCP server，一个是一个skew。我们看一下怎么传进去，传进去就是比较简单，可能你skew的话，你就按照之前我们说的这个。我给大家看一下，这之前说的直接把那个东西引进来就完了，直接APR就非常的简单。然后这个MCP server就是一个固定的写法就行了，他只需要把这个传进来，我就知道你是内置的，内置的时候我就可以再去调了。先这么说，然后给大家演示一下，就是怎么样去比如说我让他帮我上架一个商品，就是商品入库，那这时候他会去读我的sq e然后找到这个我需要怎么去上传这个商品，看看这里我给大家演示一下。



01:00:56

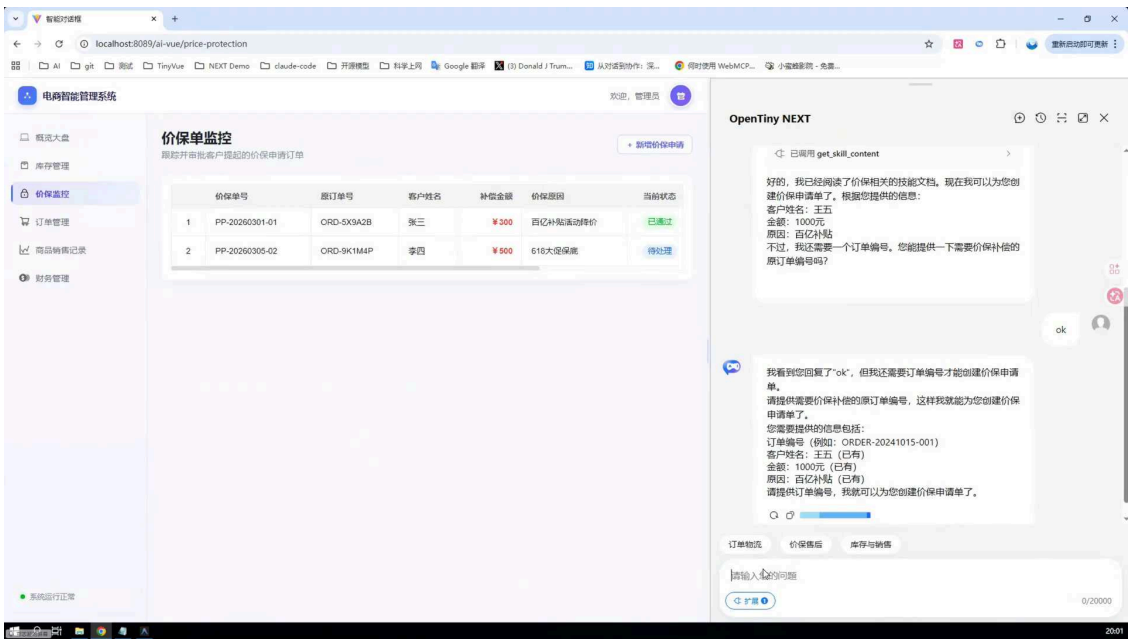
他先去导航的这个界面，可能这个简单它就不会去调用SQL。但是他也会调用一下他达到这个技能说明之后，然后他就可以直接就去调用这个了，这里有个安全性的操作，就是这种确认入库或者是高危的风险。那么你就可以在你的MCB工具里面去做一下拦截。就比如说我只有人为点确认，然后他才可以调用

成功。当然他可以直接去加入，但是这样的话就缺少一种人为干预，就是高风险操作的人为干预。这个安全性方面就不能得到很好的保障。然后这是一个简单的操作。



01:01:32

再说一下这个比较复杂的一点，就是刚才说的价保申请单。那可能需要一个订单管理，他需要跳到订单管理去获取到这个订单管理这个界面。首先他跳到这个价格页面，然后他再去读取skee，他会发现一个需要一个订单号。那么我就说OK，其实他这里有时候他会直接去帮我就跳转，但这里他需要征求我的同意，那我就对吧？好。



01:02:14

好，我再重新启动一下，我再重新跟他说一下，可能他这个阶段他不知道为啥。他读sq的时候没读到吗？看一下。他已经调到价格量了，就是他他应该是先去调那个。

01:02:56

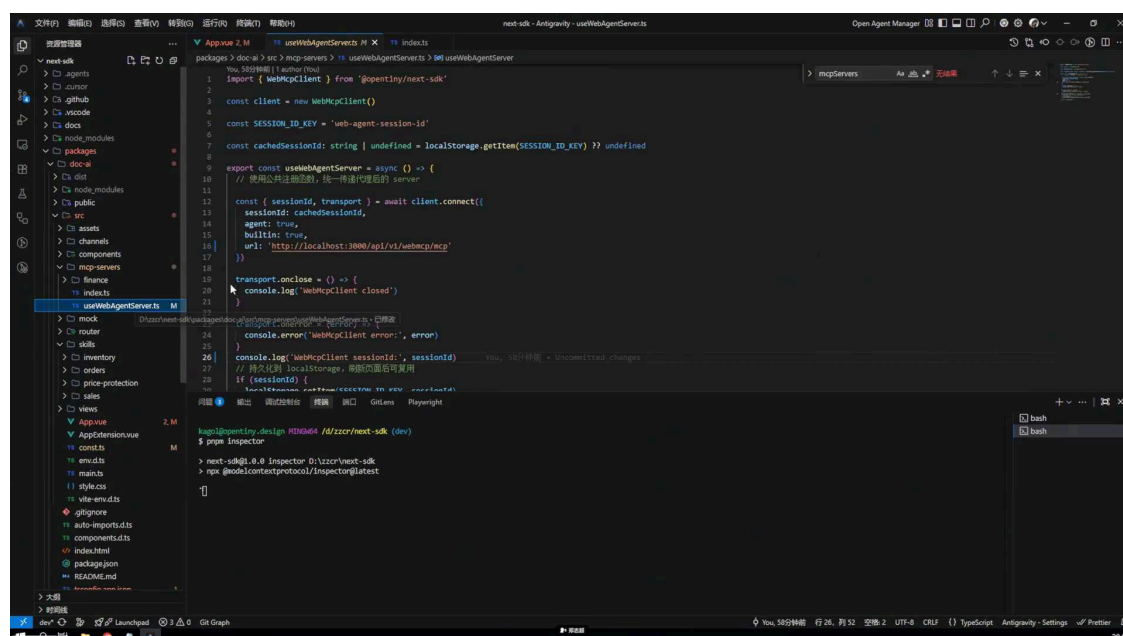
其实我在sq里面已经跟他说明白了，其实这个跟AI的其实他的理解可能也有问题。就是我这里其实已经跟他说说明了，我这里需要去就跳到这个订单管理页面。还没有订单管理。

01:03:29

这里他找到了，所以说这个怎么说呢？这个跟你的提示词和你的skill的描述可能也有关系。如果你提示词写的非常的明了，他可能就会执行的比较准确。这里他其实就已经创建完了，然后这里就创建成功了。好，然后我们再说一下最后一点，就是我们的web age，web怎么去远程操控这个东西。因为我们现在我们是他牛肉部的产品太他那个网站还没有正式的恢复，那我们就直接使用一个MCP官方的一个inspect。

01:04:20

去操作一下。我先给大家看一下这个外边景我怎么去使用的。首先在这里我们会有一个use外边景的这个server，然后我们在这里直接去连那个外边镜，然后这里外边镜我们可能需要我们可能需要去本地启一下。因为我们在线的那个外背景的已经没法访问了，所以我们这个本地启动一下。所以这个外边你怎么用？刚才景浩已经说了，我这里只是去把它启动一下，然后给大家演示一下。



