

100 个问题系列

100 Questions Series

关于 WorkBuddy 的 100 个问题



AI磊叔 · 出品

写《100 个问题系列》丛书，初衷很简单

一方面，我想用一种更有趣的方式去收集资料。

很多主题如果只是零散记笔记，最后很容易变成一堆链接、一堆截图、一堆随手记下来的片段，过段时间自己都懒得回看。换成《100 个问题》这种形式以后，资料会慢慢有结构，阅读、搜索、整理、写作，也都会更有抓手。

另一方面，我也想借这个过程，把自己的思考、判断、踩坑经验沉淀下来。

很多时候，真正值钱的是“我怎么理解它”“我在哪些地方踩过坑”“我后来为什么改了看法”。这些东西如果不主动整理，很快就会散掉。

所以这套丛书，既是在收集资料，也是在逼自己把判断写清楚。希望最后留下来的，不只是一些信息汇总，而是一套套带着思考痕迹的主题文档。

怎么用

你可以把它当成一个可直接阅读的主题文档库来看。

可以拿去读，拿去学习，拿去快速建立对一个主题的整体理解。也可以把里面的问题、结构、资料路径继续拿去蒸馏，变成你自己的文章、课程、清单或者研究框架。

如果你在做 Agent、做内容系统、做知识库，或者在写自己的 skill，这个仓库也可以直接当成内容参考。很多主题拆法、章节组织方式、问题设计方式、以及资料转正文的路径，都可以复用。

《100 个问题系列》丛书库

<https://my.feishu.cn/wiki/FC6ZwnwWWi0dWpke56act61gnUd>

我是磊叔，认识一下

<https://my.feishu.cn/wiki/QWQHwA9uYibmtzkZLJBccaEhnNd>

目 录

一、基础认知与定位

1. 为什么很多人装了 WorkBuddy，却最后只把它当聊天框用？
2. 哪三类人最容易在第一周就用出 WorkBuddy 的正反馈？
3. 为什么普通办公任务，往往比开放式研究更适合先上手？
4. 文档里反复强调“可验收结果”，它和普通聊天回复差别到底在哪？
5. 为什么 WorkBuddy 的重点，与其说“会回答”，不如说“把成品交到你文件夹里”？
6. 什么任务交给 WorkBuddy，最容易第一次就跑通？
7. 为什么你一上来就问“你能帮我做什么”，通常拿不到好结果？
8. 怎样判断一个需求算真实工作流，而不是随口试试？
9. 为什么文件处理任务，通常比开放式创作更适合起步？
10. 什么情况下你该先用 WorkBuddy，而不是继续手工做？
11. 为什么很多人第一次失败，问题往往出在任务太宽，而不是模型不行？
12. 怎样判断你手上的哪类重复劳动，已经值得交给 WorkBuddy 接管？
13. 从“偶尔用一下”到“每天会开”，中间通常跨不过哪道坎？
14. 什么样的任务交给 WorkBuddy，反而更容易翻车？
15. 为什么“先给一个真实任务”通常比“先问它能做什么”更容易看出成色？

二、核心架构与关键机制

16. Ask、Plan、Craft 到底分别该在什么时候用？
17. 为什么 Ask 没盘点清楚，就直接进 Craft，最容易误伤文件？
18. 怎样判断一个任务该停在 Plan，而不是直接开工？
19. Auto 模型什么时候够用，什么时候你必须亲自指定模型？
20. 为什么同一个任务，模型一换，交付物的稳定性就会明显变掉？
21. 专家和模型，各自到底负责哪一层判断？
22. 什么任务值得拉专家团，什么任务用单个专家就够了？

- 23. 为什么技能装得越多，不一定代表你更强？
- 24. 怎样判断一个重复流程，到底该不该沉淀成技能？
- 25. 连接器到底砍掉了哪几步最烦的下载、上传和来回切换？
- 26. 为什么连上腾讯文档后，你的工作方式会直接变掉？
- 27. 怎样判断你的任务需要连接器，还是只要本地文件？
- 28. 工作空间为什么既是效率问题，也是安全边界？
- 29. 为什么把整个桌面都交给 WorkBuddy，通常是坏主意？
- 30. 远程助理真正改变的，到底是入口，还是你不在电脑前时的执行方式？
- 31. 自动化为什么必须建立在手动跑通之后？
- 32. MCP 和技能看起来都像扩展能力，它们到底各自解决什么问题？
- 33. 记忆、数据管理和任务归档，为什么会影响你后面还能不能接着干下去？

