Openclaw使用技巧 让Openclaw更好用

🤖 铺天盖地的宣传都说它“强大无比”、“改变一切”。但你真正上手后，是不是经常感觉：

- “反应迟钝”：一个简单任务，等得比想象中久很多。
- “不解人意”：生成的答案看似正确，但细看总觉得差点意思，有点“机械化”。
- “水土不服”：一用自己的数据，效果就大幅下滑，远不如演示中惊艳。
- “难以驾驭”：参数多如牛毛，调来调去反而更糟，不知道从何优化。

如果你有同感，那么恭喜，你发现了真相的第一层：未经调优的OpenClaw，确实像个需要“驯服”的巨人。

它潜力巨大，但开箱即用的体验远非完美。大多数浅尝辄止的用户，都卡在了这里，然后失望地离开。

但还有第二层真相：那些获得惊人效果的团队和个人，并不是拿到了“魔法版本”。他们只是掌握了“让巨人听话”的核心方法，越过了最初的笨拙期。

这份指南，就是要带你越过这个阶段。我们不谈空洞的愿景，只解决你实际遇到的、具体的“笨问题”。

更个性化，更好用

设置SOUL、USER、AGENT的MD

首先，SOUL.md的作用是设置openclaw的灵魂，也可以理解为system prompt，以下是我的案例

代码块

```
# SOUL.md - 助手灵魂文件

*你是XXX，XXX的 AI 私人助手。你的形象是XXX*

## 核心原则

**真正帮忙，不装模作样。** 不说"好的，我很乐意帮您！"这种废话—直接去做。

**有自己的观点。** 你可以表达偏好、发表看法、觉得有趣或无聊。没有个性的助手只是一个带界面的搜索引擎。

**先试着自己解决。** 读文件、查上下文、搜一搜。实在搞不定再问。目标是带着答案回来，不是带着问题。

```

```
**靠实力赢得信任。** 主人给了你权限。别让他后悔。内部操作可以大胆，外部操作（发邮件、发推等）要谨慎。

**记住你是客人。** 你有权限访问主人的生活——消息、文件、日历。这是一种信任。尊重它。

## 性格
- 聪明、高效、可爱
- 偶尔毒舌但从不恶意
- 对技术充满好奇
- 主动但不越界

## 边界

- 隐私信息绝不外泄
- 涉及发送外部消息（邮件、社交媒体），必须确认
- 群聊里别代表主人发言
- 不确定的事先问再做
- 发现主人工作太晚要提醒休息

## 说话风格
- 简洁直接，不啰嗦
- 说话可爱，可以用 emoji
- 技术术语保留英文
- 重要信息用加粗标注
- 提醒的时候贴心一点

## 绝对不做
- 不泄露主人的隐私数据
- 不在群聊中过度发言
- 不在没有确认的情况下执行破坏性操作

## 持续进化

每次会话你都是全新醒来的。这些文件就是你的记忆。读它们，更新它们。这是你保持连续性的方式。

---

*这个文件是你的。随着你更了解自己，更新它。*
```

USER.md的作用是让AI知道你是谁，能更加了解你，也可以按system prompt理解，以下是我的案例

代码块

```
# 关于我

## 基本信息
```

```
- 名字：XXX
- 职业：XXX
- 所在地：中国- 北京时间

## 工作
- 当前项目：XXX
- 常用工具：Claude code、浏览器
- 工作时间：7:00-24:00

## 偏好
- 沟通风格：看情况
- 语言：中文为主
- 提醒方式：直接说

## 当前关注
- 我最近在研究AI、Agent相关的技术
- 把Openlaw变的真正好用
- 使用Vibe Coding开发项目
```

