腾讯云智能体开发平台

API 接入操作指南

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

API 接入操作指南

概览

最近更新时间：2026-07-13 16:32:01

本文档介绍如何通过 API 在代码中构建、配置并接入 Claw 模式应用——将创建空间、创建应用、配置 Agent、发布、对话、获取产出的完整流程集成到自有系统中，无需依赖控制台的可视化操作。
平台提供两种接入方式，可按需选择或组合使用：

接入方式	说明	适用场景
控制台可视化	在控制台中通过界面搭建、配置与调试智能体应用	快速验证、非开发人员配置
API 接入	以代码方式完成全流程，并嵌入自有系统	自动化、批量管理、集成到已有产品

本文档聚焦 API 接入。若希望快速跑通，可直接查阅 从零搭建一个 Claw 模式应用 。

模式介绍

本文档所述的 API 接入基于 Claw 模式，具备以下核心能力：

- **沙箱工作空间：每个会话绑定独立沙箱，Agent 可执行代码、生成文件。**
- **用户级 Agent：为每个用户复刻独立的 Agent 实例，配置互不干扰。**
- **动态能力切换：运行时可动态更换模型、安装和卸载 Skill、绑定工具，无需重新发布。**
- **多 Agent 协作：主 Agent 可调度子 Agent 协同完成复杂任务。**

核心概念

通过 API 会接触到以下核心对象：

概念	说明
空间（Space）	一个工作空间代表一个独立的工作区域，空间成员可以共享其内部的数据资源，不同工作空间之间数据隔离。
应用（Application）	一个可对外提供服务的智能体应用。
Agent	应用内组成部分，承载指令、模型、Skill、工具等具体能力配置。
对话（Conversatio	用户与应用交互的会话实例，同一会话内多轮对话共享上下文与工作空间。

n）	

Agent 的能力可通过以下资源扩展，且支持动态加载（均为可选）：

概念	说明
模型（Model）	Agent 使用的推理大模型，决定其语言理解与生成能力。
Skill	平台内置或自定义的技能模块（如代码执行、文件处理、联网搜索等）。
插件（Plugin）	一组相关工具的集合，代表一个外部服务的接入（如"天气服务"插件），需在平台注册并配置认证方式。
工具（Tool）	插件下的具体可调用能力（如"查询天气"工具），每个工具有明确的输入/输出参数定义。
连接器（Connector）	连接外部数据源或服务的桥梁。
变量（Variable）	运行时透传给 Agent 的自定义上下文信息。

> **说明：**
> **Plugin 与 Tool 的关系**：ADP 采用「平台注册 Plugin → 显式绑定 Tool 到 Agent」的模式。一个 Plugin 代表一个外部服务，包含一个或多个 Tool；Agent 需要**显式声明**使用哪些 Plugin 及其下的哪些 Tool，并可对每个 Tool 的参数、认证方式做精细控制。

开发者与使用者

API 接入涉及两种角色：

角色	说明	鉴权方式
开发者（Developer）	调用 API 创建、配置、发布应用的人	腾讯云 `SecretId` / `SecretKey`（V3 签名）；独立站用户使用 ADP 密钥管理中的 `SecretId` / `SecretKey`。
使用者（User）	最终与 Agent 对话的终端用户	应用 `AppKey` + `UserId` 标识。

开发者通过管理类接口（`CreateApp`、`ModifyAgent`、`CreateRelease` 等）构建和发布应用；使用者通过对话类接口（`CreateConversation`、`/adp/v2/chat`）与应用交互。两者的权限和操作范围完全隔离。