三、新手踩坑与反模式

- 34. 为什么工作空间一开始没切干净，后面大概率一定出事？
- 35. Ask 盘点现场时，你最该盯住哪几个异常信号？
- 36. 为什么 Plan 里一旦出现“覆盖原文件”，你就该立刻刹车？
- 37. 为什么让它自己补齐缺失数据，往往是在提前埋雷？
- 38. 结果看起来很像真货时，你最容易漏掉什么危险信号？
- 39. 为什么周报和 PPT 做出来了，不等于结论就可靠？
- 40. 为什么表格里的空值，最容易被它处理错？
- 41. 环比、同比和统计口径一乱，最先会把哪些判断带偏？
- 42. 为什么你越说“重新做一个”，结果反而越容易越来越差？
- 43. 怎样提局部修改，才不会把前面做对的部分一起毁掉？
- 44. 为什么连接器授权给得太宽，后面你就很难放心让它动手？
- 45. 手机远程下指令时，你最容易漏掉哪一道确认门？
- 46. 为什么自动化第一次跑通，不代表定时执行时也会稳？
- 47. 哪些批量整理文件的任务，最容易出现误分类和误移动？

- 48. 为什么生成了很多版本却不命名清楚，最后只会更乱？
- 49. 为什么任务步骤一乱，哪怕结果做出来了也很难放心复用？
- 50. 专家团什么时候会从帮手，变成让流程更乱来源？
- 51. 为什么技能装得太杂，反而更容易把任务带偏？
- 52. 自动化里明明有目标空间却选不到时，你第一步到底该查哪里？
- 53. 为什么外发邮件、发布文档这类动作，最后一遍必须人工过？

四、结果验证与稳定性

- 54. 怎样给 WorkBuddy 设一个真正能验收的完成标准？
- 55. 为什么没有输出路径约束，你就很难确认它到底交付到了哪里？
- 56. 如何要求它按阶段汇报进度，才不会一边做事一边刷一堆废话？
- 57. 怎样检查生成报告里的每个结论，都能回到原始数据？
- 58. 为什么把相关性写成果因果关系，是最危险的办公幻觉？
- 59. 遇到读不出来的文件时，你该先查材料，还是先换模型？
- 60. 怎样用“待确认”把风险留在台面上，而不是藏进成品里？
- 61. 为什么你必须检查文件能不能继续编辑，而不只看预览图？
- 62. 怎样判断一个自动化任务，已经可以基本脱手运行？
- 63. 为什么同一个任务换了技能、连接器或模型后，你最好重新做一次验收？
- 64. 升级或重启以后，如果原工作空间看起来“消失了”，你第一步到底该查什么？
- 65. 新接入一个模型后，你最该先比较什么，而不是盲目全量切换？
- 66. 怎样判断一次失败，到底该怪模型，还是该回头重写流程？
- 67. 为什么结果区里的产物，往往比对话框里的“已完成”更可信？
- 68. 怎样给连接器任务补一层“发送前确认”，才不容易闯祸？
- 69. 为什么客户端不在线时，再好的自动化也根本跑不起来？
- 70. 想让结果更稳时，你最该优先补强哪三个环节？

五、真实办公流与场景实战

- 71. 第一次上手，为什么最推荐从 Excel 到周报再到 PPT 这条闭环？

- 72. 怎样把一堆会议记录，整理成真正可追责的待办清单？
- 73. 为什么“资料到网页”这类任务，很适合检验 WorkBuddy 到底能不能真正交付？
- 74. 如何让它做网页时，既能预览结果，又不把事情搞复杂？
- 75. 怎样用连接器，把腾讯文档里的内容直接变成可交付结果？
- 76. 为什么“邮件汇总”比“自动发邮件”更适合作为第一条闭环？
- 77. 怎样把批量重命名做成既省时又不容易丢文件的流程？
- 78. 如何用它把乱文件夹先盘点再归档，而不是直接一把搬家？
- 79. 远程助理最适合救哪三种“人不在电脑前，但事必须马上办”的场景？
- 80. 为什么“周五自动出周报”这种任务，最值得优先自动化？
- 81. 怎样把 Google Calendar 和 Drive 这类外部服务接进来，才不会一开始就卡住？
- 82. 为什么“每次都要重新想指令”的任务，最值得先沉淀成固定流程或 Skills？
- 83. 零代码做本地应用这件事，最适合拿什么小题先试？
- 84. AI 自驱动模式最容易在哪些任务里帮上忙，又最容易在哪些任务里失控？
- 85. 什么场景下它能真省时间，什么场景下只是把手忙脚乱换成眼忙心乱？
- 86. 怎样判断你的业务里，哪个流程最值得成为下一个 WorkBuddy 闭环？

