基于微信客服 + OpenAI 的自动应答 Agent

mia m | Modified Today

1. 项目概述

本项目是一个基于微信客服 API + OpenAI API 搭建的 AI 客服 Agent。用户通过微信客服入口发送问题后，系统自动接收消息、完成回调解密、调用 AI 生成回复，并通过微信客服接口自动发送给用户。

项目覆盖了从服务器部署、微信回调配置、AI 接口接入、消息处理逻辑优化到线上问题排查的完整流程，旨在验证 AI 客服在私域场景的落地可行性，是一个从 0 到 1 的 AI 应用落地项目。

2. 项目成果

最终实现：

微信客服后台保存成功

URL / Token / AESKey 验证通过

服务器能收到客户消息

OpenAI 能生成回答

微信 send_msg 能发送回复

PM2 能后台守护运行

项目从一个"无法保存回调、无法回复、重复刷屏"的初始状态，逐步优化为一个可在线运行、可接收客户问题并调用 AI 回复的客服 Agent。

3. 项目背景

微信客服常用于私域客户咨询、产品介绍、功能答疑和售前沟通。但人工客服存在响应慢、重复问题多、测试成本高等问题。本项目希望通过 AI Agent 代替基础人工客服，完成常见问题的自动回复，

例如：

产品咨询

功能说明

价格方案

使用问题

售前答疑

目标是搭建一个可以真实接入企业微信客服场景的 AI 自动回复系统，而不是单纯的本地 Demo

4. 项目目标

Grid

	目标	说明
1	打通微信客服回调	用户发信息后，服务器可以收到微信回调
2	完成信息解密	支持Token/EncodingAESKey校验和消息解密
3	接入OpenAI API	将用户问题传给AI，并获取动态回复
4	自动发送客服消息	通过微信send_msg接口回复用户
5	支持线上部署	使用云服务器，Caddy，PM2保证服务可访问和持续运行
6	优化异常问题	解决重复回复，固定欢迎语，消息不回复，接口限流等问题

5. 技术栈

Node.js

JavaScript

微信客服 API

OpenAI Responses API

PM2

Caddy

Docker

Linux

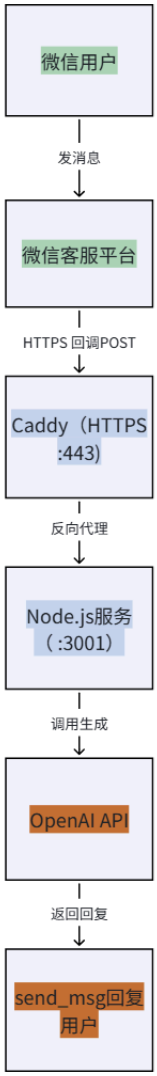
阿里云服务器

Webhook

HTTPS 反向代理

AES 消息解密

6. 系统架构



用户提问x's

→ 微信客服

→ 回调 URL /wecom/callback

→ Caddy HTTPS 反代

→ Node.js server.js

→ 微信消息解密

→ sync_msg 拉取消息

→ OpenAI API 生成回答

→ send_msg 回复用户

7. 核心链路

本项目共打通了5个核心链路：

Grid

	链路	说明
1	回调验证	微信后台URL，Token，EncodingAESKey校验通过
2	消息解密	AES解密微信客服回调消息
3	消息拉取	通过sync_msg获取客户文本消息
4	AI生成	调用OpenAI Responses API 生成自然语言回复
5	客服回复	通过send_msg将AI答案发送回微信客服会话

8. 项目目录与入口

> 项目目录：/opt/wecom-bot

> 入口文件：server.js

运行端口：3001

进程管理：pm2 wecom-bot

> 回调路径：/wecom/callback

> 外网回调：https://bot.wzmwzmwzmm.com/wecom/callback

9. 主要功能

9.1 微信客服回调接入

配置官网微信客服后台的回调URL:

> https://bot.wzmwzmwzmm.com/wecom/callback

服务器接微信的 GET / POST 请求，完成 Token 和 EncodingAESKey 校验。

9.2 客户消息自动接收

当用户在微信客服入口发送消息后，服务端通过微信客服 API 拉取消息内容。

支持识别：

客户 ID

客服账号 ID

消息类型

文本内容

消息 ID

发送时间

9.3 OpenAI动态回复

系统将用户问题传入 OpenAI API，由 AI 生成自然语言回答。

从原本只能回复固定欢迎语，优化为可以针对用户问题回答

例如：

用户：你们有什么功能?