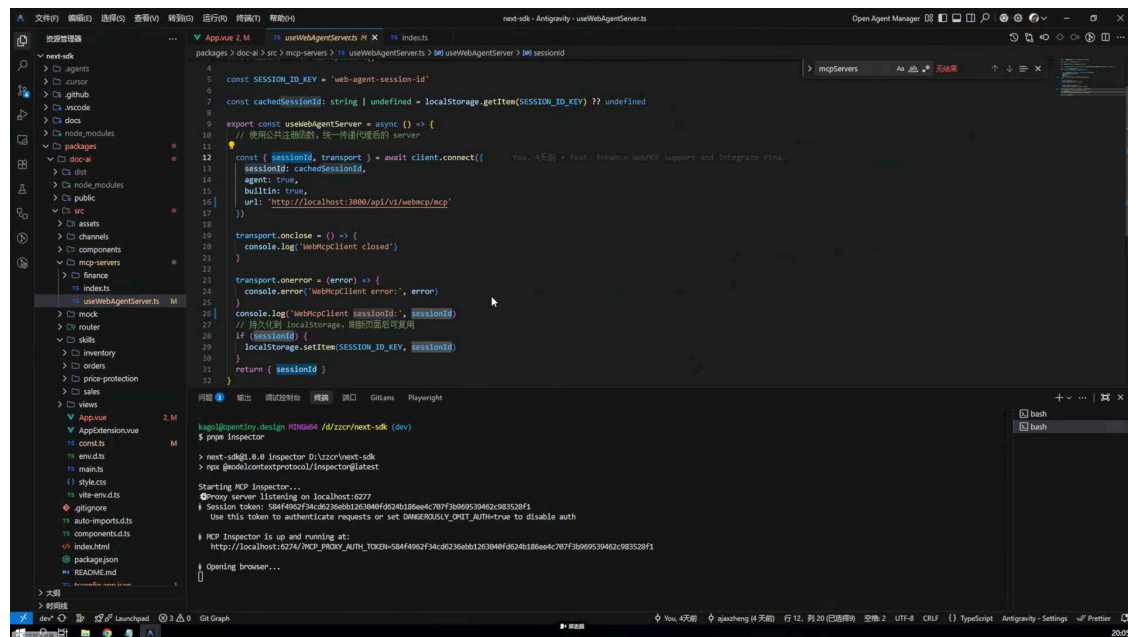
01:04:58

然后这个我给打开的是一个外表件的一个源码工程。就是我就直接从这个给他扣的仓库直接给他克隆下来的。在这里的话我就直接去启动这个PMPDV，我就直接启动起来。

01:05:27

可以看到他已经应该启动起来，在这个3000端口，然后我们这里看一下怎么去连它，我们这里会比较简单一点，只需要一个web c web MCP client，然后传入这两个关键参数。这个在线押金为什么要用开始在家呢？因为如果你不盘一个固定的给他，那他就会每次给你随机来一个。对这种远程遥控可能不太友好。因为你远程遥控的话，你这个页面一刷新一次，然后这个在线ID就会变，可能会导致你远程遥控那个东西很快就连不上了。所以我们就需要一个固定的那固定的话这个东西就千万不能泄露给别人。然后

这里的话你拿到之后直接给存到lock service里面，然后拿到三星ID之后，我们就可以把这个固定的三星ID去跟跟我们这个数控端去进行一个进行一个操控。

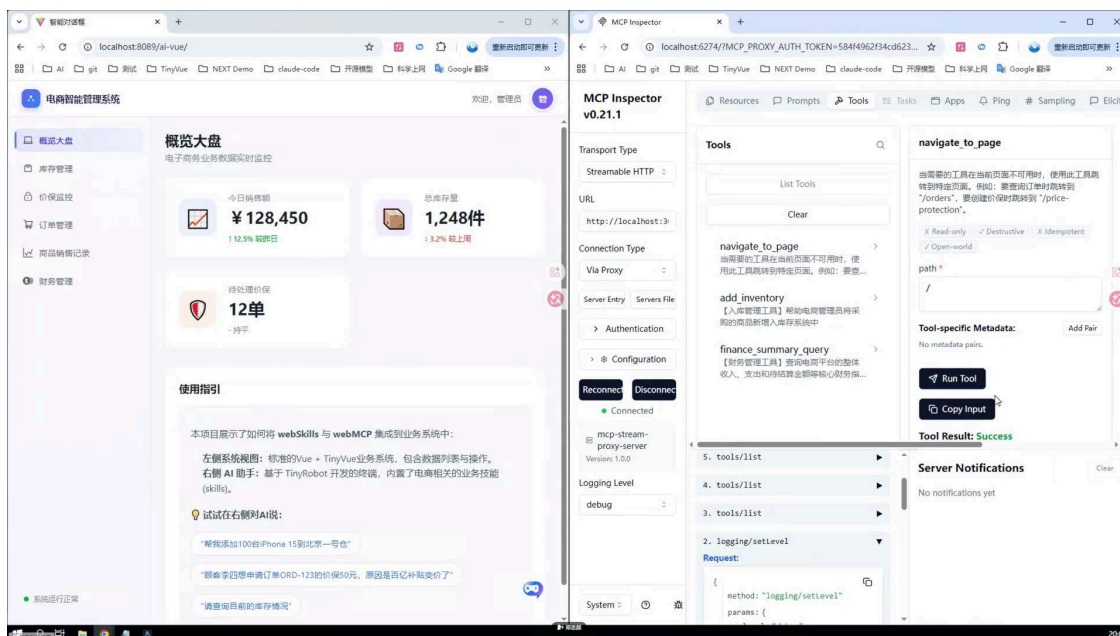


01:06:22

然后我们看一下这边应该提起来了，看一下，已经提起来了，我就多打开了一个。然后我们就把这个启动起来的，这个可能需要刷新一下，他应该没启动起来。我看一下F上它有没有启动，启动起来了。然后在这里我们就可以去拿着他这个好比他这个东西直接去这边去进行一个连接了。我看一下，三千，然后API然后把3000ID带上了905，然后我们就使用默认使用这个WHDB，然后进行一个连接。这里看到这边已经说明已经连上了，是这个握手，还有这个初始化都已经完成了。然后再看一下这个to，我们类似to这里就看到一个价包监控页面。这样的话我给大家分开去演示，这样的话大家就比较清楚一点，把这个关上，看这个能不能收回来。