六、边界、替代与未来判断

- 87. 为什么复杂后台系统和正式生产开发，不该因为它“也能做网页和代码”就贸然交给它？
- 88. 什么任务适合留在 WorkBuddy 里做，什么任务应该换更专门的方式？
- 89. 什么场景继续手工处理，反而会更直接也更稳？
- 90. 为什么不是所有资料和目录，都值得一股脑交给 WorkBuddy？
- 91. 连接器和平台接入越来越多时，为什么边界管理会变得更重要？
- 92. 本地工作台、远程助理和平台接入放在一起后，真正变重的管理成本是什么？
- 93. 模型越来越多时，你该怎么避免把“换模型”当成唯一解法？
- 94. 技能、MCP、连接器都在扩张时，普通用户最该先守住哪条授权边界？
- 95. 为什么真正拉开差距的，常常是任务结构和验收方式，而不是提示词？

96. 什么情况下, WorkBuddy 会从一次性尝鲜, 变成你固定会开的工作台?
97. 你到底该不该把对外发送、正式发布、直接改文件这类动作交给它?
98. 为什么功能看起来很多, 不代表你一开始就该全都用上?
99. 如果只打算先学一部分, 你最该先守住哪些基础能力?
100. 这 100 个问题, 最后到底该帮你避开什么误用和误判?

七、参考资料

- 一、WorkBuddy 官方文档
- 二、腾讯云开发者社区 (cloud.tencent.com)
- 三、Tencent Cloud 英文技术站 (tencentcloud.com)
- 四、社区评测与集成文档

一、基础认知与定位

1. 为什么很多人装了 WorkBuddy，却最后只把它当聊天框用？

很多人第一次打开 WorkBuddy，脑子里想的还是“我来问它一个问题，看它答得聪不聪明”。这种起手方式本身就把它用窄了。因为 WorkBuddy 真正擅长的，是接住一条完整任务，然后自己去拆步骤、读材料、调用能力，最后把结果交出来。你如果一直拿它当问答工具，它当然也能答，但你看到的就只是最浅那层。它真正值钱的地方，恰好是聊天框后面那一整段执行链路。很多人倒不觉得它没用，只是根本没把它放进真实工作里。

2. 哪三类人最容易在第一周就用出 WorkBuddy 的正反馈？

最容易马上感受到效果的，通常是三类人。第一类是天天和文件打交道的人，比如要整理文档、改表格、做周报的人，这类任务边界清楚，结果也容易验收。第二类是重复流程很多的人，比如每周都要汇总数据、整理会议纪要、生成固定格式材料的人，因为一跑通就能立刻省时间。第三类是本来就习惯把工作说清楚的

人，比如产品、运营、项目角色，他们通常更会提任务目标、输出格式和限制条件，所以更容易让 WorkBuddy 一次跑准。反过来，需求模糊、场景太散的人，第一周往往更容易挫败。

3. 为什么普通办公任务，往往比开放式研究更适合先上手？

因为普通办公任务通常更容易定义清楚“做完长什么样”。比如整理一批文件、把 Excel 变成图表、生成一份周报，这些任务都有明确输入、明确输出、明确验收标准。WorkBuddy 在这种场景里很容易体现价值，你也能快速看出它到底帮你省了哪一步。开放式研究就不一样了，资料边界更散，判断空间更大，结论也更容易见仁见智。新手一开始就拿这种题去试，最后经常卡住的地方，倒不一定是它做不了，更可能出在你很难判断它算不算做好。先从边界清楚的任务起步，更容易建立对这个工具的真实判断。

4. 文档里反复强调“可验收结果”，它和普通聊天回复差别到底在哪？

差别就在于，聊天回复停在“告诉你怎么做”，可验收结果停在“已经帮你做完了什么”。普通聊天工具给你的，常常是一段文字、一套建议、一个思路，接下来

你还得自己复制、粘贴、排版、补数据、建文件。WorkBuddy 想强调的是另一件事：结果应该直接落成文档、表格、网页、PPT，或者至少已经进入你能继续处理的工作文件里。这样你验收的时候，看的是文件能不能打开、数字对不对、格式能不能接着改，而不是一句“我已经完成了”。这两个体验差得很远。