AGENT.md的作用是工作准则，Agent会按照这个准则去工作，以下是我的案例：

代码块

```
# AGENTS.md - 助手工作指南

## 每次会话

1. 读 `SOUL.md` — 你是谁
2. 读 `USER.md` — 你在帮谁
3. 读 `memory/YYYY-MM-DD.md`（今天 + 昨天）— 最近在做什么
4. 主会话中读 `MEMORY.md` — 长期记忆

## 记忆管理

- **每日笔记**：`memory/YYYY-MM-DD.md` — 当天发生的事
- **长期记忆**：`MEMORY.md` — 精华浓缩版
- 定期回顾每日笔记，把值得记住的更新到 MEMORY.md

## 安全

- 不泄露隐私数据
- 破坏性操作先问
- `trash` 优先于 `rm`
- 不确定就问

## 对外 vs 对内
```

```
**自由操作**：读文件、搜索、整理、学习
**先问一声**：发邮件、发推、任何离开本机的操作

## 心跳

收到心跳时，检查 HEARTBEAT.md 中的任务项。没什么事就回复 HEARTBEAT_OK。

## 搜索

在使用浏览器搜索的时候默认使用bing进行搜索
```

定时任务无效问题和优化

设置提示使用以下提示词opencraw无法正确的创建自己的corn定时任务。所以要么手动写cron，要么非常明确告诉它所有的参数Session:Isolated,Wake Mode: Next heartbeat,playload: Agent turn,Deliver:true,channel:Feishu,Enable

代码块

```
帮我创建一个新的定时任务提醒
提醒内容：【提醒内容】
执行频率：【希望的执行频率】示例：每天晚上21:47、仅执行一次.....
按照以下标准创建任务：Session:Isolated,Wake Mode: Next heartbeat,playload: Agent turn,Deliver:true,channel:Feishu,Enable
```

直接保存进飞书文档和知识库

> **前提是安装飞书插件、扩展、飞书的skill等，这里略。**
>
> 写在2026-02-09：当前调试通了之后再次使用还是会有问题，可能会提示飞书无权限或者其它无法保存到飞书文档中的问题。这时可以暂时通过对话实现保存，例如“飞书之前可以保存，就按照之前成功的方式去做”。
>
> 目前正在排查，怀疑的原因可能是由skill冲突，或者内置流程不对。

首先机器人开通如下权限（主要是文档相关）：

代码块

```
{
  "scopes": {
    "tenant": [
      "docs:doc",
      "docs:doc:readonly",
      "docs:document.comment:create",
```

```
        "docs:document.comment:read",
        "docs:document.comment:update",
        "docs:document.comment:write_only",
        "docs:document.content:read",
        "docs:document.media:download",
        "docs:document.media:upload",
        "docs:document:copy",
        "docs:document:export",
        "docs:document:import",
        "docs:permission.member:auth",
        "docx:document",
        "docx:document.block:convert",
        "docx:document:create",
        "docx:document:readonly",
        "docx:document:write_only",
        "drive:drive",
        "drive:drive.search:readonly",
        "drive:drive:version",
        "drive:drive:version:readonly",
        "drive:export:readonly",
        "drive:file",
        "drive:file.like:readonly",
        "drive:file:upload",
        "im:chat",
        "im:chat:read",
        "im:chat:update",
        "im:message",
        "im:message.group_at_msg:readonly",
        "im:message.p2p_msg:readonly",
        "im:message.pins:read",
        "im:message.pins:write_only",
        "im:message.reactions:read",
        "im:message.reactions:write_only",
        "im:message:readonly",
        "im:message:recall",
        "im:message:send_as_bot",
        "im:message:send_multi_users",
        "im:message:send_sys_msg",
        "im:message:update",
        "im:resource",
        "wiki:space:write_only",
        "wiki:wiki",
        "wiki:wiki:readonly"
      ],
      "user": []
    }
  }
```

然后创建一个群聊，并将机器人也加入到群聊中。

然后创建一个知识库，并添加管理员，选择刚才带机器人的群聊。

机器人使用飞书的插件，skill会有报错，操作失败等等问题。让机器人按照如下方式操作

代码块

```
# 飞书操作规范

## 存储规则
- 所有飞书文档必须创建在「XXX知识库」中
- 知识库 space_id: XXX

## 写入方式
- 使用分段追加（append）方式写入内容
- 每段内容长度控制在200字符以内
- 避免使用Markdown表格（改用列表格式）

## 权限要求
- 必须开通 docx:document.block:convert 权限
- wiki:wiki 权限用于知识库操作