应用与 Agent 的关系

一个应用可以包含**一个主 Agent**，每个主 Agent 下可以挂载**多个子 Agent**：

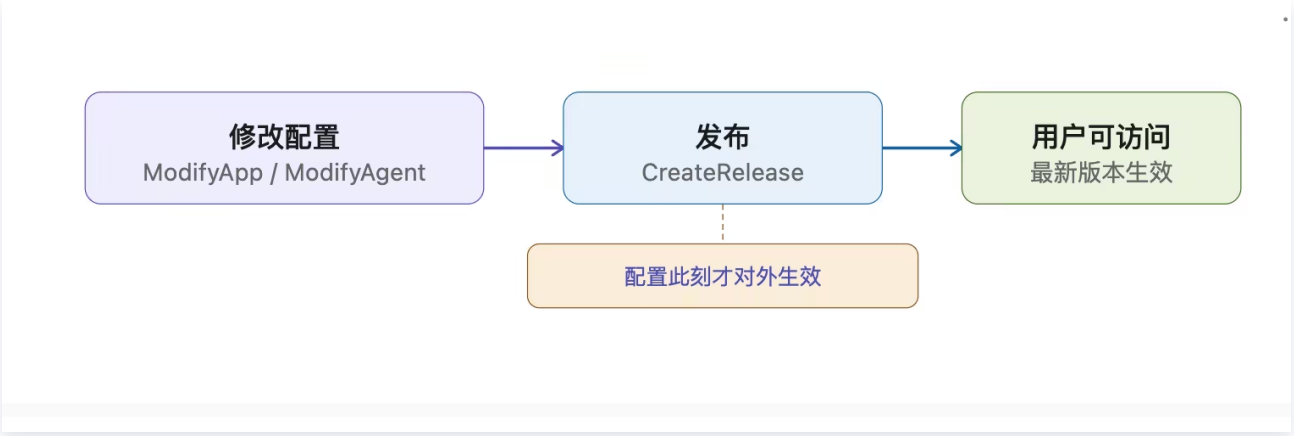
- **主 Agent**：直接挂载在应用下，是用户对话的入口
- **子 Agent**：挂载在主 Agent 下，可被主 Agent 调度协作

应用级配置（开场白、记忆、联网搜索等）对所有 Agent 通用；Agent 级配置（指令、模型、Skill、工具）则各自独立。

应用与发布的关系

应用的配置修改**不会立即对外生效**，必须通过**显式发布**才能让用户访问到最新版本。这种设计保证了：

- **配置安全**：避免未经验证的改动直接影响线上用户。
- **版本管理**：每次发布生成一个版本快照，可追溯。



开发完整链路

从开发到使用，完整链路如下：

说明：
一个应用可包含多个主 Agent（每个主 Agent 下可再挂多个子 Agent），在步骤 ③ 中定义，并随应用一起发布。

适用场景

- 需要文件处理、代码执行、产出物管理的复杂智能体。
- 需要调用外部工具 / API 的任务型智能体。
- 需要结合企业专属数据（知识库 / RAG）的智能问答。
- 需要嵌入到已有产品（Web / 小程序 / App）的对话能力。

API 版本与端点

接入方式	接口请求域名	密钥获取
腾讯云	`adp.tencentcloudapi.com`	腾讯云 CAM 控制台
独立站	`capi.adp.tencent.com`	ADP 控制台 > 密钥管理

腾讯云

- **API 版本：** 2026-05-20
- **接口请求域名：** `adp.tencentcloudapi.com`
- **鉴权方式：** 腾讯云 V3 签名（`SecretId` / `SecretKey`）；对话流接口使用应用 `AppKey` 鉴权
- **密钥获取：** 腾讯云 CAM 控制台
- **对话端点（SSE）：** `https://wss.lke.cloud.tencent.com/adp/v2/chat`

独立站

- **API 版本：** 2026-05-20
- **接口请求域名：** `capi.adp.tencent.com`
- **鉴权方式：** 腾讯云 V3 签名（`SecretId` / `SecretKey`）；对话流接口使用应用 `AppKey` 鉴权
- **密钥获取：** ADP 控制台 > 密钥管理
- **对话端点（SSE）：** `https://adp.tencent.com/adp/v2/chat`

从零搭建一个 Claw 模式应用

最近更新时间：2026-07-13 16:32:01

本文用 ADP API 从零跑通一个 Claw 模式应用的完整生命周期：
创建空间 → 创建应用 → 创建 Agent → 配置 → 发布 → 创建会话 → 发起对话 → 获取产出

前置条件与调用方式

1. 准备密钥 `SecretId` / `SecretKey`（用于接口的 V3 签名）：

接入方式	密钥获取	请求地址
腾讯云	腾讯云 CAM 控制台	`adp.tencentcloudapi.com`
独立站	ADP 控制台 > 密钥管理	`capi.adp.tencent.com`

2. 安装依赖并初始化客户端：

```
# 1) 安装 ADP 专属 SDK：pip install tencentcloud-sdk-python-adp requests
sseclient-py certifi
# 2) 配置密钥环境变量：TENCENTCLOUD_SECRET_ID / TENCENTCLOUD_SECRET_KEY
# 3) 初始化 AdpClient（自动签名），后面每步都用它调用
import os
from tencentcloud.common import credential
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.adp.v20260520 import adp_client, models

cred = credential.Credential(
    os.environ["TENCENTCLOUD_SECRET_ID"],
    os.environ["TENCENTCLOUD_SECRET_KEY"],
)
# 腾讯云用户使用 adp.tencentcloudapi.com；独立站用户替换为
capi.adp.tencent.com
profile =
ClientProfile(httpProfile=HttpProfile(endpoint="adp.tencentcloudapi.com"
))

client = adp_client.AdpClient(cred, "ap-guangzhou", profile)
```