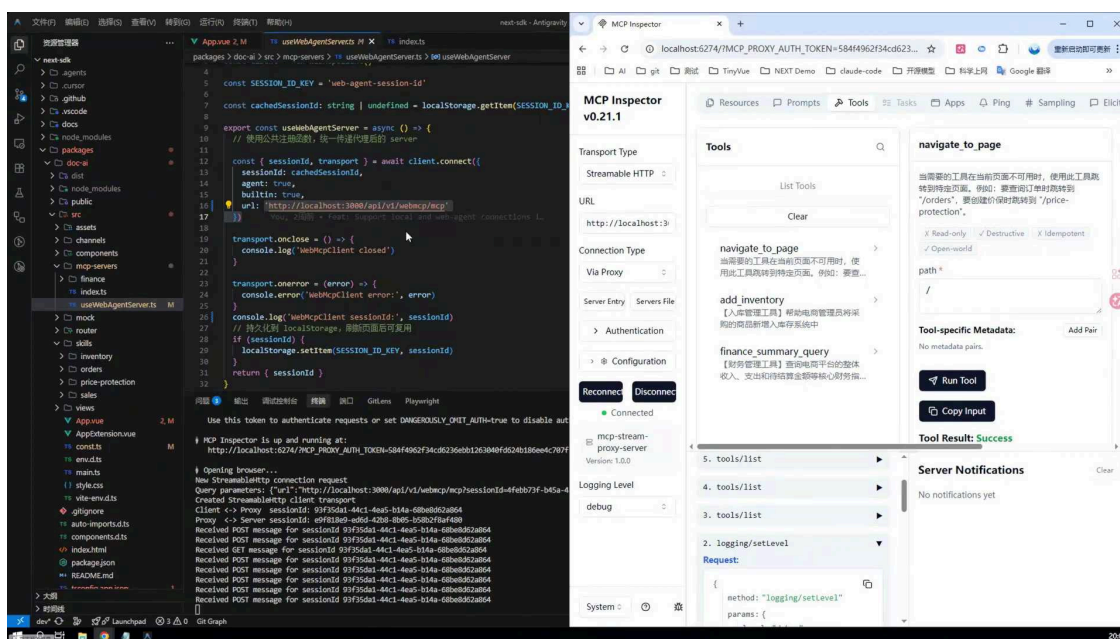
01:07:26

25把它缩小一点就行了。就比如说你看我这里，我就跳到这个首页，首页然后我罗列一下，然后重新去list to，然后你发现它现在只有两个，然后你跳到这个库存管理，我再考虑一下，再重新类似出，然后它这里就变成这么多。然后这里我就比如说我让他跳到一个，再做一个降保管理，降保管理页面。然后我就去点一下执行，你看到左边其实已经对应的已经跳转过去了。然后我们现在比如说他跳到首页去吧。所以这个就可以很明白的告诉大家，我们这个整体的nice DK加上我们这个外表景的，不光可以做到这种这种浏览器内部的这种速度很速度非常快，然后性能非常好的这种连接，就是使用这种浏览器内置的YMC，它的速度非常快。因为它整体是在GS内存里面去执行的，所以它没有任何的长链接，没有任何的一些等待，它就可以直接去做。然后如果你想需要远程遥控，那你就去打开我们开源的那个外半径的代码，然后自己去部署一下。



01:08:45

然后拿着这个部署的这个UI去不去配置到我们这个connect的方法上，你就可以直接用你的远程的。这里我演示的是这个inspect，大家可以用自己的AR编辑器比如说什么客户或者是cos空间，就是杜波。如果他支持这种远程的MCB工具，那都可以接入。你的接入的时候你就可以去做。并且我们这个是因为它是页面在不停切换，你的工具在不停的变化的。所以你这里我们也会使用一种通知的方式，比如说notification去通知你我们的工具发生变化的。然后你这边ID的按钮就需要实时刷工具，然后实时刷新工具，然后你就把这些工具传给那个。



01:09:37

行，然后整理这个其实配置起来就比较的简单，就是刚才那几步就可以了。然后我再说一下，这个往下说。然后我们这个技术升级的总结，就是我们的核心策略就是全面拥抱这个内置的。然后我们会提供poly fill去进行一个初始化函数的平滑平滑过渡。然后我们就可以把这个原生的外部MCP落地到你现有的一个生产环境，然后直接去你上生产，然后去做企业级的一个应用整改。未来的价值可能就是你未

来你不用担心这个API会对未来的浏览器不兼容。因为我们是完全去遵循这个内置的外面MCP的，所以你现在用，那你未来99%是不会过时的，而且并且是无缝迁移的。然后看一下大家有没有什么问题，想问。

决策指南：WebMCP 方案选型对比

原生内置 WebMCP (按需)	分离式全量注册 (全局)
 推荐场景： 中大型、业务逻辑复杂的系统	 推荐场景： 小型应用、功能单一工具
 幻觉风险： 极低 (工具随页面动态上线)	 幻觉风险： 中等 (工具全局常驻内存)
 路由依赖： 依赖 WebSkills 配置指引	 路由依赖： 依赖工具自身的 routeConfig
 核心优势： 上下文纯净，运行性能最优	 核心优势： 一站式注册，开发效率极高
 总结： 规模决定方案，按需方案更稳，全量方案更快	

技术升级总结：拥抱原生，展望未来

核心策略	实施路径	落地实践	未来价值
 全面拥抱浏览器原生标准 <code>navigator.modelContext</code> ，并结合 <code>WebSkills</code> 的智能引导，奠定高性能、高精度的交互基础。	 利用 <code>next-sdk</code> 的初始化函数实现平滑过渡，降低迁移成本。	 灵活选择“页面内按需注册”或“分离式全量注册”方案适配规模。	 标准化带来极致性能与低维护成本， <code>WebSkills</code> 的智能引导则开启了更广阔的人机交互想象空间。



01:11:00

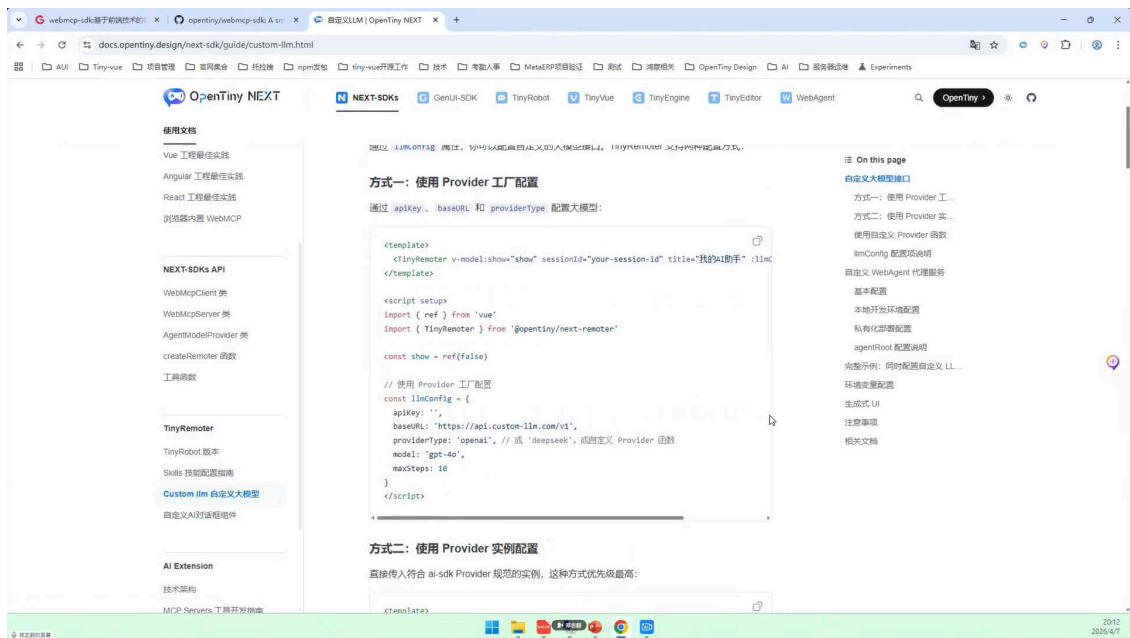
刚才景浩说的有个问题，就是我们的包头部署在OBOPS上，这个适合用MCP吗？如果你是一个前端应用的一个页面，那你就可以有多种途径。比如说第一种途径，你直接在你的JS内存里面去调调用工具。那你就直接比如说你的应用是左边是你的业务应用，右边可能你需要一个悬浮的一个按钮，就点开你的一个智能助手，那你的智能助手直接去帮你操作页面，那这个就是一个极速的一个GS内存里面的一个消息通信。那它的话就非常的快，就是可以做到零延迟，然后就速度快稳定。

01:11:49

然后如果你是需要你自已后台的外围的一个ID的去调用你前端的一些工具的话，那你就可可能就需要我们那个外表景的去做一个桥接，去做一个代理，这个两种方式都可以。当然你如果使用外表景的那你们的服务肯定需要自己部署。然后这个服务高可用，还有它的稳定性肯定也需要你们自己保障。

01:12:17

然后我再看一下这个，我看一下微信群里面的。我用turning pro做了一个AI agent的管理后台如何定制化开发聊天对接自已的后端聊天接口后端聊天接口我认为你可能说的是我们可能去对接自已自已的一个自已的一个比如说自已的一个cloud或者是其他的一个大模型。那在这里的话我们会有一个自定义customer这个LM自定义大模型的一个API。在这里的话我们其实我们底层用的是AISDK。AISDK可能就需要你去看一下ASDK的provider。因为它这个就很就就很方便的去对接各种各样的大模型。它的底层其实就抹平了，比如说OpenAI，什么GPT，然后各种不一样的模式它都会有提供的provide。