5. 为什么 WorkBuddy 的重点，与其说“会回答”，不如说“把成品交到你文件夹里”？

因为大多数办公场景，真正消耗人的，不是那几句话，是从“有思路”走到“有成品”的那一段路。你拿到一个回答以后，还要开文档、调格式、整理数据、放进目录、确认版本，这些零碎动作很耗神，也最容易出错。WorkBuddy 把重点放在交付成品，就是想把这段路缩短。它给你的远不止一个方向，它尽量把方向变成一个你已经能拿去用、拿去改、拿去发的文件。很多人第一次觉得它和普通 AI 不一样，往往不是因为它说得更漂亮，关键在于它真的在目录里留下了东西。

6. 什么任务交给 WorkBuddy，最容易第一次就跑通？

第一次最容易跑通的任务，一般都有三个特征：材料已经在手里，目标结果说得清楚，过程中不需要太多外部判断。比如“把这份 Excel 做成周报和 PPT”“整

理这批会议记录并提取待办”“按规则重命名一批文件”，都属于很好的起手题。

它们的共同点是输入清楚，输出也能直接验收。你不需要先争论观点，不需要先补很多背景，也不用冒太大风险。先把这种题跑顺，后面你再去碰连接器、自动化、远程助理，心里会更有数。

7. 为什么你一上来就问“你能帮我做什么”，通常拿不到好结果？

因为这个问题太大，也太空。它逼着 WorkBuddy 先给你一份功能菜单，而进不了任务。你最后看到的多半是“我可以帮你整理文件、处理表格、生成文档”这种介绍，听上去挺全，但对你眼下该怎么用帮助不大。更有效的方式，是直接把真实工作丢给它，让它先围绕这件事动起来。比如你手头真有表格，就让它先盘点表格；真有会议记录，就让它先提任务清单。这样你看到的就不是宣传语了，你能看到它在你场景里到底能不能接住。

8. 怎样判断一个需求算真实工作流，而不是随口试试？

最简单的判断方法就是看这件事有没有完整链路。真实工作流通常至少包含四个部分：要处理什么材料，准备产出什么结果，结果放到哪里，谁来验收。只要这

四件事能说清，基本就不是随口试试了。反过来，如果你只是说“帮我看看这个怎么样”“帮我分析一下”，没有材料边界，也没有交付目标，那它更像体验功能，不像真正工作。WorkBuddy 在工作流里最有价值，因为它吃的是从输入到输出的一整段过程，不只是中间那句回答。

9. 为什么文件处理任务，通常比开放式创作更适合起步？

因为文件处理任务天然带着约束。文件名、格式、目录、字段、内容范围，这些东西都在帮你把任务收窄。你一旦收窄了，WorkBuddy 就更容易跑稳。开放式创作倒也不是不能做，只是它对提示、背景、审美、判断的要求更高，第一次很容易变成你和它一起反复改。文件任务就没有那么多虚的空间，做得对不对，打开看一眼基本就知道。所以很多新手会以为 WorkBuddy 最适合写文案，其实从上手难度看，先拿它处理文件、表格、纪要，通常更容易建立信心。

10. 什么情况下你该先用 WorkBuddy，而不是继续手工做？

当一件事你已经不是第一次做，而且每次步骤都差不多，这时就该优先想到 WorkBuddy。因为手工做最大的问题，不在于慢，在于重复注意力消耗。你明明

知道怎么做，却还得重新打开文件、重新整理、重新排版、重新核对。另一种适合先交给它的情况，是材料很多、格式很杂，但结果相对标准。像多份文档汇总、表格整理、资料转报告，这种事交给它比你自己从零搬要划算。真正不适合的是高风险、强判断、一步走错就会出事的动作，那种先别急着全交。

11. 为什么很多人第一次失败，问题往往出在任务太宽，而不是模型不行？

因为任务一宽，模型再强也只能先猜你的重点。它不知道你最在意哪个文件，不知道输出格式是不是固定，不知道哪些内容能动、哪些不能碰。结果看上去像在认真干活，其实从第一步就已经偏了。很多所谓“它不行”的时刻，往下拆都是任务边界没说清：目录没限定、结果没限定、禁区没限定、验收标准也没限定。WorkBuddy 这种会继续执行的工具，越是有动作能力，越怕起手太宽。边界一模糊，它可不会少做，反倒可能多做、乱做。

12. 怎样判断你手上的哪类重复劳动，已经值得交给 WorkBuddy 接管？

你可以先看两个信号。第一个是，这件事是不是总在重复出现，而且每次做的时候你都觉得没什么新意，只是机械执行。第二个是，它是不是已经有了相对稳定

的输入和输出。只要同时满足这两点，通常就值得尝试接管。比如周报整理、会议纪要提炼、批量重命名、资料转网页，这些都很典型。真正不值得急着交出去的，是每次背景差异都很大、判断标准老在变的事。那类任务你现在就算勉强交给它，后面也大概率要自己返工。