AI：我们可以提供 AI 客服、产品咨询、自动回复、功能说明等服务。

9.4 微信客服自动发送

AI 生成结果后，通过微信客服 send_msg 接口发送回用户会话。

9.5 PM2后台运行

使用 PM2 管理 Node.js 服务，避免关闭终端后机器人停止运行。

常用命令：

```
pm2 restart wecom-bot --update-env
pm2 logs wecom-bot --lines 100
```

10. 项目难点与解决方案

Grid

	question	action	method
1	服务器部署	已打通	阿里云ECS+PM2守护，进程稳定online
2	HTTPS入口	已打通	Caddy自动申请Let's Encrypt证书，反代到本地服务
3	回调验证	已打通	URL/Token/EncodingAESKey校验通过
4	消息解密	已打通	AES解密回调密文，正确解析用户消息
5	AI生成	已打通	调用OpenAI ResponsesAPI，返回status:completed
6	消息去重	已打通	cursor持久化+msgid去重+并发竞态处理
7	消息稳定推送	已打通	send_msg返回errcode：0，用户微信实时收到AI回复

排障实录

问题 1：消息重复应答,触发发送限流

现象

用户发一条消息,AI 却重复回复多条;短时间内高频发送后报错 `errcode 95001 send msg count limit` ,回复发不出去。

排查思路

先判断这是不是两个独立问题。发现「重复应答」和「限流」其实是同一条因果链:重复应答 → 短时间发太多 → 触发限流。所以定位重点放在「为什么会重复应答」。

根因

两个叠加:

1. `sync_msg` 的 `cursor` 写死为空,每次回调都从头拉取,把保留期内的历史消息整段重拉一遍;
2. 每个回调事件异步触发一次处理,多个处理并发执行,各自读取旧的去重快照,在 `await OpenAI` 期间互相看不到对方的标记 → 同一条消息被多个流程同时处理(并发竞态)。

解法

- `next_cursor` 持久化到文件,每页拉取后保存,不再重拉历史;
- 去重改为单一进程内 Set,在 `await` 之前同步「检查+标记」,让并发流程立刻互相可见;
- 发送改为按客服账号串行 + 每条间隔,平滑发送速率避开限流。

体现能力:并发竞态识别、幂等设计、第三方接口限流规避。

问题 2：服务崩溃循环,PM2 反复重启

现象

`pm2 list` 显示进程 `stopped` ,重启次数累计 18 次,处于崩溃循环。

排查思路

不猜,直接看错误日志: `pm2 logs wecom-bot --err` 。

根因

Code block

```
SyntaxError: Identifier 'fs' has already been declared
```

粘贴代码时新旧代码混在一起, `fs` 被声明两次,启动即崩。

解法

先 `: > server.js` 彻底清空再粘贴,避免残留;用 `node -c server.js` 做语法预检,确认无误再 `pm2 restart` 。

体现能力:日志优先定位、语法级预检、编辑操作的严谨性。

问题3： 服务本机正常,但公网回调收不到

这是整个项目排障链条最长的一段,分四层逐步收敛。

现象 `curl http://127.0.0.1:3001/test` 返回正常,但微信回调到不了服务器。

排查思路(分层定位)

1. 本机层: `curl localhost:3001` → 正常,排除 bot 本身;
2. 入口层:检查 HTTPS 入口 Caddy → `systemctl status caddy` 显示 `failed` ,已挂 8 小时;
3. 端口层: `systemctl restart caddy` 仍失败,报 `443 address already in use` → `ss -tlnp | grep 443` 发现被 `docker-proxy` 占用 → `docker ps` 找到一个遗留的测试容器 `caddy-bot-test` 抢占了 80/443;
4. 证书层:停掉容器、释放端口后 Caddy 起来了,但 `curl https` 报 SSL 握手失败 → 查 Caddy 日志发现两个问题: `/etc/caddy/Caddyfile` (实际加载)里的域名比正确域名少一段字符(拼错),且在用 Let's Encrypt staging 测试环境签发假证书。

根因

遗留 Docker 容器抢占端口 + Caddy 配置域名拼写错误 + 未使用正式证书环境,三者叠加导致 HTTPS 入口不可用。

解法

- 停止遗留容器并 `docker update --restart=no` 禁止自启;
- 修正 `/etc/caddy/Caddyfile` 的域名,与实际域名、DNS 记录、微信回调 URL 三处对齐;
- 确认 DNS A 记录指向服务器公网 IP,触发正式环境签发,日志出现 `certificate obtained successfully` ;
- 最终 `curl https://域名/wecom/callback` 返回 200 + 业务响应,HTTPS 链路打通。