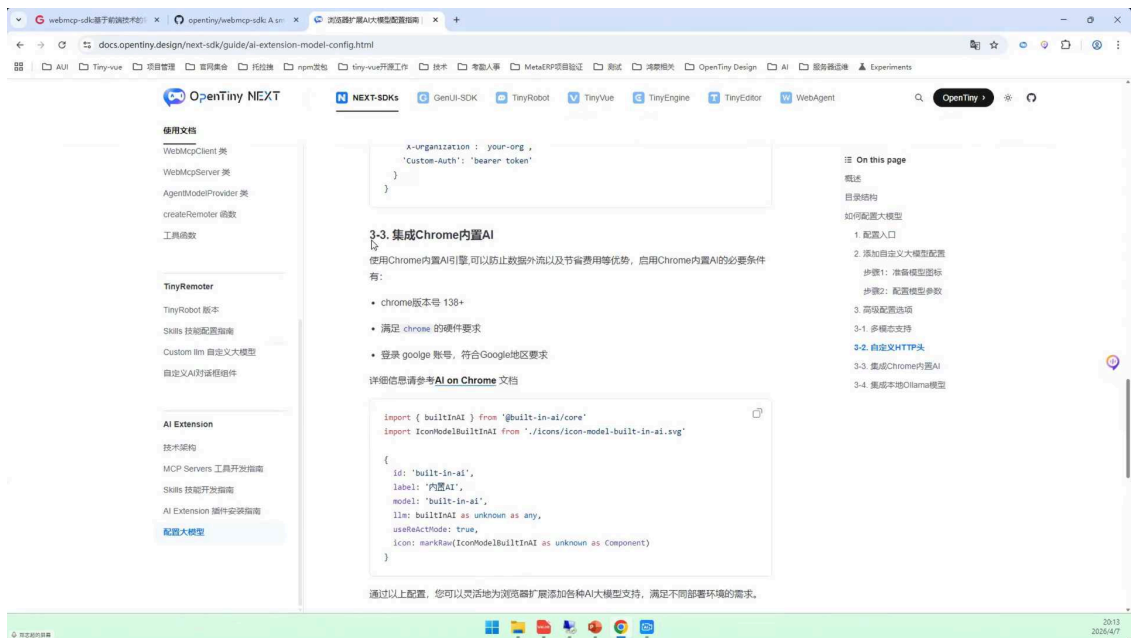


01:13:18

但是如果你是真正的你的AI的接口，是比如说自己培自己自己训练的，然后它的接口跟业界都不一样，那可能需要你自定义一个破坏的。你看我们这里会提供默认两种，一种是deep sig一种是open end就是Price type，你可以直接用。但如果你需要自己自己的一个provide的那你就可能需要provide函数从ASDK里面直接拿。比如说你可能需要一个自定义的，或者你需要一个。

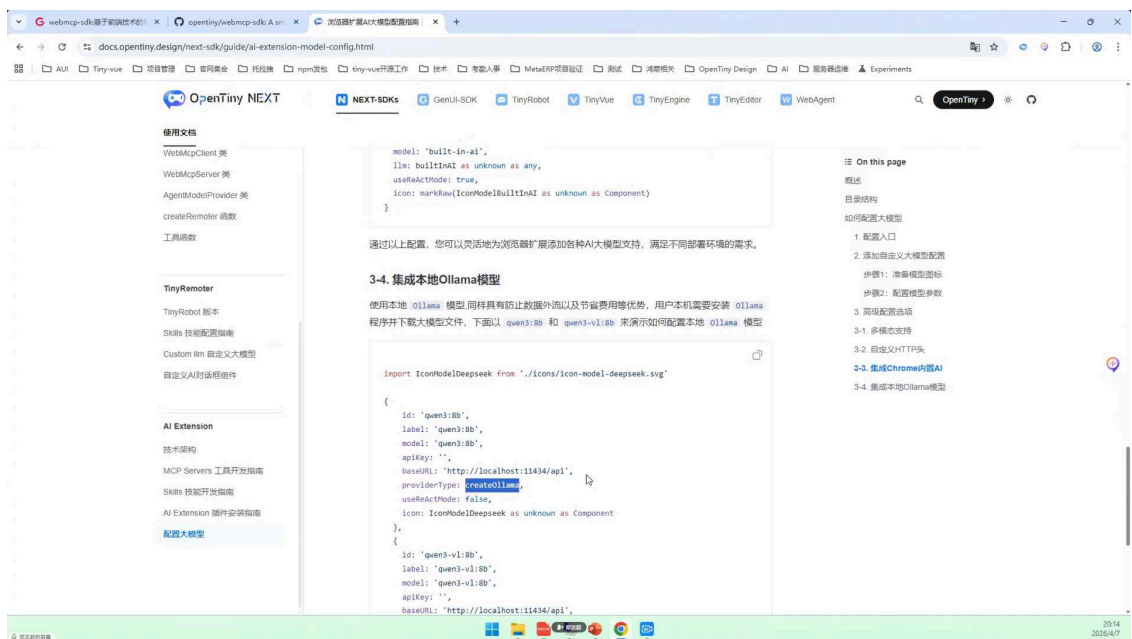
01:13:50

我看一下，还有一个，比如说这里面有一个配置的模型，这里面其实也有一个。看下这里，你看这里，其实我们还调动了这种谷歌内置的这种内置AI谷歌内置的。它那个叫他的那个L叫什么名之类的忘了，就是他自己内置的，就是现在他谷歌不光有内置的外面C它也有内置的。类似的AI的话，它用起来，虽然说它的它那个规模比较小，但是它用起来还是挺快的，他的速度会非常快，但当然他的能力不一定非常强，但他的速度确实非常快。做一些复杂的一些重复的操作才是可以的。所以我们这里其实你这样配的话，我们也支持那个内置的。



01:14:41

然后你只需要去引入这个第三方的一个build in AI这个其实也是AICK生态里面的一个谷歌内置的AI的一个provider，然后传进来我们就可以用了。比如说你自己本地部署了一个ollama的模型，那你就可以直接使用我们这个can career欧拉玛传过来，然后把那个把你的模型，把你的best UI的APIK传进来，然后你就可以直接去用了。就是说你现在想配置自己的大模型，那你就去找自己的pride。找不到那就没办法，你只能自己去自定义大模型，自定义provide。但是AICK应该囊括是99%，这个90%是所有的provide，你直接拿过来就用就行。

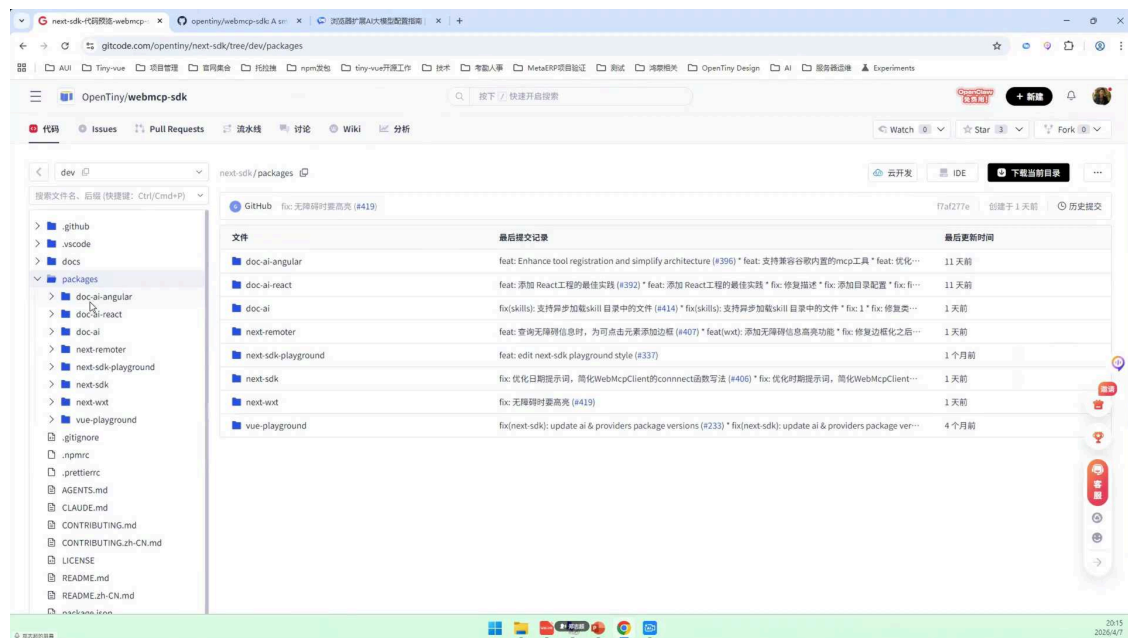


01:15:29

老项目VRVR能兼容吗？VR应该是可以的。首先我们nice DK它是一个与框架无关的。其实他你可以看到他这里给大家看代码就是SSCK这个包。他这里其实都没有用到各种什么Angella view或者是react这类东西，全部都是纯TS的来去说明。他这里是可以直接去在不同框架里面去用的，不管是V2V3、Angela还是react都可以用SAK。