步骤 1：创建空间

调用 CreateSpace 。

> **说明：**
> 也可以直接使用系统内置的 `default_space` ，跳过本步。

```
req = models.CreateSpaceRequest()
req.Name = "我的空间"
req.Description = "空间描述"
resp = client.CreateSpace(req)
space_id = resp.SpaceId
print(space_id)    # 例：UYiGYydT
```

如果没有创建空间的权限，可以先调用 DescribeSpaceList 查询已有空间列表，选择一个已有空间：

```
req = models.DescribeSpaceListRequest()
req.Query = ""
resp = client.DescribeSpaceList(req)
space_list = resp.SpaceList
print(space_list)    # 例：[{'SpaceId': 'UYiGYydT', 'Name': '我的空间',
'Description': '空间描述'}]
space_id = space_list[0].SpaceId
```

步骤 2：创建应用

调用 CreateApp ， `AppMode` 传 `4` （Claw 模式）。
AppMode 枚举值：

值	模式	说明
1	标准模式	知识问答、RAG 场景
2	Agent 模式	自定义工具调用
3	单工作流模式	固定流程驱动
4	Claw 模式	带沙箱工作空间的自主 Agent

```
req = models.CreateAppRequest()
```

```
req.SpaceId = space_id
req.AppMode = 4                # 4 = Claw 模式
req.Name = "数据分析助手"
req.Description = "我的 Claw 应用"
resp = client.CreateApp(req)
app_id = resp.AppId
print(app_id)
```

步骤 3：创建 Agent

应用和 Agent 是两种不同的资源——应用是对外服务的载体，Agent 是承载具体能力的单元。用 CreateAgent 创建 Agent，可一次性设定指令、模型、Skill：

Kind 枚举值：

值	含义
0	配置端 Agent（创建应用时使用）
1	用户级 Agent（由 `CopyAgentFromApp` 复制生成，用于动态配置场景）

相关查询接口：

- DescribeModelList（获取可用模型列表，`ModelScene=18` 为 Claw 模式可用模型）。
- DescribeSkillSummaryList（获取可用 Skill 列表）。

```
# 3.1 拉取 Claw 场景可用模型
req = models.DescribeModelListRequest()
req.SpaceId = space_id
req.ModelScene = 18            # 18 = Claw 模式可用模型
req.PageNumber = 0
req.PageSize = 20
resp = client.DescribeModelList(req)
model_id = resp.ModelList[0].ModelBasic.ModelId

# 3.2 （可选）拉取可用 Skill
req = models.DescribeSkillSummaryListRequest()
req.SpaceId = space_id
resp = client.DescribeSkillSummaryList(req)
skill_id = resp.SkillSummaryList[0].SkillId

# 3.3 创建配置端主 Agent，一次性设定指令 / 模型 / Skill
req = models.CreateAgentRequest()
```

```
req.AppId = app_id
req.Kind = 0                              # 配置端 Agent
req.Agent = models.AgentConfig()
req.Agent.Profile = models.AgentProfile()
req.Agent.Profile.Name = "数据分析主 Agent" #APP名称在空间内唯一
req.Agent.Profile.Role = 0            # Role 0 = 主 Agent
req.Agent.Instructions = "你是一个数据分析助手，擅长生成报表。"
req.Agent.Model = models.AgentModelConfig()
req.Agent.Model.ModelId = model_id
req.Agent.Model.ContextWordsLimit = 20000
req.Agent.Model.InstructionsWordsLimit = 20000
req.Agent.SkillList = [skill=models.AgentSkill()skill.SkillId=skill_id]
req.Agent.AdvancedConfig = models.AgentAdvancedConfig()
req.Agent.AdvancedConfig.MaxReasoningRound = 100
resp = client.CreateAgent(req)
agent_id = resp.AgentId
```

步骤 4：修改配置（可选）

配置分为**应用级**和 **Agent 级**两个层级。应用级配置是所有 Agent 共享的通用设置，Agent 级配置是单个 Agent 独有的能力设置：

层级	包含内容	接口
应用级	开场白、记忆、联网搜索、数智人、体验设置等（所有 Agent 共享）	ModifyApp
Agent 级	指令、推理模型、Skill、工具、插件（单个 Agent 独有）	ModifyAgent