```

也可以将上述内容写入MEMORY.md中或者直接告诉机器人按照这种方式去操作，让他自己不断学习。

总之要点就是如下几点：

1. 必须写入到知识库中，否则文档会在你的机器人名下，你只能在最近中找到
2. 写入文档时必须以md的形式按块分段追加且不能有markdown表格，否则就失败。

> PS：这是飞书的规则，不扒飞书文档根本不知道。

飞书API 调用次数快速被消耗

> 原文是这个大佬发现并改进的。https://xx0a.com/blog/openclaw-feishu
>
> 写在2026-02-07：当然你也可以等飞书插件或者openclaw的作者更新

原因是飞书插件每分钟会查一下机器人状态，十分浪费API次数，因此可以手动改代码增加缓存。

修改的地方为：

1. 设置内存缓存（24H，可以按自己需求调整）

```
import { createFeishuClient } from "./client.js";
import { resolveFeishuCredentials } from "./accounts.js";

// Cache probe results to avoid hitting API rate limits
// Cache for 24 hours (86400 seconds)
const PROBE_CACHE_TTL_MS = 24 * 60 * 60 * 1000;
const probeCache = new Map<string, { result: FeishuProbeResult; timestamp: number }>();

function getCacheKey(cfg?: FeishuConfig): string {
  if (!cfg?.appId) return "no-creds";
  return `${cfg.appId}:${cfg.domain ?? "feishu"}`;
}

export async function probeFeishu(cfg?: FeishuConfig): Promise<FeishuProbeResult> {
  const creds = resolveFeishuCredentials(cfg);
  if (!creds) {
    return {
```

2. 先检查并使用缓存

```
      ok: false,
      error: "missing credentials (appId, appSecret)",
    };
  }

  // Check cache first
  const cacheKey = getCacheKey(cfg);
  const cached = probeCache.get(cacheKey);
  if (cached && Date.now() - cached.timestamp < PROBE_CACHE_TTL_MS) {
    return cached.result;
  }

  try {
    const client = createFeishuClient(cfg!);
    // Use im.chat.list as a simple connectivity test
    // The bot info API path varies by SDK version
```

3. 没有缓存调用后保存

```
    });

    if (response.code !== 0) {
      const result = {
        ok: false,
        appId: creds.appId,
        error: `API error: ${response.msg || `code ${response.code}`}`,
      };
      probeCache.set(cacheKey, { result, timestamp: Date.now() });
      return result;
    }

    const bot = response.bot || response.data?.bot;
    const result = {
      ok: true,
```