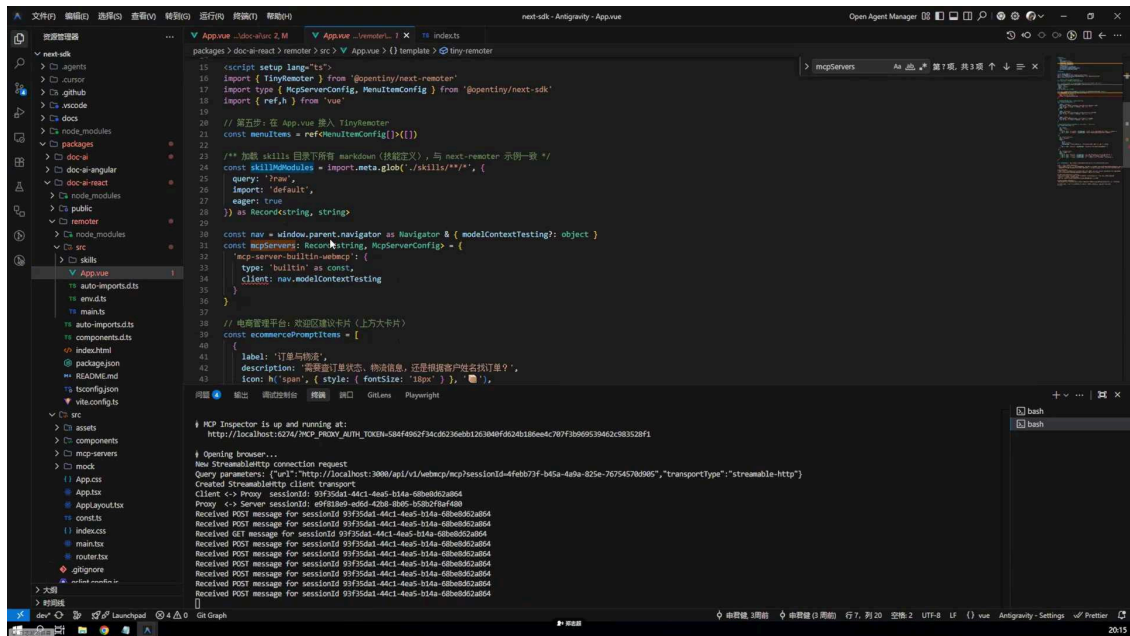
01:15:57

那remote呢怎么用？Remote怎么用？那就可以参考我们那个react和Angela的最佳时间文档。最佳时间的工程都在我们的get put里面的仓库里面，或者是get up仓库里。给大家看一下这这里面在这里packages里面就可以看到我们有这个dock AI Angela doc AI react，这两个就会是你想要的答案。因为它就是这种，因为我们我们那个remote是view 3的，所以它就是可以让你去借鉴参考的。然后这里面他可以看到这里面remote其实用的是一个I嵌进来的。



01:16:33

然后这里需要注意的一点，就唯一的一点就是其他的其实都一样。你这个service怎么传，还有这个怎么转都一样。但是唯一一点就是这里你需要去用温度parent去navigate，去拿你FM上一层的那个温度对象里面的这个navigate对象，然后传给我们。然后我们就直接可以调你的应用里面的这个原生的web MCP了。

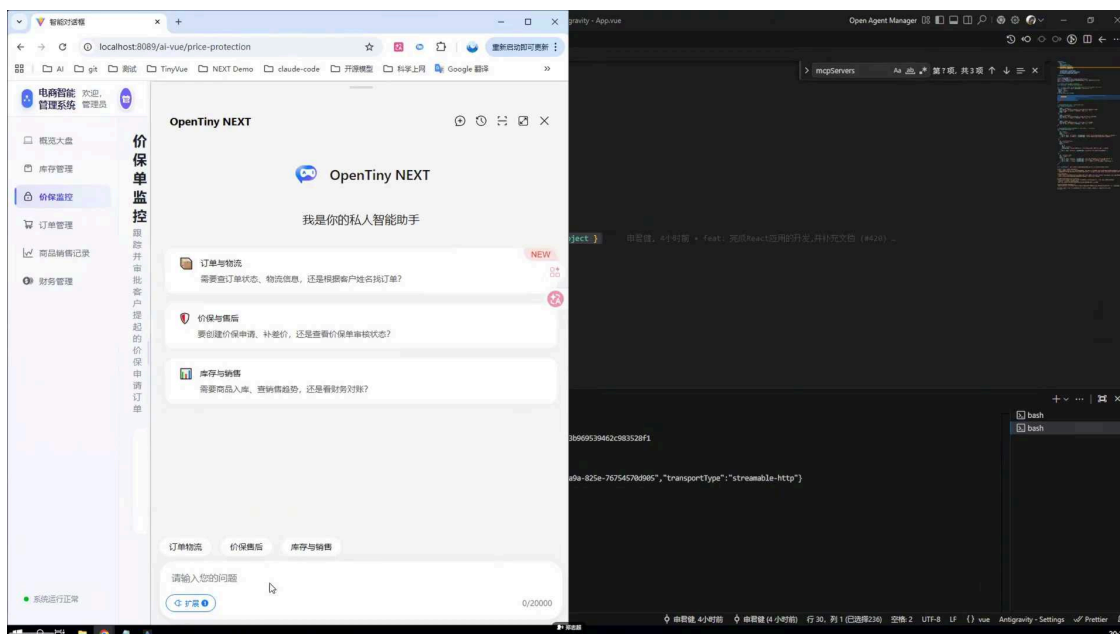


01:16:58

然后再看最后一个问题，就是使用第三方接入能否集成页面。比如说在open cloud接入页面，直接在open cloud显示调用的页面，使用第三方接入当然可以了。使用第三方接入就使用我们的这个外边景的等我集成页面。这个集成页面我没有听懂，可能你说的是不是生成式UI这种东西，生成式UI这个东西，那它可能就需要你你的agent自己去对接我们的一个生产计划。因为你agent没有这个能力，你我们就算给你吐出那种数据，你也渲染不出来。所以这块的话，因为我们下一期可能就是那个冠名那个同冠名同学给大家去分享一下这个甚至是UI这块的内容。所以他他可能会是你一个答案，就是你可能是不是希望中common cloud直接显示这个页面，这个UI比如说这个表单直接在在你的AR对话框里面去展示。

01:17:59

但是我们remote其实也是把这个生成优化一起集成进来了。这里我们其实remote是支持配置这个UIable那个属性。它配置成功之后，你只需要你的后台去接入他们的生成UI，你就可以直接在这里去，比如说帮我生成一个表单，就在这个对话框里面给你生成使用它就没有生成出来的一个表单，然后你就可以直接去用。当然我是我是这么觉得，就是深圳这句话可能是未来的一个终极方案。

