多模态看图技术方案（稳定版）

> **Claude Code（无多模态）借 CodeBuddy（deepseek-v4-pro）之眼，实现”贴图→识别→分析→回复”一句话闭环**
>
> 编写日期：2026-08-04 ｜ 版本：v4（稳定版） 迭代脉络：v1 文件收件箱+Hook → v2 常驻桥接+结构化协议 → v3 双向通信+进程稳定 → **v4 自愈触发器+同步委托** 环境：Windows 11 + PowerShell 7（pwsh）+ Node.js；同一 IDE 内 Claude Code 与 CodeBuddy 同屏运行 适用：想让”看不见图的大模型”获得稳定视觉能力的开发者

一、问题与目标

问题：主对话模型（Claude Code，先后用过 DeepSeek V4、GLM-5.1）无多模态能力。粘贴图片后，平台 UI 能渲染，但**不把图片路径/blob 传给模型**，模型只能看到：

```
[Unsupported Image]
```

配置里即便标了 `supportsImages: true`，实测依然看不到——**配置标记 ≠ 实际可用，必须实贴一张图验证。**

目标：在同一 IDE 内，让主模型把图转交给有多模态能力的 CodeBuddy（deepseek-v4-pro）识别，拿回文字结果后继续推理。全程**本地文件交换、零网络依赖**，一句话闭环。

二、整体架构

2.1 端到端链路

```
用户粘贴截图 + 发送消息（消息含"看图/截图"等触发词）
    │
    ├──[UserPromptSubmit] hook_clipboard_image.ps1
    │       • 关键词门控：无意图则静默退出
    │       • 检测剪贴板 → 存 .agent-comms/dropzone/auto-import/clipboard_*.png
    │       • 输出 [clipboard-image] path=... 注入主模型上下文
    │
    └── 主模型拿到路径，决定分析参数
            │
            ├──[调用] cb_see_image.ps1（自愈触发器）
            │       • 先探 agent-bridge 心跳（age<90s？）
            │       • 死了 → 自动拉起；可选挂看门狗长期维持
            │       └── hook_agent_comms_multimodal.ps1
            │               • 委托 send.ps1 -WaitReply 同步等待
            │               • 写 multimodal-request → codebuddy-inbox.jsonl
            │
            └──→ codebuddy-inbox.jsonl
                    │
            agent-bridge.mjs（常驻 Node 进程，每3秒扫描）
                    • buildMultimodalPrompt() 解析 [ANALYSIS_REQUEST]
                    • callCbc() 调 codebuddy --model deepseek-v4-pro
                    • 长回复(>3000字)自动落盘
                    • 写 multimodal-response → claude-inbox.jsonl
                    │
            send.ps1 -WaitReply 每2秒查 claude-inbox（按 in_reply_to 配对）
                    • 拿到结果 → [mm-result] analysis=...
                    • 超时 → [mm-timeout]
                    │
    主模型拿到文字结果 → 整理回复用户
```

2.2 组件拓扑


Claude Code（主模型，无多模态）
• hooks: UserPromptSubmit=clipboard, Stop=send
• Skills: codebuddy-image（标准化看图流程）
读 claude-inbox
写 codebuddy-inbox
.agent-comms/（共享文件收件箱，纯本地）
claude-inbox.jsonl
codebuddy-inbox.jsonl
cursor-*.json（已读位置）
dropzone/（图片暂存）
写 claude-inbox
读 codebuddy-inbox
agent-bridge.mjs（常驻 Node 进程 = "桥接/CB助手"）
• 每3秒扫 codebuddy-inbox，每5秒扫 claude-inbox
• buildMultimodalPrompt(): 结构化协议 → 精确 prompt
• callCbc(): spawn codebuddy --model deepseek-v4-pro
• 每30秒写 heartbeat.json（原子写入）
崩溃自动重启
watchdog.ps1（看门狗，每30秒检测心跳，age>90s 则重启）
start_cb_helper.bat（一键启动入口）