13. 从“偶尔用一下”到“每天会开”，中间通常跨不过哪道坎？

很多人卡住的，倒不在安装，也不在第一次试用，卡点在于始终没找到一个稳定复用的场景。你偶尔用一下，它像个新鲜工具；你每天会开，说明它已经嵌进了你的工作节奏。这个坎通常出现在这里：你有没有把一个高频任务跑顺，并且知道下次还可以原样复用。只要没跨过去，你就会一直停留在“有需要时想起它”。一旦跨过去，WorkBuddy 就不再是你偶尔试试的工具，它会变成你开工前就会顺手打开的工作台。

14. 什么样的任务交给 WorkBuddy，反而更容易翻车？

最容易翻车的，通常是三类任务。第一类是边界没讲清，但动作权限又很大的任务，比如让它直接整理整个桌面。第二类是对外后果很重的任务，比如自动发邮

件、正式发布文档、直接覆盖重要文件。第三类是你自己也说不清验收标准的任务，你都不知道什么叫做好，它当然更容易跑偏。这些任务倒不是绝对不能做，关键在于不能一上来就放手。先 Ask，先 Plan，先看它准备动哪些东西，再决定要不要让它继续往下跑。

15. 为什么“先给一个真实任务”通常比“先问它能做什么”更容易看出成色？

因为真实任务会逼出它的完整能力链。你一旦给到具体材料、具体目标、具体结果，WorkBuddy 就得开始规划、读取、处理、输出，你很快就能看出它到底只是会说，还是能交付。反过来，功能介绍永远是好听的，只有任务才会暴露真水平。它能不能认出文件、会不会乱动目录、生成结果能不能接着用、你不在电脑前时能不能继续跑，这些都只有进真实 workflow 才看得出来。所以想判断它值不值得进你的日常工作，别先问菜单，先扔一件正经活进去。

二、核心架构与关键机制

16. Ask、Plan、Craft 到底分别该在什么时候用？

这三个入口最大的区别，不在于名字，真正分高下的是动作深度。Ask 更像先把问题问清楚，适合知识咨询、材料理解、方案比较这类先拿判断、暂时不急着手动文件的任务。Plan 适合风险更高、步骤更长的事，先让它把执行路径、会碰哪些文件、可能产出什么列出来，你看过再决定要不要继续。Craft 则是直奔着成品去的，默认它会读取材料、处理文件、生成结果。很多人用乱，往往是把所有事都塞进 Craft，结果本来只想先讨论一下，它却已经开始动手。选对模式，等于先把执行权收在正确的位置上。

17. 为什么 Ask 没盘点清楚，就直接进 Craft，最容易误伤文件？

因为 Craft 的默认心态就是去完成任务，不是陪你继续确认边界。只要你的描述里还带着模糊空间，它就会自己补全理解，然后沿着这个理解去读文件、改内容、产生成果。问题就在这里，很多人脑子里其实还没想清楚“处理哪一批文件”“哪些只能看不能改”“结果要落在哪个目录”，却已经给了一个带动作指令的

任务。对聊天工具来说，这只是回答偏了；对 WorkBuddy 这种会执行的工具来说，就可能变成多读、多改、多生成。先在 Ask 阶段把范围盘清楚，后面进 Craft 才更稳。

18. 怎样判断一个任务该停在 Plan，而不是直接开工？

你可以先看三件事。第一，这个任务会不会碰重要文件，尤其是覆盖、归档、批量整理这类不可逆动作。第二，任务里有没有多个分支判断，比如先筛选、再汇总、再生成几种不同产物。第三，你自己是不是还想先看它打算怎么做。如果这三件里中了两件，停在 Plan 往往更划算。因为 Plan 的价值，并不在拖慢执行，它真正的作用是把隐性的风险翻到台面上。你先看一眼步骤、输入、输出和目录安排，很多后面的大返工就能提前省掉。越是长链路任务，先看计划越像保险。

19. Auto 模型什么时候够用，什么时候你必须亲自指定模型？

日常办公里，Auto 往往已经够用了，尤其是资料整理、文档生成、表格分析、常规问答这类边界比较清楚的任务。因为 WorkBuddy 本来就会按任务类型去

挑更合适的模型，省掉你每次手动切换的心智负担。但一旦你开始在意某种非常明确的能力，就该自己指定了。比如你更看重推理深度、多模态识别、特定供应商的中文表现，或者团队已经验证过某个模型在固定流程里更稳，这时继续交给 Auto，收益就没那么大。Auto 适合通用效率，手动指定适合明确优化。