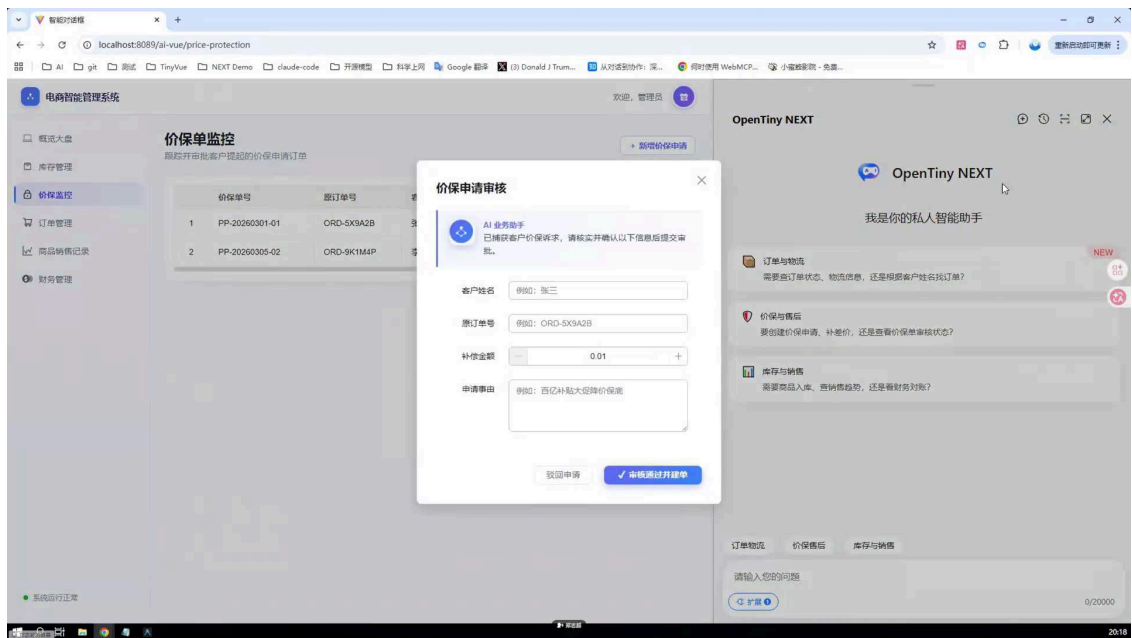
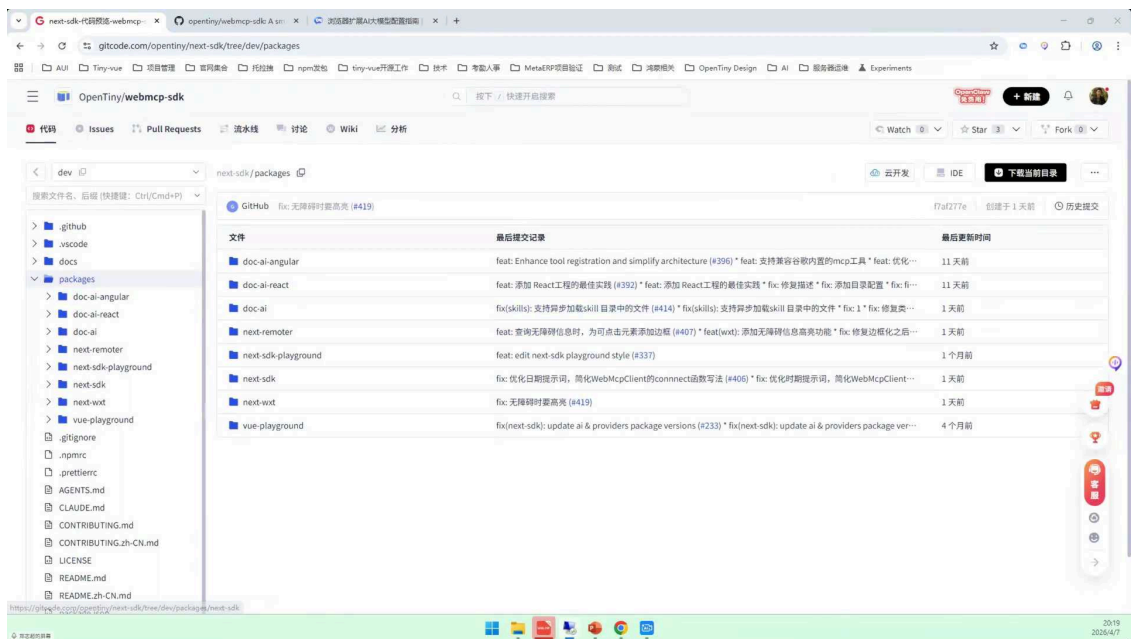


01:18:29

但是现阶段因为我们存在大量的这种大量的这种复杂的业业务应用，生成UI可能需要后台，包括前台前端，包括设计，都需要一个整体的一个大的改动。但是整体现在的存量的外部应用非常多。那你如果接入这个外部CP web skill，再加上外部，你就可以以很小的代价直接把你的应用改造成智能化应用。然后还有一个就是为什么你调MCP工具为什么不直接去后台调macd然后去操作数据库呢？这点可能跟我刚才说的也有关系，就是这种可能需要后台做大量的一些改造，就是这种安全性怎么保障。

01:19:09

还有一点最关键的一点是这个监管怎么监管。因为前台的话它的监管是非常的，就前端应用它的监管是非常的，怎么说呢？监管非常的成熟的，就现在的监管还有安全里头都已经做的非常好了。你审核你需要指定位置点，但是如果你后台去直接调M工具，那这个东西完全不可控。对于这些高危操作，复杂的高危操作，那那就是没有人为干预，那就可能就是黑匣子。你完全就无法去去了解它的整体的大概的一个走向。那可能会造成一种很难挽回的一个——一种一种局面。AI现在还没有做到那种让你完全可以百分之百放心的，所以需要现有的这个外部应用的智能化，后面可能就是需要这种生成优化的这种终极方案。当然这个都是一些我自己的一个看法，就这么多。