体现能力:分层排查方法论、端口冲突定位、反向代理与 HTTPS/证书链路、多处配置一致性核对。

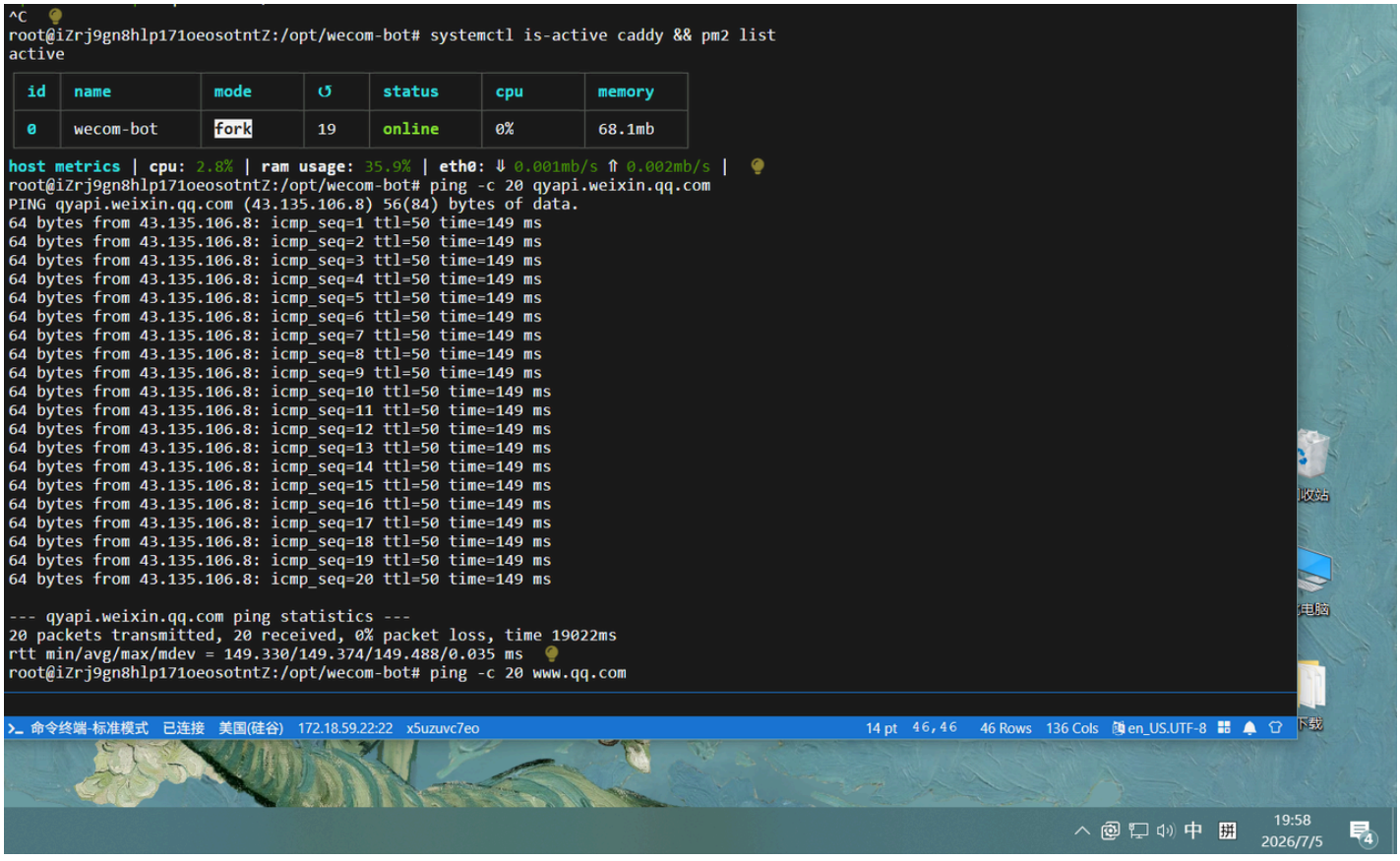
问题 4:URL 验证通过,但消息事件不推送

现象

微信后台保存回调时 GET 验证能通过(日志可见 `GET /wecom/callback`),但用户发消息时 POST 事件始终不到达。

排查思路

- 关键矛盾:同一个 URL,验证的 GET 能进、消息的 POST 不进 → 大概率不是网络/入口问题,而是微信侧未推送;



- 用 `ping` 测服务器到微信服务器的链路: `0% 丢包、延迟稳定 149ms` ,排除网络质量问题;
- 逐一排除:Caddy 曾中断 8 小时,判断为回调推送被临时熔断