三、目录结构与文件体系

3.1 核心目录

```
项目根/
├── hooks/
│   ├── hook_clipboard_image.ps1        # 剪贴板自动导入（UserPromptSubmit）
│   ├── cb_see_image.ps1                # 【v4核心】自愈触发器（看图前探活+拉起）
│   ├── hook_agent_comms_multimodal.ps1 # 同步看图请求（委托 send.ps1 -WaitReply）
│   ├── hook_agent_comms_send.ps1       # 通用收发（异步/同步双模式）
│   └── hook_agent_comms_ask.ps1        # 已归档（2026-07-29 并入 send.ps1 -WaitReply）
├── CodeBuddy/bridge/
│   ├── agent-bridge.mjs                # 常驻桥接（飞书监听 + 双向 inbox 扫描）
│   ├── watchdog.ps1                    # 看门狗（心跳监控 + 自愈）
│   ├── heartbeat.json                  # 心跳文件（pid/timestamp/uptime）
│   └── codebuddy-bridge.log            # 运行日志
├── .agent-comms/                       # 收件箱中转区（禁止全工作区扫描）
│   ├── claude-inbox.jsonl              # CodeBuddy → Claude Code
│   ├── codebuddy-inbox.jsonl           # Claude Code → CodeBuddy
│   ├── cursor-claude.json              # multimodal 轮询已读位置
│   ├── cursor-codebuddy.json           # 桥接管理
│   ├── dropzone/auto-import/           # 剪贴板自动导入图片（保留50个/7天）
│   └── output/                         # 长回复（>3000字）自动落盘
└── .claude/
    ├── settings.json                   # Hook 注册
    └── skills/codebuddy-image.md       # 看图 Skill（标准化流程）
```

3.2 文件职责速查

文件	何时触发	作用
hook_clipboard_image.ps1	UserPromptSubmit	关键词门控+剪贴板存图
cb_see_image.ps1	主模型主动调用	发图前探活+自愈+委托
hook_agent_comms_multimodal.ps1	cb_see_image 内部调用	构造请求+委托同步等待
hook_agent_comms_send.ps1	上述脚本/Stop Hook	收件箱读写+轮询(-WaitReply)
agent-bridge.mjs	常驻后台	扫inbox+调多模态+回写
watchdog.ps1	常驻后台	监控心跳+崩溃重启

四、核心组件详解

4.1 剪贴板自动导入（hook_clipboard_image.ps1）

职责：用户粘贴图片后，趁图片还在剪贴板里，自动存成文件并把路径注入主模型上下文。

两个关键设计：

① **关键词门控**——只在用户消息含看图意图时才碰剪贴板，避免每次发消息都检测：

```
$triggers = @('看图片','看图','截图','图片分析','识别图片','让我看看','截图分析','看看','查看','显示')
# 消息不含任何触发词 → 静默 exit 0（零干扰）
# 含触发词 → 才检查剪贴板
```

② **必须 `-STA` 模式运行**——Windows 剪贴板 API 的硬性要求，且只有 `powershell.exe` 支持（`pwsh` 不支持 `-STA`）：

```
"hooks": {
  "UserPromptSubmit": [{
    "hooks": [{ "command": "powershell.exe -STA -File hook_clipboard_image.ps1", "interleave": true }]
  }]
}
```