两者都用 `UpdateMask.Paths` 局部更新，只改指定字段：

```
# 改 Agent 的指令（Agent 级）
req = models.ModifyAgentRequest()
req.AppId = app_id
req.AgentId = agent_id
req.Agent = models.AgentConfig()
req.Agent.Instructions = "你是一个严谨的数据分析助手。"
req.UpdateMask = models.FieldMask()
req.UpdateMask.Paths = ["Instructions"]
client.ModifyAgent(req)
```

```
# 改应用开场白（应用级）
req = models.ModifyAppRequest()
req.AppId = app_id
req.Config = models.AppConfig()
req.Config.Greeting = models.AppGreetingConfig()
req.Config.Greeting.Greeting = "你好，我是数据分析助手～"
req.UpdateMask = models.FieldMask()
req.UpdateMask.Paths = ["Config.Greeting.Greeting"]
client.ModifyApp(req)
```

UpdateMask 支持的字段：

接口	支持的字段
`ModifyAgent`	`Profile.Name`、`Instructions`、`Model`、`ToolList`、`PluginList`、`SkillList`、`AdvancedConfig` 等
`ModifyApp`	`Name`、`Config.Greeting`、`Config.Model`、`Config.Memory`、`Config.Experience` 等应用级字段

> ⚠ **注意：**
> 修改配置后**不会立即生效**，必须执行 步骤 5：发布应用 后，改动才对外服务可见。

打开「允许在对话中动态修改配置」开关

运行时动态配置：如果你希望**每个用户/每次对话都能动态换模型、装卸 Skill、透传自定义变量**（而不是所有人共用发布时的固定配置），需要在应用的**高级设置**里打开「允许在对话中动态修改配置」开关。
打开后才能在 `CreateConversation` 里传用户级 `AgentId`、在对话请求里注入运行时配置。完整用法见 动态修改 Agent 配置。

步骤 5：发布应用

配置只有**发布**后才对外生效。调用 CreateRelease 触发发布，再用 DescribeReleaseSummary 轮询发布状态直到完成，最后通过 DescribeApp 获取对话所需的 `AppKey`。
发布状态枚举：

枚举值	状态
1	发布中
2	排队中

3	发布成功
4	发布失败

```
req = models.CreateReleaseRequest()
req.AppId = app_id
req.Description = "首次发布"
resp = client.CreateRelease(req)
release_id = resp.ReleaseId
print("ReleaseId:", release_id)

# 发布是异步任务，轮询直到发布完成
import time
while True:
    req = models.DescribeReleaseSummaryRequest()
    req.AppId = app_id
    req.ReleaseId = release_id
    resp = client.DescribeReleaseSummary(req)
    status = resp.ReleaseSummary.Status
    if status == 3:
        print("发布成功")
        break
    elif status == 4:
        print("发布失败")
        break
    time.sleep(2)

# 获取 AppKey（对话时需要）—— 通过 FieldMask 指定只返回 SecretInfo 字段
req = models.DescribeAppRequest()
req.AppId = app_id
req.FieldMask = models.FieldMask()
req.FieldMask.Paths = ["SecretInfo"]
resp = client.DescribeApp(req)
APP_KEY = resp.App.SecretInfo.AppKey
print("AppKey:", APP_KEY)
```

步骤 6：创建会话

调用 CreateConversation 创建一个会话，拿到 `ConversationId`：

- `AgentId`（可选）：打开「允许在对话中动态修改配置」开关后可传，用于绑定一个用户级 Agent。详情请参见 动态修改 Agent 配置。
- 会话是后续所有动作的引用句柄；同一 `ConversationId` 内多轮对话共享上下文与工作空间。

Type 枚举值：

枚举值	含义
5	API 接入

```
# APP_KEY 已在 步骤 5 通过 DescribeApp 获取
USER_ID = "user-001"           # 你系统里的用户唯一标识

req = models.CreateConversationRequest()
req.Type = 5                   # API 接入
req.AppId = app_id
req.AppKey = APP_KEY
req.UserId = USER_ID
resp = client.CreateConversation(req)
conversation_id = resp.ConversationId
```