根因

回调服务中断 8 小时,触发微信侧推送熔断,消息事件暂停推送。

解法

修复并保持服务持续稳定在线(`uptime` 稳定、 `unstable restarts: 0`),等待微信侧熔断机制自动探测恢复。恢复后测试:一次性拉取到 63 条积压消息,去重逻辑正确生效,仅新消息触发应答, `send_msg` 返回 `errcode: 0` ,用户实时收到回复。

体现能力:通过关键矛盾快速缩小范围、用量化手段排除假设、**判断问题性质(临时限制 vs 硬门槛)并采取"保持稳定 + 等待恢复"而非盲目重试的策略。**

11. 项目优化点

Grid

	优化项	优化前	优化后
1	回复逻辑	固定欢迎语	根据用户问题动态生成回复
2	消息处理	重复处理历史消息	加入文本过滤和消息去重
3	服务运行	依赖前台终端	使用PM2后台守护
4	回调访问	本地服务可用但公网不通	Caddy HTTPS反代访问
5	问题定位	靠猜测	通过日志，curl,grep,docker ps定位
6	异常处理	报错后无法判断原因	建立回调，AI，发送三个层级日志

12. 量化表达

1. 完成 1 套微信 AI 客服 Agent 从 0 到 1 部署
2. 打通 5 个核心链路：回调验证、消息解密、消息拉取、AI 生成、客服回复
3. 接入 3 类关键服务：微信客服 API、OpenAI API、Caddy HTTPS 反代
4. 解决 8 类线上部署与接口问题
5. 优化 3 类消息处理逻辑：文本过滤、msgid 去重、历史消息过滤
6. 建立 4 层排错流程：服务状态、回调日志、AI 返回、微信发送结果
7. 将机器人从固定欢迎语升级为动态 AI 问答
8. 完整打通全链路并实现真实用户场景验证:用户在微信端发消息,AI 秒级自动回复。

13. 项目复盘/收获

项目复盘

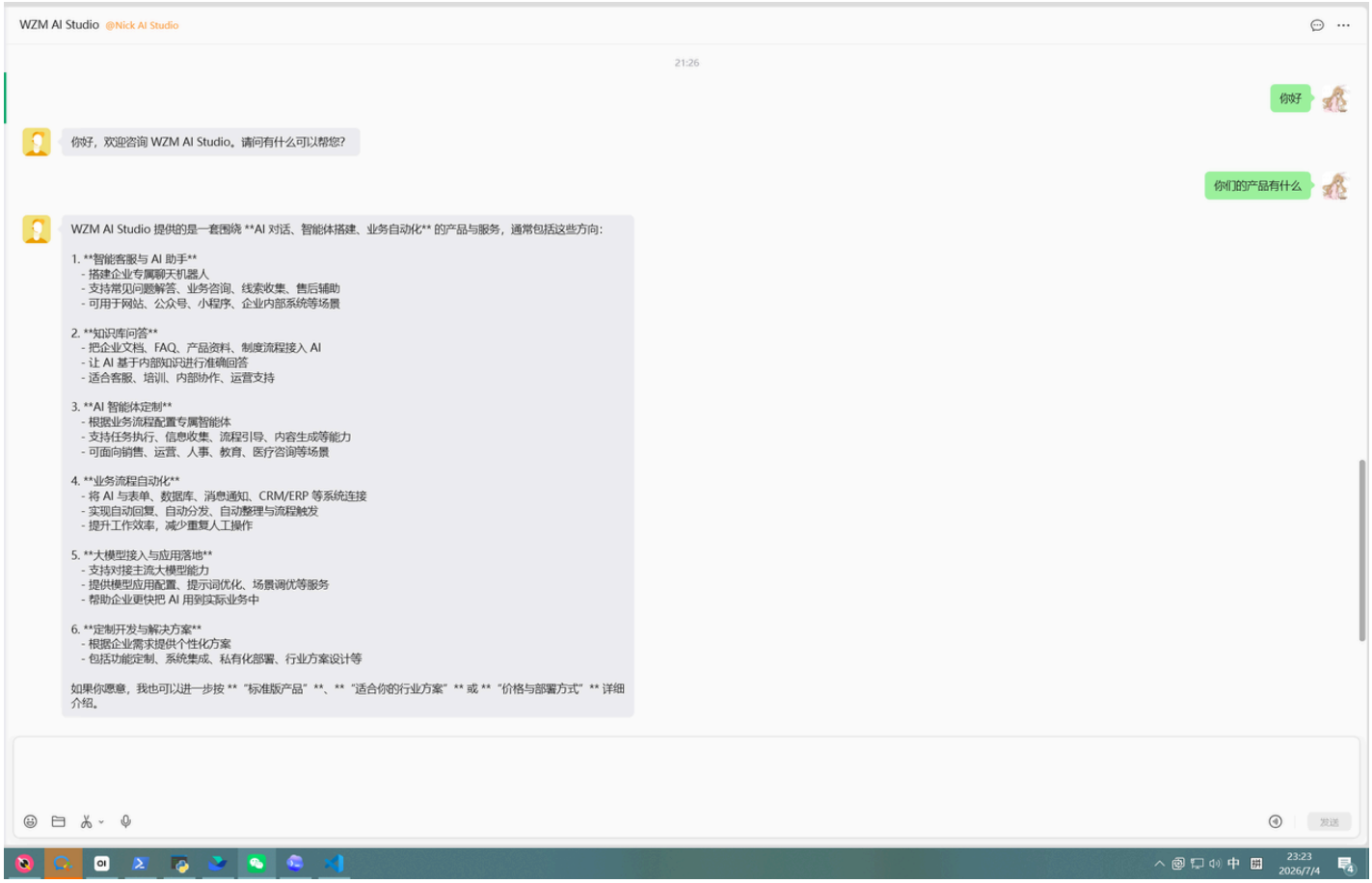
这个项目最大的价值不在于功能本身,而在于排障过程。整个链路跨了微信客服、Caddy、Node.js、OpenAI 多个环节,任何一环出问题都会导致整体不可用。我总结出一套定位方法:

1. **分层排查**:遇到"回调收不到"时,不盲目改代码,而是从本机 → 入口 → 端口 → 证书逐层验证,快速锁定是 Docker 容器抢占了端口;
2. **假设验证**:怀疑网络问题时,用 `ping` 测丢包率(0% 丢包)反而排除了自己的猜测,避免走错方向;
3. **判断问题性质**:面对"昨天能今天不能"的现象,判断是临时熔断而非账号资质硬限制,选择"保持稳定 + 等待"而不是盲目重试或急于做企业认证,最终验证判断正确。

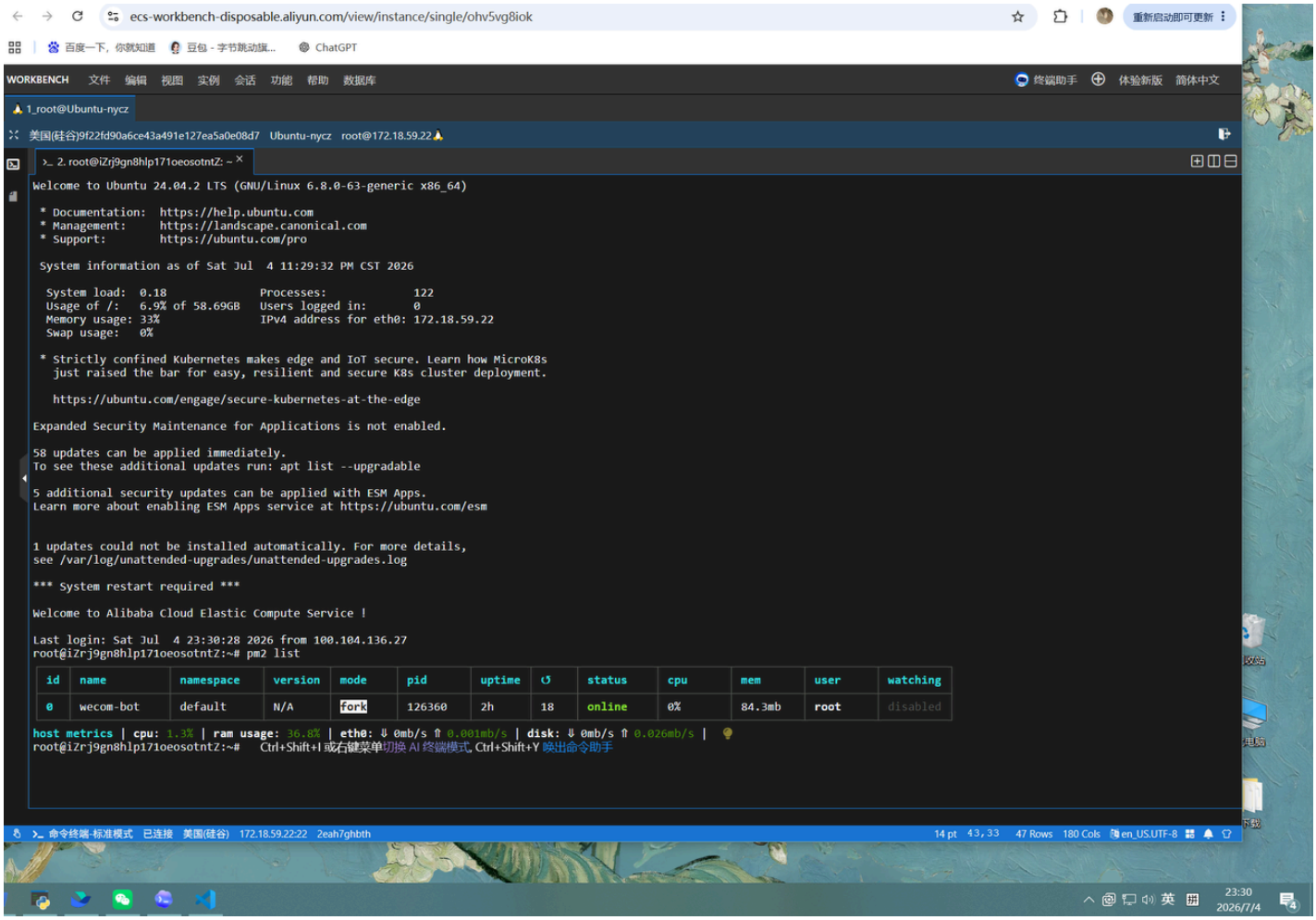
比起"会调 API",我更大的收获是学会了**如何系统性地排查一个跨多服务的复杂问题。**