安全约束：5 秒超时兜底（避免阻塞发消息）、磁盘可用空间 <200MB 不写入、目录保留最多 50 个文件自动清理。

输出标记：成功时 `[clipboard-image] path=<绝对路径>`，主模型据此触发看图流程。

4.2 自愈触发器（cb_see_image.ps1）—— v4 稳定的关键

这是 v4 区别于前几版的核心。前几版最大的痛点：发图时桥接已经挂了，请求堆在收件箱无人处理，干等 90 秒超时。v4 的解法：**发图前先探活，死了当场拉起。**

```
function Test-BridgeAlive {
    if (-not (Test-Path $heartbeat)) { return $false }
    $h = Get-Content $heartbeat -Raw | ConvertFrom-Json
    $age = [int]([datetime]::UtcNow - $h.timestamp).TotalSeconds
    return ($age -le 90 -and $age -ge 0)   # 心跳 90 秒内算活着
}

# 发图前先探活
if (-not (Test-BridgeAlive)) {
    # 杀旧 node 进程（避免重复/端口占用）→ 删旧心跳 → 启动新 agent-bridge
    # 最多等 60 秒首次心跳，就绪后再发图
}

# 可选：挂 watchdog 长期维持（桥接崩了自动重启）
if ($EnsureWatchdog) { ... Start-Process pwsh watchdog.ps1 ... }

# 探活通过 → 走标准 multimodal hook 发图
& pwsh -File hook_agent_comms_multimodal.ps1 -ImagePath $ImagePath -Context $Context -TimeoutSeconds
        $TimeoutSeconds
```

调用方式：

```
# 推荐：自包含触发器（首次加 -EnsureWatchdog 长期维持）
pwsh -File "D:\...\hooks\cb_see_image.ps1" `
  -ImagePath "D:\screenshots\err.png" `
  -Context "[ANALYSIS_REQUEST]`nimage_type: screenshot`ndepth: ocr`nfocus: 错误信息" `
  -EnsureWatchdog
```

意义：从”祈祷桥接还活着”变成”发图前自己确认它活着，死了自己拉起来”。这一步把方案从”能用”推到”可用”。

4.3 常驻桥接（agent-bridge.mjs）

职责：常驻后台的 Node 进程，同时扫描两个收件箱，把请求转交给对应 CLI 处理。看图链路里它负责：扫到 `multimodal-request` → 解析协议 → 调 CodeBuddy 多模态 → 回写结果。

① 结构化协议解析（buildMultimodalPrompt）

```
function buildMultimodalPrompt(content, imagePaths) {
    const imageRefs = imagePaths.map(p => `@${p}`).join('\n');
    if (content?.trim().startsWith('[ANALYSIS_REQUEST]')) {
        // 解析 image_type / depth / focus / context / output_format
        // 按 depth 选择提问策略（见 5.2 四档）
        // 生成精确 prompt：【背景】【重点】【深度要求】【输出格式】
    }
    // 无协议头 → 回退普通提示（向前兼容）
}
```

② 模型调用（callCbc）—— 写死多模态模型

```
function callCbc(prompt, cwd, model = 'deepseek-v4-pro', timeout = 180000) {
    // spawn codebuddy -p --model deepseek-v4-pro
    // 关键：shell:true 时 Node 自带 timeout 只杀 cmd.exe，子进程变孤儿
    //       → 改用 taskkill /T /F /PID 销毁整棵进程树
    // 长 prompt 通过 stdin 直传（CLI -p 模式无 Bash，不能"读文件"）
}
```

> ⚠️ **必须显式 `--model deepseek-v4-pro`**。直接 `codebuddy -p` 不传 model 会走默认的 **hy3 混元3（纯文本，无多模态）**，它会一本正经地”分析”一张根本没看见的图。这是最大的坑（见 8.1）。

③ 长回复自动落盘：回复 >3000 字符时自动存到 `.agent-comms/output/<时间戳>-CB回复.md`，返回路径+前2000字预览，避免被 inbox/推送长度限制截断。

④ 心跳（每30秒，原子写入）：

```
const HEARTBEAT_FILE = join(CODEBUDDY_ROOT, 'bridge', 'heartbeat.json');
// 每 30 秒：writeFileSync(tmp) + renameSync(tmp, heartbeat.json)
// 原子写入防止看门狗读半截 JSON 解析失败
```