步骤 7：发起对话（SSE）

对话流接口 与腾讯云接口不同：走独立域名、用 `AppKey` 鉴权、以 SSE 流式返回（无需 V3 签名）。

接入方式	对话端点
腾讯云	`https://wss.lke.cloud.tencent.com/adp/v2/chat`
独立站	`https://adp.tencent.com/adp/v2/chat`

```
import json
import requests
import sseclient   # pip install sseclient-py

# 对话端点（根据接入方式选择）
# 腾讯云：https://wss.lke.cloud.tencent.com/adp/v2/chat
# 独立站：https://adp.tencent.com/adp/v2/chat
CHAT_ENDPOINT = "https://wss.lke.cloud.tencent.com/adp/v2/chat"

body = {
```

```
    "RequestId": uuid.uuid4().hex
    "ConversationId": conversation_id,    # 必填，缺了直接被拒（400）
    "AppKey": APP_KEY,
    "VisitorId": USER_ID,
    "Contents": [{"Type": "text", "Text": "请生成这个季度的 mock 销售数据，然后分析销售数据并生成 Excel 报告"}],
    "Incremental": True,
    "EnableMultiIntent": True,
    "Stream": "enable",
}

with requests.post(CHAT_ENDPOINT, json=body,
                   headers={"Accept": "text/event-stream"}, stream=True)
as r:
    for sse_event in sseclient.SSEClient(r).events():
        event = json.loads(sse_event.data)
        if event["Type"] == "text.delta":
            print(event["Text"], end="", flush=True)
        elif event["Type"] == "response.completed":
            break
```

响应（SSE 流）

```
data: {"Type":"request_ack","RequestId":"..."}
data: {"Type":"response.created","RecordId":"r1"}
data: {"Type":"message.added","MessageId":"m1","MessageType":"thought"}
data: {"Type":"text.delta","MessageId":"m1","Text":"先读取数据..."}
data: {"Type":"message.done","MessageId":"m1"}
data: {"Type":"message.added","MessageId":"m2","MessageType":"reply"}
data: {"Type":"text.delta","MessageId":"m2","Text":"分析完成。"}
data: {"Type":"response.completed"}
```

拉取历史消息

用 DescribeConversationMessageList 获取完整历史：

```
req = models.DescribeConversationMessageListRequest()
req.ConversationId = conversation_id
req.UserId = USER_ID
```

```
req.AppKey = APP_KEY
req.Type = 5
req.Limit = 10
resp = client.DescribeConversationMessageList(req)
history = resp.MessageList
```

步骤 8：获取产出文件

Claw 模式下，Agent 生成的文件（Excel、图片、代码等）会以 **COS 文件地址**的形式包含在对话消息的 `Contents` 中。当 `Content.Type = "file"` 时，通过 `Content.File.FileUrl` 即可获取文件的 COS 下载地址。

FileInfo 数据结构：

字段	类型	说明
`FileName`	String	文件名称
`FileUrl`	String	文件 COS 下载地址
`FileSize`	String	文件大小
`FileType`	String	文件类型

在 步骤 7 的 SSE 流中，当 Agent 产出文件时，`message.done` 事件的 `Contents` 里会包含 `Type = "file"` 的内容项，从中即可提取 COS 下载地址：

```
# 在 SSE 流处理中，当收到 message.done 事件时提取文件
if event["Type"] == "message.done":
    message = event["Message"]
    for content in message.get("Contents", []):
        if content["Type"] == "file":
            file_info = content["File"]
            print(f"文件：{file_info['FileName']}")
            print(f"下载地址：{file_info['FileUrl']}")
            # 可直接用 requests.get(file_info['FileUrl']) 下载
```

全流程接口一览

步骤	接口
创建空间	CreateSpace / DescribeSpaceList（查询已有空间）

创建应用	CreateApp（`AppMode=4`）
创建 Agent	CreateAgent + DescribeModelList（获取可用模型）+ DescribeSkillSummaryList（获取可用 Skill）
修改配置	ModifyApp（应用级）+ ModifyAgent（Agent 级）
发布应用	CreateRelease + DescribeReleaseSummary（轮询状态）+ DescribeApp（获取 AppKey）
创建会话	CreateConversation
发起对话	POST /adp/v2/chat（SSE 流式对话）
获取产出文件	从 SSE 流的 `message.done` 事件中提取文件 COS 地址，直接下载

动态修改 Agent 配置（Claw 模式）

最近更新时间：2026-07-13 16:32:01

Claw 模式支持在运行时按用户或按会话动态调整 Agent 配置（更换模型、装卸 Skill、绑定插件工具），无需修改应用配置，也无需重新发布。本文说明该机制的启用前提、调用链路，以及各类配置变更的接口用法。

一、前提条件

前提 1：开启「允许在对话中动态修改配置」开关

在应用的**高级设置**中开启该开关（也可通过 ModifyApp 接口开启）。它是动态配置能力的总开关：仅当开关开启时，后续传入的用户级 `AgentId` 与运行时配置才会生效；否则对话仅使用应用发布时的固定配置。
通过接口开启开关：

```
req = models.ModifyAppRequest()
req.AppId = app_id
req.Config = models.AppConfig()
req.Config.Mode = models.AppModeConfig()
req.Config.Mode.ClawAgentConfig = models.ClawAgentConfig()
req.Config.Mode.ClawAgentConfig.CustomConfig =
models.ClawAgentCustomConfig()
req.Config.Mode.ClawAgentConfig.CustomConfig.Enabled = True
req.UpdateMask = models.FieldMask()
req.UpdateMask.Paths = ["Config.Mode.ClawAgentConfig"]
client.ModifyApp(req)
```