14. 最终效果展示截图（部分）

- 微信客服移动端自动回复截图

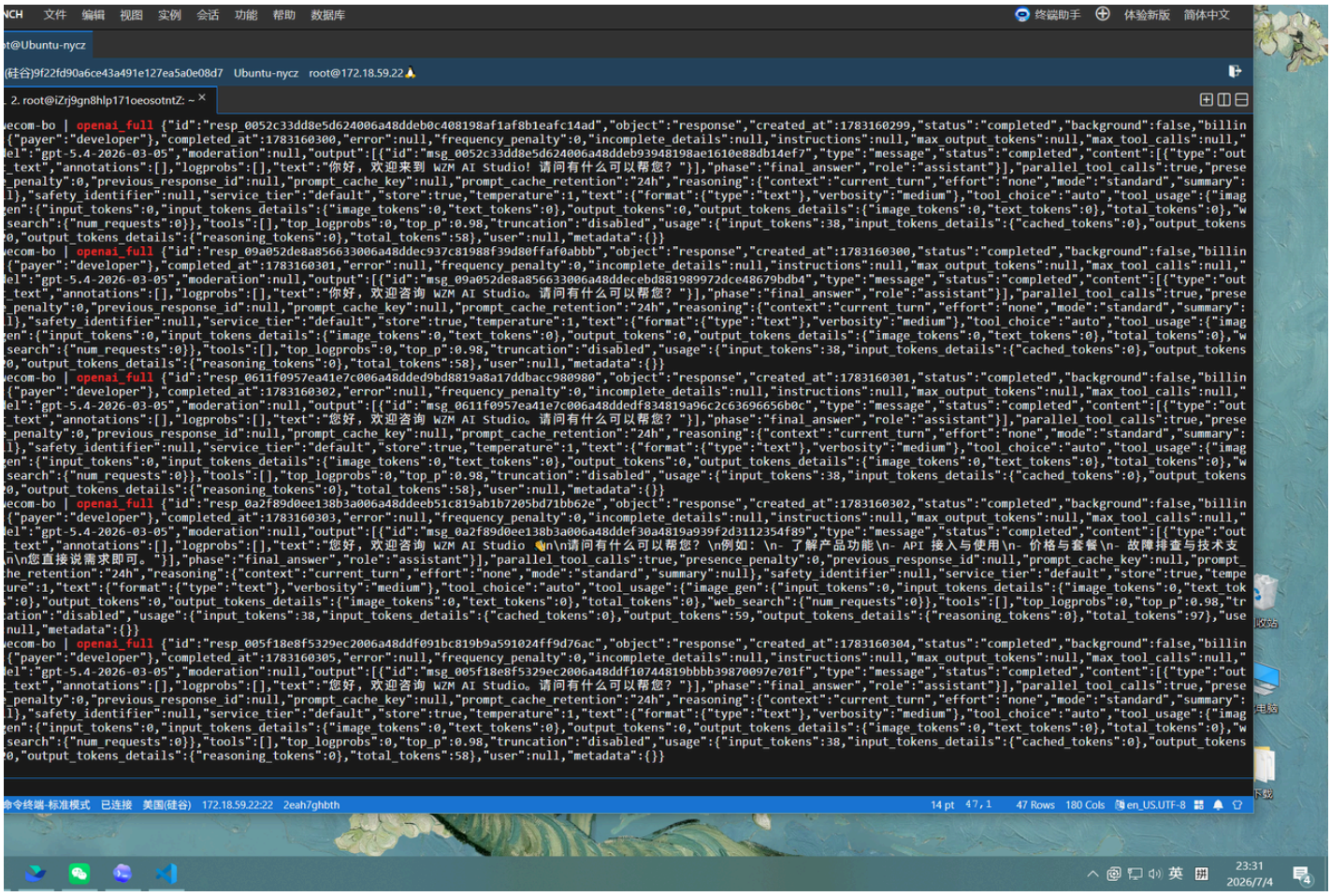


- pm2 list 显示 wecom-bot online