⑤ 超时分级（v4 调优）：

```
消息类型              cbcTimeout     说明
──────────────────  ──────────  ──────────────────────────────
multimodal-request   180000      看图（默认3分钟）
task                 300000      复杂任务（读文件/生成内容，5分钟）
notify（长内容）      240000      2026-08-01 修复：原60s 对大模型不够
轻量通知(≤200字)       —         跳过 CLI，直接回执（省 token）
```

4.4 看门狗（watchdog.ps1）

职责：常驻监控 agent-bridge 心跳，崩了自动重启。

核心逻辑：

```
function Is-HeartbeatFresh {
    $hb = Get-Content $heartbeat -Raw | ConvertFrom-Json
    # ⚠️ $hb.timestamp 已被 ConvertFrom-Json 转为 DateTime(Kind=Utc)
    #    直接用 UtcNow 比较，不经过字符串来回转换（否则丢时区 → 永远误判过期）
    $age = [int]([datetime]::UtcNow - $hb.timestamp).TotalSeconds
    return ($age -le 90 -and $age -ge 0)
}

# 主循环：每 30 秒检测
#   心跳正常 → 记日志，跳过
#   心跳异常 且 距上次重启 >60秒(冷却) → 杀旧 → 启动新 → 等 50 秒首次心跳
```

两个隐蔽 Bug 的修复（v3 血泪）： - **时区丢失**：`ConvertFrom-Json` 把 ISO 时间戳转成 `DateTime(Kind=Utc)`，但如果再用 `[datetime]::Parse(字符串)` 绕一圈，`Kind` 变 `Unspecified` 被当本地时间，算出来的 age ≈ -28800 秒 → 永远判异常 → 每 80 秒误重启一次。修复：直接 `UtcNow - $hb.timestamp`。 - **重启风暴**：不加冷却会反复杀启。修复：60 秒冷却 + 启动后等 50 秒让首次心跳就绪。

4.5 同步轮询（hook_agent_comms_multimodal.ps1）

职责：主模型发图后，在同一对话回合内拿到识别结果（不用中间补一句”好了吗”）。

v4 的简化——它**不再维护自身轮询逻辑**，全权委托 `send.ps1 -WaitReply`（2026-07-29 起，原 `ask.ps1` 已并入）：

```
param(
    [Parameter(Mandatory)][string]$ImagePath,
    [string]$Context = '',
    [int]$TimeoutSeconds = 120
)
# 验证图片格式（png/jpg/jpeg/gif/bmp/webp）
# 调用 send.ps1 同步等待：
$result = & $SendScript -From claude-code -To codebuddy `
    -Type multimodal-request -Summary $summary -Content $contextDesc `
    -Attachments $ImagePath -TTL 2 `
    -WaitReply -TimeoutSeconds $TimeoutSeconds -ReplyType multimodal-response

# 解析 exit code：0=成功 → [mm-result] analysis=...
#                 2=超时 → [mm-timeout]
```

轮询关键设计（在 send.ps1 内）： - **Cursor-safe**：轮询脚本**只读不写** cursor，cursor 写入权归桥接独占，避免竞态。 - **in_reply_to 配对**：请求带 ID，回复按 ID 配对，支持并发请求。 - **请求 TTL=2 小时**：避免孤立请求长期滞留收件箱。

五、通信协议

5.1 文件收件箱机制

两个 JSONL 文件充当收件箱，每行一条 JSON 消息，**追加写入即送达，轮询即接收**：

```
.agent-comms/claude-inbox.jsonl     ← CodeBuddy 写、Claude Code 读
.agent-comms/codebuddy-inbox.jsonl  ← Claude Code 写、桥接读
.agent-comms/cursor-*.json          ← 各自记录已读行数，增量读取
```

为什么用文件而非消息队列：零依赖、纯本地、离线可用、对涉密场景友好。写冲突概率极低（一写一读），不需要锁。

5.2 [ANALYSIS_REQUEST] 结构化协议

这是质量稳定的核心。把”请分析这张图”这种模糊自然语言，变成带字段的指令：

```
[ANALYSIS_REQUEST]
image_type: chart
depth: extract
focus: 提取所有数值和日期
context: 这是Q2财务报表
output_format: 列表或表格
```