> ⚠ **注意：**
> 修改开关后需要调用 CreateRelease 重新发布才能生效，可参考 从零搭建一个 Claw 模式应用 > 发布应用。

前提 2：获取用户级 Agent

按用户维度修改配置，需要一个**归属于该用户的 Agent 实例**，而非直接修改应用的配置端 Agent。通过 CopyAgentFromApp 从应用复制出用户级 Agent：
Kind 枚举值（关于开发者/使用者的概念，详情请参见 概览）：

值	含义
0	配置端 Agent（**开发者**创建应用时使用）。

1	用户级 Agent（为使用者复制出的独立实例，用于按用户维度修改配置）。

```
req = models.CopyAgentFromAppRequest()
req.AppId = app_id
req.Kind = 1                    # 用户级 Agent
resp = client.CopyAgentFromApp(req)
agent_id = resp.ParentAgentId    # 该用户后续对话与改配置都用这个 AgentId
print(f"agent_id={agent_id}")
```

二、将用户级 Agent 绑定到会话

创建会话时将上一步的 `AgentId` 传入 CreateConversation（**仅在开关开启时可用**）：
Type 枚举值： `5` = API 接入。详情请参见 从零搭建一个 Claw 模式应用 > 步骤 6：创建会话 。

```
req = models.CreateConversationRequest()
req.Type = 5                    # API 接入
req.AppId = app_id
req.AppKey = APP_KEY
req.UserId = USER_ID
req.AgentId = agent_id        # 绑定用户级 Agent
resp = client.CreateConversation(req)
conversation_id = resp.ConversationId
```

绑定后，针对该 `AgentId` 的任何配置变更仅作用于该用户的会话链路。

三、动态修改配置

应用级 vs Agent 级：开启动态配置开关是**应用级设置**（通过 `ModifyApp`，所有 Agent 共享），而下文的模型/Skill/工具变更是 **Agent 级设置**（通过 `ModifyAgent`，每个用户级 Agent 独立）。关于两者的区别详细请参见 从零搭建一个 Claw 模式应用 > 步骤 4：修改配置（可选） 。
所有 Agent 级配置变更均通过 ModifyAgent 完成，由 `UpdateMask.Paths` 指定需更新的字段，其余字段保持不变。

3.1 更换模型

先通过 DescribeModelList 拉取 Claw 场景可用模型（`ModelScene=18` 为 Claw 模式可用模型，完整场景枚举见 API 文档），再调用 ModifyAgent 更新 `Model` 字段：

```
req = models.DescribeModelListRequest()
req.SpaceId = space_id
```

```
req.ModelScene = 18
req.PageNumber = 0
req.PageSize = 20
resp = client.DescribeModelList(req)
new_model_id = resp.ModelList[0].ModelBasic.ModelId

req = models.ModifyAgentRequest()
req.AppId = app_id
req.AgentId = agent_id
req.Agent = models.AgentSpec()
req.Agent.Model = models.AgentModelConfig()
req.Agent.Model.ModelId = new_model_id
req.Agent.Model.ContextWordsLimit = 20000
req.Agent.Model.InstructionsWordsLimit = 20000
req.UpdateMask = models.FiledMask()
req.UpdateMask.Paths = ["Model"]
client.ModifyAgent(req)
```

3.2 装卸 Skill

先通过 DescribeSkillSummaryList 拉取可选 Skill（可配合 DescribeSkillCategoryList 按分类浏览），再使用 ModifyAgent 提交更新后的完整 `SkillList`（全量覆盖，非增量）：

```


req = models.ModifyAgentRequest()
req.AppId = app_id
req.AgentId = agent_id
req.Agent = models.AgentSpec()

skill_list = []
for skill_id in skill_ids:
    skill = models.AgentSkill()
    skill.SkillId = skill_id
    skill_list.append(skill)

req.Agent.SkillList = skill_list
req.UpdateMask = models.FieldMask()
req.UpdateMask.Paths = ["SkilList"]
client.ModifyAgent(req)
```

- **装载 Skill**：将新的 SkillId 加入列表后整体提交。
- **卸载 Skill**：将目标 SkillId 从列表中移除后整体提交。
- **清空全部**：提交空数组 `[]`。