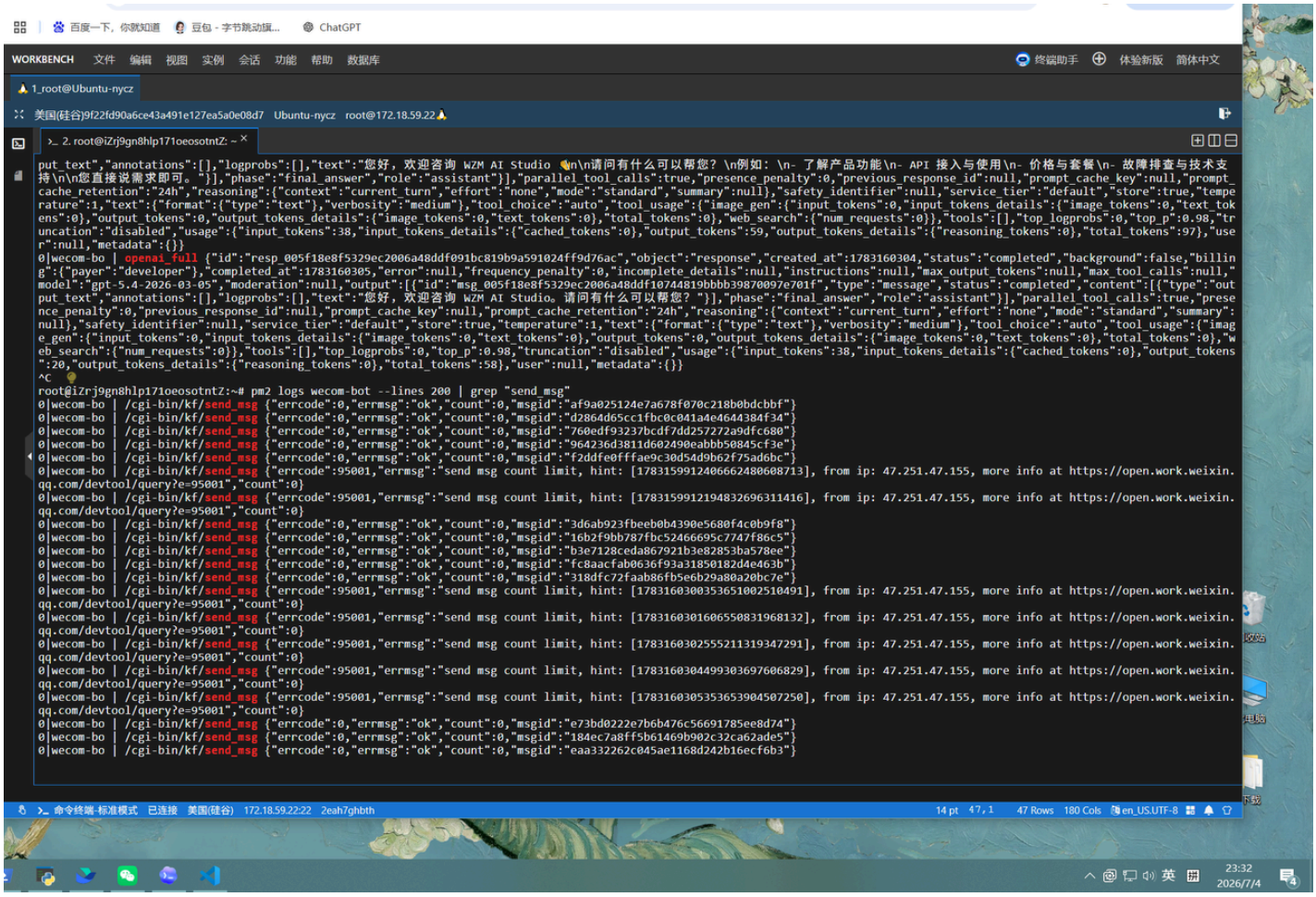
使用 PM2 管理 Node.js 服务，wecom-bot 进程保持 online 状态，支持后台持续运行。