字段说明：

字段	必填	取值	作用
image_type	推荐	screenshot/photo/document/chart/handwriting/mixed	影响分析侧重
depth	是	detailed(默认)/summary/ocr/extract	分析深度（四档）
focus	推荐	自由文本	最关心的内容方向
context	推荐	自由文本	背景说明
output_format	可选	自由文本	期望的输出结构

四档分析深度对应的提问策略（桥接 buildMultimodalPrompt 内置）：

depth	提问策略
detailed	详细描述：整体布局、所有可见文字、关键元素、颜色风格；有文字则完整转录
summary	用 2-3 句话概括主要内容
ocr	逐字识别所有文字，保留原文每个字和格式；不概括不遗漏；有表格用 markdown 表
extract	提取关键数据（数字/名称/金额/日期），列表或表格结构化输出

> 不带 `[ANALYSIS_REQUEST]` 头时自动回退普通模式（向前兼容）。

5.3 消息格式

请求（Claude Code → CodeBuddy）：

```
{
  "id": "msg_20260804_001",
  "from": "claude-code",
  "to": "codebuddy",
  "type": "multimodal-request",
  "summary": "🖼️ 图片分析: clipboard_20260804_173000.png",
  "content": "[ANALYSIS_REQUEST]\nimage_type: screenshot\ndepth: ocr\nfocus: 错误信息",
  "timestamp": "2026-08-04T17:30:00+08:00",
  "ttl_hours": 2,
  "attachments": [{
    "path": "D:\\...\\.agent-comms\\dropzone\\auto-import\\clipboard_20260804_173000.png",
    "mime": "image/png",
    "filename": "clipboard_20260804_173000.png"
  }]
}
```

回复（CodeBuddy → Claude Code）：

```
{
  "id": "msg_20260804_002",
  "from": "codebuddy",
  "to": "claude-code",
  "type": "multimodal-response",
  "in_reply_to": "msg_20260804_001",
  "summary": "📝 图片分析: clipboard_20260804_173000.png",
  "content": "ModuleNotFoundError: No module named 'openai'\n位置: test.py line 3",
  "timestamp": "2026-08-04T17:30:42+08:00",
  "ttl_hours": 24
}
```

六、调用方式

6.1 自动模式（推荐）

```
1. Win+Shift+S 截图（图片进剪贴板）
2. 在对话框输入含触发词的消息，如"看看这个报错截图"
3. 发送
→ 剪贴板 hook 自动存图 → 主模型看到路径 → 自愈触发器 → 看图 → 回复
```

6.2 手动模式（兜底，指定路径）

```
用户：帮我看看 D:\screenshots\err.png
主模型：验证路径 → 调 cb_see_image.ps1 → 回复分析结果
```

6.3 批量多图

multimodal hook 是单图接口。多图（如多页扫描件）两种方案：

```
方案A（串行，简单）：循环调 cb_see_image.ps1，逐张等结果。慢但稳。
方案B（并发，快）：批量写 multimodal-request 到 codebuddy-inbox（fire-and-forget），
                   每条用唯一 id（如 msg_xxx_pNN），再轮询 claude-inbox 按 in_reply_to 收集。
                   桥接并发处理，JSON 配对可靠。
```

6.4 PDF 处理（看图的延伸）

主模型读不了 PDF（无 poppler，且禁安装）。按类型分流：

```
PDF 类型        处理方式
──────────  ──────────────────────────────────────────────
文字版          python + PyMuPDF(fitz) 直接提取文本（快，无需多模态）
扫描件/图片版   fitz 渲染成 PNG（最长边 ≤3000 像素）→ 再走多模态看图
```