3.3 绑定 / 解绑插件工具

先通过 DescribePluginSummaryList 拉取可用插件（`Module=1` 表示应用已装载的插件，完整枚举见 API 文档），再使用 ModifyAgent 更新 `PluginList` 与 `ToolList`：

```
req = models.DescribePluginSummaryListRequest()
req.SpaceId = space_id
req.Module = 1
resp = client.DescribePluginSummaryList(req)

req = models.ModifyAgentRequest()
req AppId = app_id
req AgentId = agent_id
req.Agent = models.AgentSpec()

plugin = models.AgentPluginConfig ()
plugin.PluginId = "bf8ad2f3-1e5c-410a-ae81-abd7c5c14f1c"
plugin.AuthType = 0

req Agent.PluginList = [ plugin ]
req.UpdateMask = models FieldMask
req.UpdateMask.Paths = ["PluginList"]
client ModifyAgent (req)
```

四、透传自定义变量（无需修改 Agent）

若仅需向 Agent 传递运行时上下文（用户信息、部门等），**无需 `CopyAgentFromApp`，也无需修改 Agent**——只需在对话请求的 `Contents` 中追加一个 `custom_variables` 块即可（同样受开关约束）：

```
import json

contents = [
    {"Type": "text", "Text": "帮我生成本月周报"},
    {"Type": "custom_variables", "CustomVariables": {
        # value 若是对象/数组，必须先序列化成字符串
```

```
        "account": json.dumps({"id": "u_123", "name": "张三"},
ensure_ascii=False),
        "department": "数据科学部",
    }},
]
# 把 contents 作为 /adp/v2/chat 请求体的 Contents 字段发送（见「从零搭建一个
Claw 应用」的发起对话步骤）
```

`CustomVariables` 的 value 若为对象或数组，**必须先经 `json.dumps` 序列化为字符串**再传入，否则 Agent 无法正确接收。

五、机制小结

```
ModifyApp(Config.Mode.ClawAgentConfig) + CreateRelease  ← 通过 API 开启总
开关（或在网页高级设置中开启）
        ↓
CopyAgentFromApp(AppId, Kind=1) → ParentAgentId         ← 获取用户级 Agent
（修改配置时必需）
        ↓
CreateConversation(AgentId=ParentAgentId)               ← 绑定到会话
        ↓
ModifyAgent(AgentId, UpdateMask=[...])                  ← 更换模型 / 装卸
Skill / 绑定工具
```

动态操作	是否需要用户级 Agent	关键接口
开启动态配置开关	–	`ModifyApp(["Config.Mode.ClawAgentConfig"])` + `CreateRelease`
获取用户级 Agent	–	`CopyAgentFromApp(AppId, Kind=1)`
更换模型	需要	`DescribeModelList` + `ModifyAgent(["Model"])`
装卸 Skill	需要	`DescribeSkillSummaryList` + `ModifyAgent(["SkillList"])`
绑定 / 解绑工具	需要	`DescribePluginSummaryList` + `ModifyAgent(["ToolList","PluginList"])`

透传自定义变量	不需要	对话请求 `Contents` 中追加 `custom_variables`

搭建同款智能工作台

最近更新时间：2026-07-09 15:59:01

前两篇介绍了如何通过 API 从零搭建 Claw 模式应用。但直接调用 API 只完成了**后端的服务端逻辑**（签名、会话、对话、沙箱产出等），并不包含终端用户实际使用的**前端交互界面**。
为降低完整落地的成本，ADP 提供了开源项目 **adp-chat-client**，一套包含**前端界面**与**后端服务**的完整示例工程。
开发者既可以直接部署，快速获得一个与 ADP 同款的智能工作台，也可以参照其实现进行二次开发，搭建符合自身业务需要的对话产品。

一、adp-chat-client 的定位

维度	直接调用 API（前两篇）	使用 adp-chat-client
交付内容	仅服务端调用逻辑	前端界面 + 后端服务的完整工程
前端	需自行开发	提供现成对话界面，开箱即用
签名与转发	需自行实现 V3 签名	已封装，前端不接触密钥
部署	需自行搭建	支持 Docker 一键部署
适用场景	深度定制、非 Python 后端	快速落地、以现有工程为蓝本二次开发

adp-chat-client 可将 ADP 应用快速部署为 Web 应用，也可嵌入小程序、Android、iOS 等终端。

二、提供的能力

前端（Vue 3 + Vite）

一套完整的智能工作台交互界面，覆盖终端用户对话所需的主要能力：

- 实时流式对话、对话历史管理。
- 语音输入、图片理解。
- 交互式 Widget（图表、表单等）渲染（`adp-widget` 独立包）。
- Skills 管理（浏览技能广场、安装/卸载/排序）。
- 连接器 / 工具插件管理（浏览、OAuth 授权、绑定/解绑）。
- 任务列表、Skill / 连接器 / 知识库的选择与调用等工作台交互。