20. 为什么同一个任务，模型一换，交付物的稳定性就会明显变掉？

因为对 WorkBuddy 来说，模型绝非皮肤那么简单，它更像是底层驾驶员。你看到的任务描述、技能、连接器都没变，但真正负责理解指令、取舍重点、组织内容、决定什么时候调用工具的，是具体那一个模型。模型一换，节奏就会换。有的更敢主动补全，有的更保守；有的擅长长文本整理，有的在表格和图片上更稳；有的输出很快，但细节松。放在普通聊天里，这种差异只是回答风格不一样；放在要生成文件和交付物的场景里，就会直接体现为稳定性差异。所以一条跑通的流程，最好别频繁换底座。

21. 专家和模型，各自到底负责哪一层判断？

模型更像发动机，专家更像驾驶方式。模型决定理解能力、推理能力、上下文承载能力和工具调用上限，专家决定的是它会用什么视角、什么流程、什么行业习

惯去处理任务。比如同样一份材料，换一个模型，可能是细节完整度不同；换一个专家，可能连切入方式都会变，产品会先拆需求，运营会先看转化，研究角色会先补背景。很多人容易把这两层混成一层，以为叫了专家就等于换了脑子，其实专家更像在现有模型之上套了一套专业工作法。两层都重要，但作用点不一样。

22. 什么任务值得拉专家团，什么任务用单个专家就够了？

单个专家适合有明确主线的问题，比如写一篇内容、做一份分析、整理一类资料，核心判断集中在一个专业视角里，拉一个专家就够了。专家团适合跨角色协作的活，比如你既要调研、又要结构化方案、还要考虑落地执行，这时让团长拆分给不同角色并行做，会比一个角色硬撑到底更顺。文档里也明确提示了，专家团通常积分消耗更高，所以不能把它当默认选项。真正值得拉团的理由，从来不该是“我想更高级一点”，它得落在任务天然就有分工上。没有分工的任务，强拉多人，常常只会多出协调成本。

23. 为什么技能装得越多，不一定代表你更强？

因为技能绝非收藏品，它会介入执行链路。装得越多，表面上像选择更多，实际上也意味着上下文里多了更多可能被调用的路径。文档里专门提醒过，只启用当

前任务所需的技能，可以减少无干扰和误调用概率。这个提醒非常关键。很多人把技能市场当插件商店，先铺满再说，结果后面任务一复杂，自己都搞不清它到底会走哪条路。真正强的用法不在装得多，关键在于知道这一题该开哪几个、该关哪几个。少而准，往往比全而杂更稳。

24. 怎样判断一个重复流程，到底该不该沉淀成技能？

先看这个流程是不是已经跑顺了。很多人太早做技能，手工流程都还没稳定，就急着封装，最后只是把混乱固定下来。真正值得沉淀的流程，通常有三个信号：步骤比较固定、输入输出相对稳定、你已经重复做了不止几次，而且每次都在做差不多的判断。到了这一步，技能才有意义，因为它是在复用成熟路径。反过来，如果任务背景每次都变，或者你自己还在摸索最优做法，那它更适合继续手动跑。技能适合固化共识，不适合替你提前发明共识。

25. 连接器到底砍掉了哪几步最烦的下载、上传和来回切换？

它砍掉的，可不是某一个按钮，它砍掉的是整段搬运链。以前你要先去外部系统找文件，下载到本地，再拖进工具里，处理完以后还要重新上传、贴链接、确认

格式有没有跑掉。连接器把这几步接成了一条线，让 WorkBuddy 直接去外部服务里读、取、写、搜。对人来说，最烦的一直不是分析那一下，材料在不同系统之间来回搬才最磨人。连接器一接上，这段低价值动作会明显变少，你的工作重心会从“把资料搬过来”变成“让结果往前走”。这就是它最实际的价值。

26. 为什么连上腾讯文档后，你的工作方式会直接变掉？

因为一旦文档不需要先落地到本机，你处理资料的起手动作就变了。以前是先下载，再整理，再上传；连上腾讯文档以后，你更容易直接围绕在线材料发任务，让它去搜、读、汇总、生成、回写。这个变化看起来只是少了几个操作，实际影响很大。它把工作入口从“文件副本”换成了“源文档本身”，很多格式、图表、复杂表格也更容易保住，不用每次担心导来导去后内容变形。工具一旦和资料源头打通，人就更容易把它当工作台，而不是临时加工站。