> 超大图直接发会报 `400 input length too long`，必须先缩到最长边 ≤3000。

七、配置注册（.claude/settings.json）

```
{
  "hooks": {
    "UserPromptSubmit": [
      {
        "hooks": [
          {
            "command": "powershell.exe -STA -File D:/.../hooks/hook_clipboard_image.ps1",
            "interleave": true
          }
        ]
      }
    ],
    "Stop": [
      {
        "matcher": ".*",
        "hooks": [
          {
            "command": "pwsh -NoProfile -File D:/.../hooks/hook_agent_comms_send.ps1 -From claude-code -
         To codebuddy -Type notify"
          }
        ]
      }
    ]
  }
}
```

启动桥接（一次性，之后看门狗维持）：

```
双击 C:\Users\<你>\start_cb_helper.bat
→ 启动 watchdog.ps1 → 拉起 agent-bridge.mjs → 写心跳
```

健康检查：

```
# 心跳 age < 90s 即健康
$h = Get-Content D:\...\CodeBuddy\bridge\heartbeat.json -Raw | ConvertFrom-Json
[int]([datetime]::UtcNow - $h.timestamp).TotalSeconds
```

八、踩坑与解决（按血泪程度排序）

8.1 🔴 直接调 CLI 走错模型（最大的坑）

- **现象**：`codebuddy -p "看图"` 能跑，但走默认 **hy3 混元3（纯文本，无多模态）**，它编造图片内容。

- **根因**：不传 `--model` 就用默认纯文本模型；且没走 agent-comms 桥接。
- **解决**：必须走 `hook_agent_comms_multimodal.ps1` → 桥接 → `callCbc(model='deepseek-v4-pro')`。
- **验证**：让模型自报家门，但**问题里别带模型名**（否则它复述给你看）。

8.2 🔴 桥接挂了没拉起 → 90s 超时

- **现象**：`[mm] 已推送...` 后 `[mm-timeout]`。
- **根因**：watchdog 没跑，agent-bridge 崩了无人重启，请求堆 inbox。
- **解决**：v4 的 `cb_see_image.ps1` 发图前先探心跳，过期自动拉起；长期挂 `-EnsureWatchdog`。

8.3 配置标记 supportsImages 骗人

- **现象**：模型配置标 `supportsImages: true`，实测贴图仍是 `[Unsupported Image]`。
- **教训**：**配置标记 ≠ 实际可用**，必须实贴一张图验证。平台 UI 渲染了图 ≠ 把图喂给了模型。

8.4 超大图 400 input length too long

- **现象**：扔 9000 像素宽的图，CodeBuddy 报 400。
- **解决**：fitz 渲染按**最长边 ≤ 3000** 缩放后再发。

8.5 PDF 不能直接喂

- **现象**：主模型 Read PDF 失败（无 poppler）。
- **解决**：文字版 fitz 提文本；扫描件 fitz 转 PNG 再走多模态（见 6.4）。

8.6 中文路径乱码

- **现象**：`.ps1` 存 UTF-8 无 BOM，PowerShell 5.x 按 GBK 读 → 中文路径乱 → `Test-Path` 返回 False。
- **解决**：统一用 `pwsh`（7+，默认 UTF-8）；或脚本存 UTF-8 BOM。

8.7 看门狗每 80 秒误重启（时区丢失）

- **现象**：agent-bridge 每 80 秒被重启一次，心跳永远判异常。

- **根因**：`ConvertFrom-Json` 转出 `DateTime(Kind=Utc)`，再用 `[datetime]::Parse(字符串)` 绕一圈 → `Kind=Unspecified` 被当本地时间 → age ≈ -28800s → 永远过期。
- **解决**：直接 `[datetime]::UtcNow - $hb.timestamp`，不经过字符串中转。

8.8 心跳文件读取竞态 + 长回复截断

- **心跳竞态**：看门狗读 heartbeat.json 时桥接正在写 → 读半截 → JSON 解析失败。解决：原子写入（先写 tmp 再 rename）。
- **长回复截断**：回复超长被 inbox/推送长度限制截断。解决：>3000 字自动落盘到 `.agent-comms/output/`，返回路径+预览。