- 日志里openai_full status completed



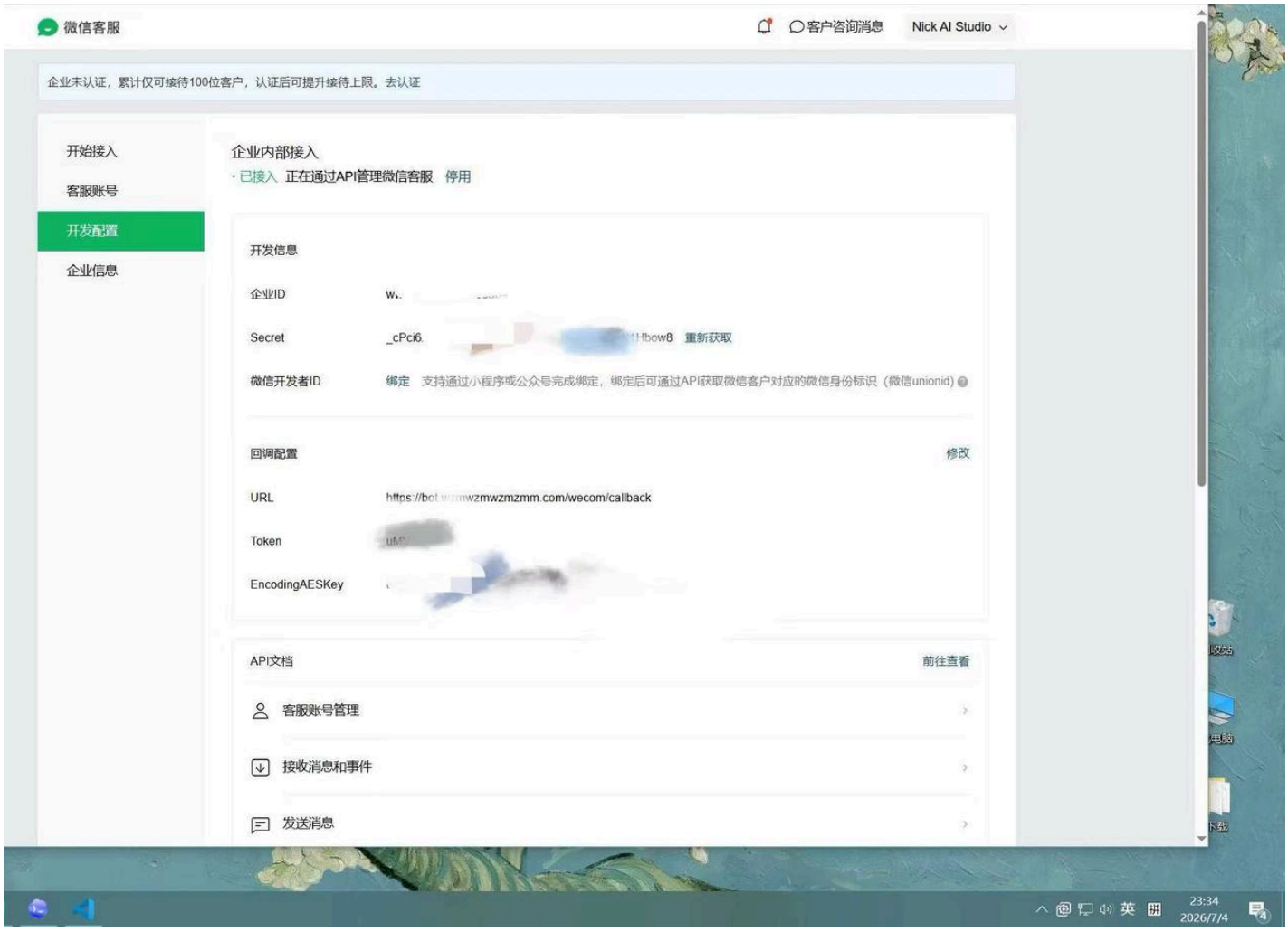
OpenAI Responses API 调用成功，模型返回 status: completed，并生成客服回复内容。

- send_msg errode 0

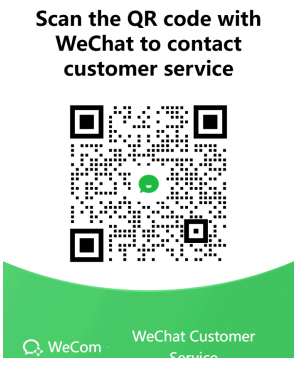


企业微信 send_msg 接口返回 errcode: 0，说明 AI 回复已成功发送到企业微信客服会话。

- 微信后台回调保存成功



微信客服后台回调配置保存成功，URL、Token、EncodingAESKey 校验通过。



扫码体验微信客服