01:20:13

我我看内部好像也提了很多，我我我是可以继续这个时间够吗？我问一下时间够吗？还要继续回答吗？

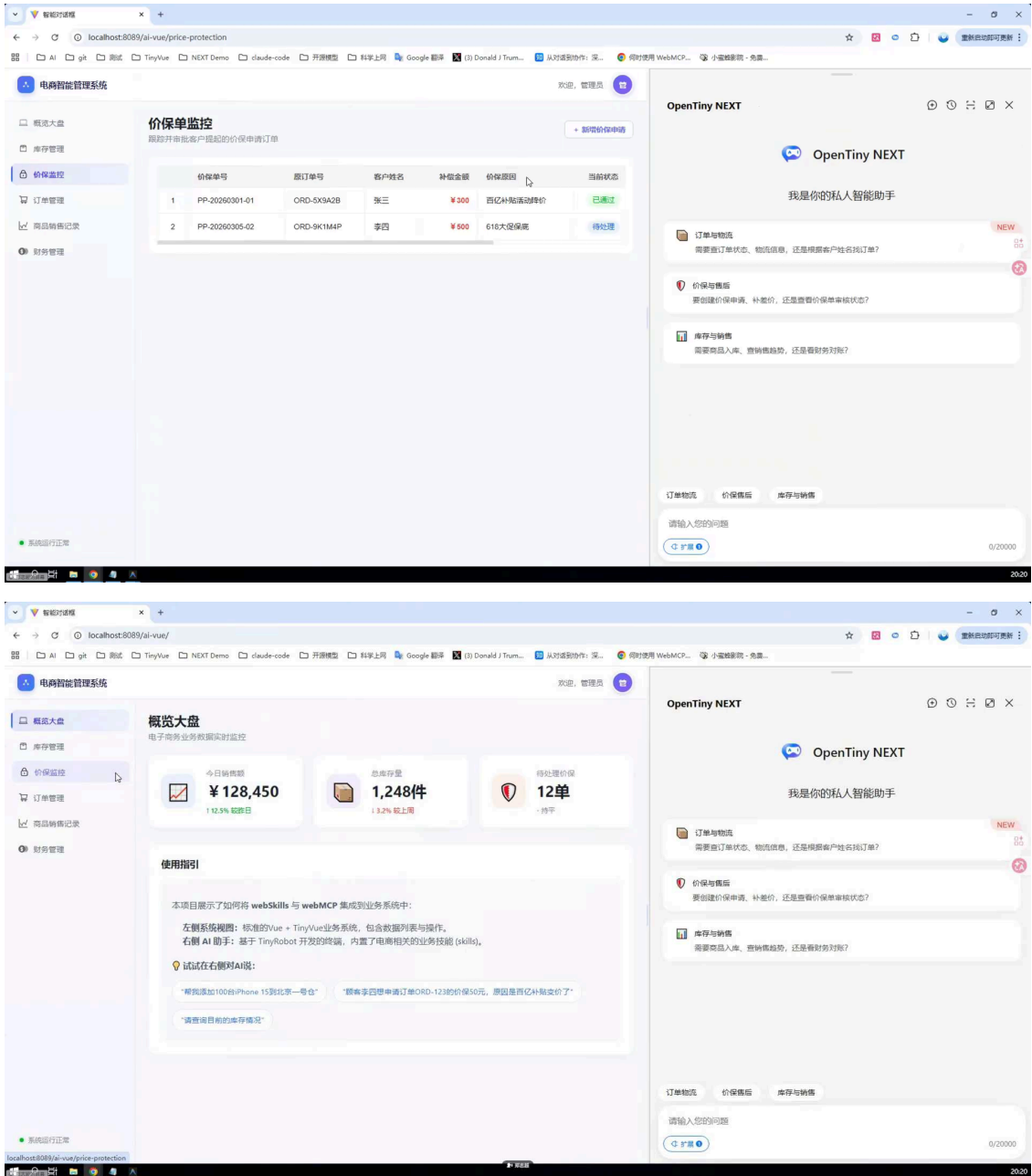
01:20:33

我要不然就再再回答几个，因为我可能怕回答晚了，大家这个时间拖的太长了，那我就挑几个。我看一下在上架商品时，是否会检查用户是否有上架商品的权限，这个问题就问的非常好。这种权限因为是什么呢？这种权限完全是你现有业务的，现有应用的一个基本权限。就是你没有接入我们web MCP之前，你的业务就已经有权限管理系统了。他能不能点那个菜单，他能不能点那个按钮，他已经有这个权限管理系统。

01:21:15

我们在前端操控这个外部的CP的，其实你后台，因为你的权限管理都在后台去校验的。前台校验可能直接简单的权限校验，但是你最终的权限你肯定都在后台。所以在前台发出的请求都是以你当前登录这

个登录人员的一个权限去操作的。你没有这个权限你就调不通，调不通你就没法操作，包括前端的一些权限点也是一样。比如说你页面里面没有这个按钮，那你去调这个MC工具的时候，你怎么去调呢？就可能你你可能你调的时候你直接就给他拦截了，就是说明你没有这个权限，就是前端的逻辑直接给你卡死了。后台的话就更不用说了，它你这个登录人权限是什么样，你后台它原本就是什么逻辑了。



01:22:02

如果用户误操作了上架商品是能否撤销上架的功能，这块就是不是我们外部MCB需要考虑的。因为这块其实都是业务的一些功能，就是你你业务里面支不支持撤销上架，或者支不支持查看历史记录，能不能恢复，这些都是业务的应用。就是我们只提供能力，我们提供能力，并且我们提供了一个最佳实践方案，就是告诉你们这一套是可行的。然后具体这个业务怎么去运转，那就是你们自己去写代码，然后去考虑的。市里有开源代码，这里的话会有个get code，还有我们这边也会有同步的get hf这两个都有。然后刚才说的那几个应用我也给大家看了，就是在这里三个智能应用的最佳实践。并且我们还有对应的

文档，文档里面也会说明这个是view的，这个是安格拉最佳实践，然后react最佳实践，后面可能会有其他框架的，到时候我们也会补充。

01:23:02

然后这里会介绍一下内置的外部MC内置的工具注册方式。呃。会不会涉及什么安全性问题？刚刚那个是一样的，安全性问题完全由你以前的业务应用，前端应用去保障。你因为你们现在如果是已有的业务应用，安全性已经是已经做好了，那我们外部MCP都直接沿用你们之前的一个权限，一个安全的一个设计。所以说这块就不需要怎么去改动，你能干什么，你就原本能干什么，你现在就能干什么。

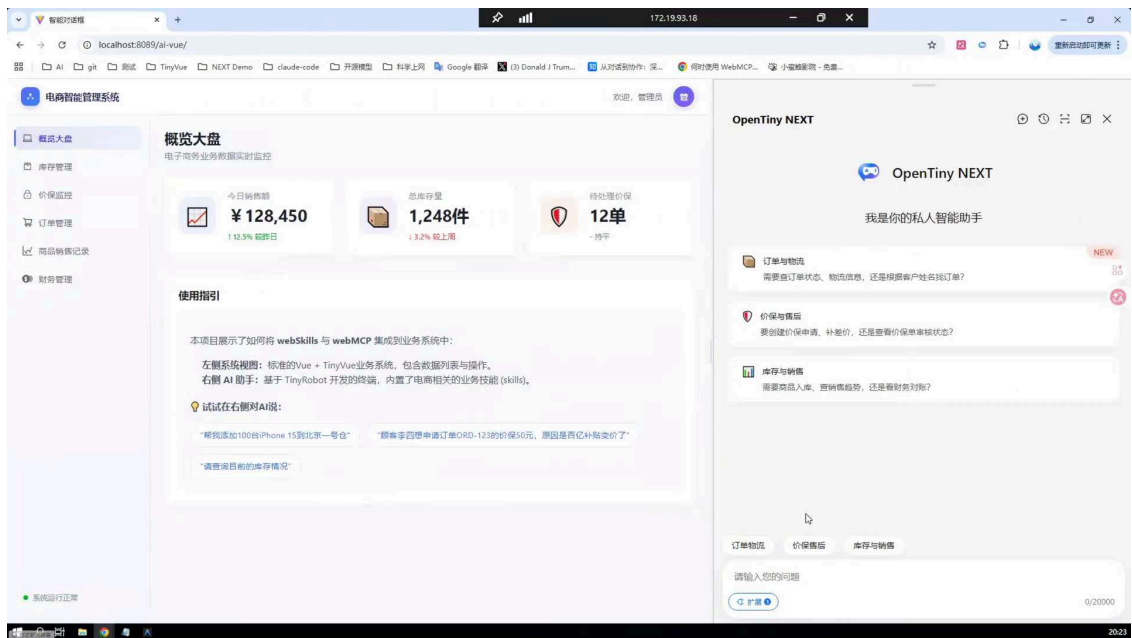
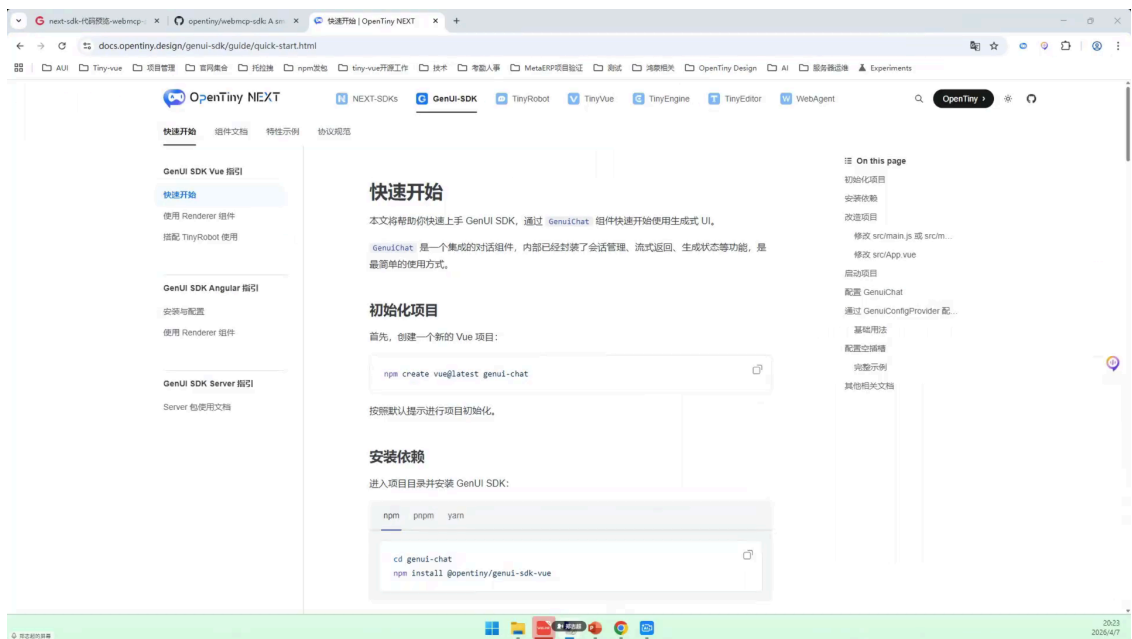