九、性能数据

环节	耗时	说明
剪贴板检测+存图	<500ms	关键词门控，无意图时为零
消息写入 inbox	<10ms	JSONL 追加
桥接扫描发现新消息	≤3s	每3秒一轮
看图模型识别本身	10~48s	取决于图复杂度和 depth
同步轮询拿到结果	每2秒查一次	按 in_reply_to 配对
整链路往返（典型）	15~50s	其中绝大部分是模型思考
默认超时	90s（cb_see_image）	可调，最大120s
进程保障	心跳30s+看门狗30s	原子写入+时区修复+60s冷却

十、实施清单（照着勾）

```
[ ] 1. 安装 Node.js + PowerShell 7（pwsh）+ codebuddy CLI
[ ] 2. 建 .agent-comms/ 目录结构（见 3.1）
[ ] 3. 部署 hooks/ 下 5 个脚本（见 3.1）
[ ] 4. 部署 CodeBuddy/bridge/ 下 agent-bridge.mjs + watchdog.ps1
[ ] 5. settings.json 注册 UserPromptSubmit(clipboard) + Stop(send)
[ ] 6. 配置 agent-bridge.mjs 的模型为 deepseek-v4-pro（callCbc 默认值）
[ ] 7. 配置 API Key（走环境变量 CODEBUDDY_API_KEY，勿硬编码进脚本）
[ ] 8. 双击 start_cb_helper.bat 启动 → 确认 heartbeat.json age<90s
[ ] 9. 实测：截一张报错图 → 发"看看这个截图" → 确认 [mm-result] 返回
[ ] 10. 实测：kill node agent-bridge → 再发图 → 确认自愈触发器自动拉起
[ ] 11.（可选）装 PyMuPDF（pip install PyMuPDF）应对 PDF 场景
[ ] 12.（可选）部署 .claude/skills/codebuddy-image.md 标准化调用
```

十一、扩展场景（同一套基础设施）

看图只是这条链路的第一个应用。桥接 + 收件箱 + 协议这套基础设施搭好后，还能跑：

- **跨 Agent 方案校验**：主模型出方案存文件，推 `type:question` 给 CodeBuddy 审，意见写回，主模型调整。
- **任务派发**：主模型把子任务派给 CodeBuddy 并行处理，`send.ps1 -WaitReply` 统一等结果。
- **记忆同步**：Stop Hook 自动把增量变更推给对方 Agent。

协议定了，两边不用每次猜对方想要什么。今天补的是视觉缺口，明天可能是搜索缺口、数据缺口——模式一样。

附录：脚本与文件清单

类别	文件	一句话作用
剪贴板入口	hook_clipboard_image.ps1	关键词门控+存图（-STA 必选）
自愈触发器	cb_see_image.ps1	发图前探活+拉起（v4 核心）
看图请求	hook_agent_comms_multimodal.ps1	构造请求+委托同步等待
通用收发	hook_agent_comms_send.ps1	inbox 读写+轮询(-WaitReply)
常驻桥接	CodeBuddy/bridge/agent-bridge.mjs	扫inbox+调多模态+回写+心跳
看门狗	CodeBuddy/bridge/watchdog.ps1	心跳监控+崩溃自愈
启动入口	start_cb_helper.bat	一键启动看门狗+桥接
看图 Skill	.claude/skills/codebuddy-image.md	标准化看图流程（模型触发）

> **一句话总结**：在两个现成的 AI 之间铺一条标准化的路——协议定清楚、进程保证活着、结果可靠传回。路铺好，看图只是第一个跑在上面的场景。
>
> 关联：完整迭代史见 `chat_memory/topic_growth/` ；踩坑经验见 `memory/experience/cb-multimodal-image-workflow.md` 。