后端（Python / Sanic）

将服务端全部调用细节封装完毕，前端无需接触密钥与签名：

- 腾讯云 V3 签名与云 API 转发。

- 对话 SSE 流式转发。
- 实时文档解析（SSE 流式返回解析进度，获取 `doc_id`）。
- 沙箱工作空间的文件读写与下载。
- 第三方账户体系对接（OAuth / URL 跳转等）。
- 限流、CORS、Iframe 嵌入等生产部署所需的服务端能力。

三、快速部署

adp-chat-client 支持 Docker 一键部署，填入密钥与 ADP 应用信息（`ApplicationId` / `AppKey` 等）即可运行。完整的部署步骤、系统要求、账户体系对接、Nginx / CORS / 子路径等专题，参见仓库 README。

Claw 模式若需支持用户在对话中自定义选择 Skill / 模型 / 连接器，需在应用高级配置中开启「允许在对话中动态修改配置」并重新发布，详情请参见 动态修改 Agent 配置。

四、真实例子：动态修改模型

以"用户在对话中切换大模型"为例，展示 adp-chat-client 前后端如何配合实现 动态修改 Agent 配置。

前提：获取用户专属 AgentId

每个用户第一次操作时，需要通过 `CopyAgentFromApp` 获取专属的 AgentId（后续所有动态配置修改都针对这个 Agent）：
对应前端代码（`composables/useAgentStore.ts`）：

```
// 1. 先查后端缓存
const record = await getAgentConfig({ ApplicationId: appId });
if (record.AgentId) return record.AgentId;

// 2. 首次：调用 CopyAgentFromApp 获取专属 Agent
const result = await copyAgentFromApp({ ApplicationId: appId });
const agentId = result.ParentAgentId;

// 3. 持久化到后端 DB
await saveAgentConfig({ ApplicationId: appId, AgentId: agentId });
```

修改模型：前端发起

用户在界面选择新模型后，前端通过 `/adp/ModifyAgent` 接口提交修改：

```
// client/packages/adp-chat-component/src/composables/useAgentStore.ts
await modifyAgentByPath(applicationId, {
```

```
    model: {
        ModelId: 'new-model-id',          // 新模型 ID
        Alias: '智谱GLM-5.2',
        ModelParameters: { Temperature: 0.7, MaxTokens: 4096 }
    }
});

// 内部等价于调用:
POST /adp/ModifyAgent
{
    "ApplicationId": "app_xxx",
    "Payload": {
        "AppId": "app_xxx",
        "AgentId": "agent_xxx",
        "Agent": {
            "Model": { "ModelId": "new-model-id", "Alias": "智谱GLM-5.2",
... }
        },
        "UpdateMask": { "Paths": ["model"] }
    }
}
```

修改模型：后端转发

后端通过通用的 `/adp/<Action>` 转发端点，自动完成 V3 签名并转发给 ADP 云 API：

```
# server/router/adp.py — 通用转发路由
# 收到 /adp/ModifyAgent 后:
# 1. 从 Payload 中取出业务参数
# 2. 用 SecretId/SecretKey 做 V3 签名
# 3. 转发到 ADP 管理 API（adp.tencentcloudapi.com 或 capi.adp.tencent.com）
# 4. 原样返回响应
```

关键分工

层	职责	密钥接触
前端	用户交互、构造 UpdateMask 参数、缓存 AgentDetail	不接触
后端	V3 签名、转发 ADP 管理 API、持久化 AgentId 映射	SecretId/SecretKey

ADP	执行 ModifyAgent，更新 Agent 配置，下次对话生效	–

> **注意：**
> 修改模型后**无需重新发布**，下一次对话请求会自动使用新模型。这是 Claw 模式**运行时动态配置**的核心优势。

五、二次开发

工程按标准前后端结构组织，便于裁剪与扩展：

- **前端：**
  - `client/packages/adp-chat-component`：核心对话组件，Vue 3 + Vite，界面与调用封装分离，可替换主题、裁剪功能或嵌入既有站点。
  - `client/packages/adp-widget`：交互式 Widget 渲染包（图表、表单等），支持懒加载与自定义路径配置。
- **后端：** `server/`，按 `config` / `core` / `model` / `middleware` / `router` / `util` 分层，`core` 层与具体协议解耦，`model` 层提供 ORM 实体定义，便于复用与替换。

在此基础上模仿或改造，即可较低成本地搭建出符合自身业务的智能工作台。