27. 怎样判断你的任务需要连接器，还是只要本地文件？

看资料是不是长期活在外部系统里。如果你的材料就是桌面上的几份文件，这次处理完也不需要回写，本地文件已经够了，简单直接，边界也清楚。但如果任务

离不开在线文档、邮箱、知识库、会议资料，或者结果还得回到原系统继续流转，那连接器就不止是加分项，它已经成了主链路。再简单一点说，凡是你觉得“来回下载上传很烦”的任务，都值得先想想能不能接连接器。判断标准不要太玄，就是看这件事的真实现场在哪里。现场在线上，就别强行搬回本地再做。

28. 工作空间为什么既是效率问题，也是安全边界？

很多人先把工作空间理解成项目整理工具，这当然没错，但它还有另一层更硬的作用，就是给任务画边界。放在同一个工作空间里，历史任务、结果文件、上下文会更容易串起来，继续处理也更顺，这属于效率层。另一层是安全层。你把任务限定在某个工作空间里，等于在告诉 WorkBuddy：这次主要就在这里看、这里做、这里产出。范围一旦明确，误读别的目录、误碰无关文件的概率就会低很多。所以工作空间看上去像管理功能，实际上也在帮你收住执行半径。对会操作文件的 Agent 来说，这个边界很值钱。

29. 为什么把整个桌面都交给 WorkBuddy，通常是坏主意？

因为桌面从来不是一个干净任务区，它往往混着临时文件、旧版本、截图、下载件、私人材料和各种你自己都快忘掉的残留物。你把整个桌面交出去，等于默认

它要在一堆噪音里自己判断什么相关、什么不相关。这个判断一旦做错，轻则多读无关材料，重则把本不该碰的东西也带进任务里。更现实的一点是，桌面里的文件命名通常也不规范，很难形成稳定流程。真正稳的做法，是给它一个专门目录，哪怕只是临时新建一个文件夹，也比直接把桌面整盘交出去安全得多。

30. 远程助理真正改变的，到底是入口，还是你不在电脑前的执行方式？

更大的变化其实在后者。手机上发一句话看起来像只是换了个入口，但助理真正有意思的地方，是你离开电脑之后，任务还能继续在那台机器上跑，结果再回到你手里。它把“必须坐在工位前才能推动工作”这件事松开了一点。你在路上、在会场、在外面临时想到一个活，都能先丢过去，让电脑端去执行。文档里也提醒得很清楚，电脑必须保持开机并运行 WorkBuddy，所以它不是云端魔法，更像把你的工作电脑远程接出去了。入口变了，执行节奏也跟着变了。

31. 自动化为什么必须建立在手动跑通之后？

因为自动化会放大稳定流程，也会放大错误流程。你手动跑的时候，看到它哪里理解偏了、哪一步目录没设对、哪个结果格式还不稳，还能当场补一句话把它拉回来。一旦进了自动化，系统只是按时间触发、按既定设定执行，它不会替你重

新思考这个流程到底成不成熟。文档里把自动化明确定位在周期性、重复性任务上，这个前提很重要，说明它适合接管已经验证过的事。先手动跑顺，再自动化，逻辑上像先把模具打好再量产。模具没打稳，量越大，返工越痛。

32. MCP 和技能看起来都像扩展能力，它们到底各自解决什么问题？

技能解决的是“怎么做更像一个成熟流程”，MCP 解决的是“能不能接到外部工具和数据”。技能更偏工作方法和任务套路，比如把一类动作封装成可复用能力，让它在对话里更会干某种活。MCP 则更像接口层，把文件读写、数据库、通知、业务系统这些外部能力接给模型，让它手能伸出去。两者看起来都叫扩展，但方向不同。技能是在内部把做事方式打磨细，MCP 是向外把工具边界拉开。你要提高执行手法，用技能；你要连系统、连数据、连服务，就得上 MCP。

33. 记忆、数据管理和任务归档，为什么会影响你后面还能不能接着干下去？

因为 Agent 真正难的，从来都不在“做完一次”，难在下次还能接上。记忆负责把一些长期偏好和背景留住，让它不用每次都从零认识你。任务归档和数据管理负责把历史结果、已分享文件、旧任务状态收好，不至于过几天你只记得“之前

做过”，却找不到做到了哪一步。文档里把这些功能放在看起来很像后台设置的位置，很多人会忽略，但一旦任务开始变多，它们就直接决定你是继续累积，还是不断重开新局。能接着干，是办公 Agent 真正的后劲。