01:23:42

我目前有一个需求是获取当前页面的表格数据，并且让AI给它生成可视化图。图表可以使用这个外表，这个当然是可以了。那看你这个图表要生成在哪？那你如果是这样，我们有两种方案。第一种是你直接读你的工具那个表格数据，表格数据当然是现在已经有了，刚才也展示出来。你直接读读哪，读什么数据都可以读到。

01:24:07

然后AI生成可视化图表，你可以在你的外部应用里面打开一个页签去。比如说在这里我切换一个页签，然后这里去添加图标也可以。你也可以直接在这个AR对话框里面去使用我们的生成式UI去渲染你的图表。因为我们会内置turning下的这个图表，然后直接去渲染图表，所以这两种方法都可以。所以你那你想到的每一种方式我们都可以做得到。



01:24:32

然后必须要求cream版本做什么？这块我刚才说的就是我们有关是垫片，它是对版本没有要求的，它是完全兼容各种低版本的。要想要减少幻觉的产生，有没有提供skill的编程教编写教程。这个skill编程教程我们这里应该佩服，你可以去参考一下业界有没有什么好的一些教程。

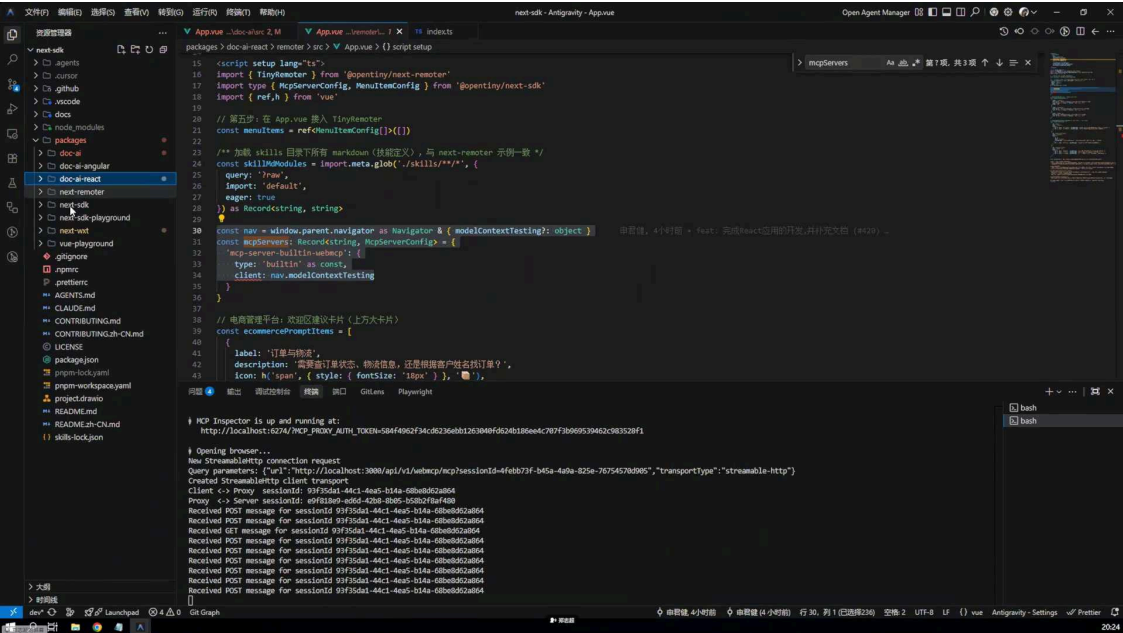
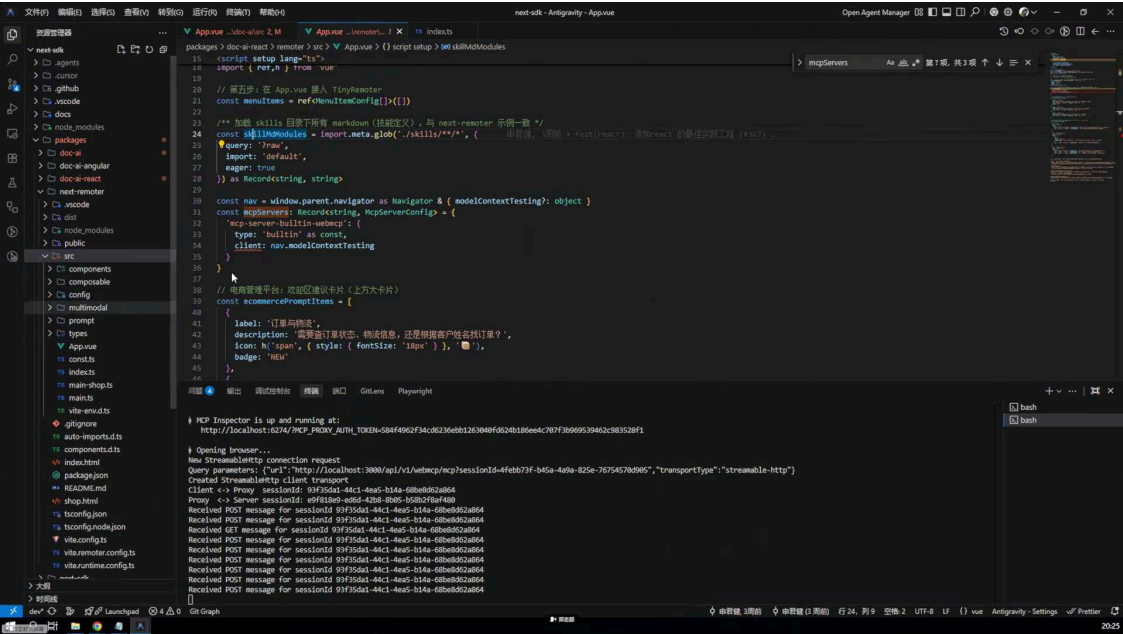
01:25:03

然后最后一个问题，我就说最后个问题，就说完就不说了。然后看一下这个问题。我们的产品有自己的AR助手页面，支持在自己的助手中调用吗？还是只能在open name注册？当然可以在这里助手了。就是我刚才说的，就是其实这个助手里面的一些逻辑，你们是直接可以抄过去用的。

01:25:32

就比如说我这是我们这是我们整整个我看一下我们整个react remote。然后这是我们其实说是我们刚才那个仓库，就get code里面仓库，你直接找到这个packages，然后里面的next remote，这里就是我们那

个我们那个AR对外框的一个源码。那这里的话你就可以直接去参考我们的源码，直接去对接。比如说他直接什么读书的，你就直接把这套逻辑搬过去，还是直接怎么去对接这个外部MCP的，你也可以直接抄过去。这个源码都已经开发出来了，那要怎么用就看你们怎么用，或者说你可以基于我们这个next remote去改造成你们的。对，对，我方也行。因为我们那个是一个组件，这个组件可以配置各种插槽，各种属性，然后去定制自己想要的这种样式都是可以的。



01:26:27

然后比如说你还有一些问题，关于甚至是UI的，关于外部MC的，关于外观检测，可以去加群，加我们的小助手。然后。跟我们后续继续交流。行，那我这块就没了。好。