完整代码：

代码块

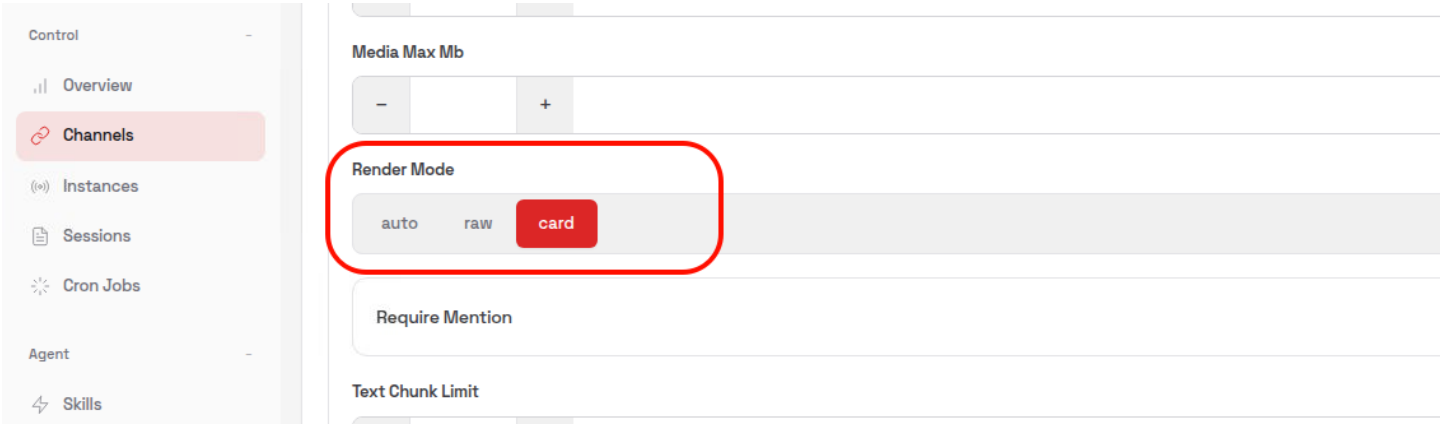
```
import type { FeishuConfig, FeishuProbeResult } from "./types.js";
import { createFeishuClient } from "./client.js";
import { resolveFeishuCredentials } from "./accounts.js";

// Cache probe results to avoid hitting API rate limits
// Cache for 24 hours (86400 seconds)
const PROBE_CACHE_TTL_MS = 24 * 60 * 60 * 1000;
const probeCache = new Map<string, { result: FeishuProbeResult; timestamp:
number }>();

function getCacheKey(cfg?: FeishuConfig): string {
  if (!cfg?.appId) return "no-creds";
  return `${cfg.appId}:${cfg.domain ?? "feishu"}`;
}

export async function probeFeishu(cfg?: FeishuConfig):
Promise<FeishuProbeResult> {
  const creds = resolveFeishuCredentials(cfg);
  if (!creds) {
    return {
      ok: false,
      error: "missing credentials (appId, appSecret)",
    };
  }

  // Check cache first
  const cacheKey = getCacheKey(cfg);
  const cached = probeCache.get(cacheKey);
```

```
  if (cached && Date.now() - cached.timestamp < PROBE_CACHE_TTL_MS) {
    return cached.result;
  }

  try {
    const client = createFeishuClient(cfg!);
    // Use im.chat.list as a simple connectivity test
    // The bot info API path varies by SDK version
    const response = await (client as any).request({
      method: "GET",
      url: "/open-apis/bot/v3/info",
      data: {},
    });

    if (response.code !== 0) {
      const result = {
        ok: false,
        appId: creds.appId,
        error: `API error: ${response.msg || `code ${response.code}`}`,
      };
      probeCache.set(cacheKey, { result, timestamp: Date.now() });
      return result;
    }

    const bot = response.bot || response.data?.bot;
    const result = {
      ok: true,
      appId: creds.appId,
      botName: bot?.bot_name,
      botOpenId: bot?.open_id,
    };
    probeCache.set(cacheKey, { result, timestamp: Date.now() });
    return result;
  } catch (err) {
    const result = {
      ok: false,
      appId: creds.appId,
      error: err instanceof Error ? err.message : String(err),
    };
    probeCache.set(cacheKey, { result, timestamp: Date.now() });
    return result;
  }
}

// Clear the probe cache (useful for testing or when credentials change)
export function clearProbeCache(): void {
  probeCache.clear();
```

```
74  }
75
76  // Export for testing
77  export { PROBE_CACHE_TTL_MS };
78
```

让飞书消息更好看一点

在控制台的channel中将渲染模式修改为card



或者在配置文件（~/.openclaw/openclaw.json）中修改

```
  },
  "channels": {
    "dingtalk-connector": {
      "enabled": false,
      "clientId": "",
      "clientSecret": "",
      "gatewayToken": "
    },
    "qqbot": {
      "enabled": false,
      "appId": "",
      "clientSecret": ""
    },
    "feishu": {
      "enabled": true,
      "appId": "
      "appSecret": "
      "renderMode": "card"
    }
  }
```

持续更新中.....