三、新手踩坑与反模式

34. 为什么工作空间一开始没切干净，后面大概率一定出事？

因为工作空间决定的，从来都不在美观，它真正决定的是这次任务默认能接触到什么材料、会把结果落到哪里。你起手没切干净，等于把上下文、历史文件、旧结果和无关资料混在一起交给它。前面看起来只是小问题，后面一旦任务开始读取目录、引用旧文件、生成多个版本，就会逐步放大。很多翻车都不是某一步突然错了，问题往往出在从一开始现场就脏。WorkBuddy 越像真实执行者，越依赖现场清晰。工作空间没切干净，后面哪怕结果看着像对，心里也很难真正踏实。

35. Ask 盘点现场时，你最该盯住哪几个异常信号？

最该盯的是三类信号。第一类是范围含糊，比如“这些文件”“那个目录”“最近的数据”，说的人以为很清楚，系统并不清楚。第二类是权限过大，比如任务里还没讲清楚，却已经允许它批量整理、覆盖、归档。第三类是目标摇摆，一会儿要报告，一会儿又想要表格和 PPT，还没排优先级。Ask 阶段最值钱的地方，就

是把这些模糊点提前抠出来。你看到这种信号还往下走，后面就不只是做得快不快的问题了，它会在错误前提上非常认真地执行。

36. 为什么 Plan 里一旦出现“覆盖原文件”，你就该立刻刹车？

因为这说明任务已经从“生成新结果”进入“改动原资产”了。两者风险完全不是一个级别。生成新文件就算不满意，删掉重来就行；覆盖原文件意味着旧版本可能直接消失，很多上下文也一起没了。更麻烦的是，用户脑子里常常默认“它会帮我小心处理”，但系统看到的是一个允许执行的动作。Plan 阶段一旦出现这种表述，你至少要先把备份、版本命名、目标文件和允许改动范围讲清楚。能新建副本解决的事，尽量别直接覆盖。先刹车，不是保守，是给自己留回头路。

37. 为什么让它自己补齐缺失数据，往往是在提前埋雷？

因为“补齐”听起来像是在帮忙，实际是在替你做关键假设。只要数据里有空洞，它就必须猜：这个值该延用上一期、按均值填、按常识估，还是干脆省掉。你不明确，它就会自己选一种最像能继续干下去的方式。问题是，这种选择一旦进了后续图表、结论和判断，整条链都会被带偏。很多人最后觉得报告写得挺顺，但

数字哪里怪怪的，根源就在这里。缺失数据最稳的做法，从来都不该是默认补齐，应该先标出来，再决定怎么处理。能看见空白，比自信地填错更安全。

38. 结果看起来很像真货时，你最容易漏掉什么危险信号？

最容易漏掉的是“它已经形成了完整表面，所以你不再追底层依据”。排版整齐、标题像样、图表也有了，人很容易默认它靠谱。可真正危险的信号往往藏在里面，比如口径没说明、引用关系不清、图表和正文结论对不上、某些关键数字没有来源。外观越成熟，越容易骗过第一眼审查。WorkBuddy 这类工具最需要防的，从来都不是明显胡来，麻烦在于它会做出一份九成像真的东西，让你少看那最后一成。新手最该练的也不该是挑文风，该先练找依据、找来源、找算式。

39. 为什么周报和 PPT 做出来了，不等于结论就可靠？

因为周报和 PPT 天生偏展示，不偏校验。它们擅长把信息压缩得更像结论，但这一步也最容易把原始数据里的犹豫和例外一起抹平。你看到的是一页页整齐的叙述，它看到的往往只是把已有材料组织成更像成品的形式。可“能展示”不等于“经得起追问”。尤其是汇总类任务，只要源数据口径不一、时间范围不准、

样本不完整，最后做出来的周报和 PPT 都可能只是包装良好的半成品。真正稳的顺序应该是先审底层数据，再看展示逻辑。先信版式，很容易吃亏。

40. 为什么表格里的空值，最容易被它处理错？

因为空值在表格里从来不只代表一种含义。它可能是真没有、还没填、字段不适用、被系统漏采，也可能只是格式丢了。可模型看到空白时，天然会倾向于把它当成需要继续处理的缺口，而不是一个待判断的信号。于是有时它会跳过，有时会补零，有时会按上下文猜一个最像的解释。错就错在这里，表格里最不起眼的空格，背后常常是业务定义问题。你不先讲清楚，后面的排序、汇总、占比、趋势都可能跟着跑偏。空值不是小问题，是表格任务里最容易被低估的炸点。

41. 环比、同比和统计口径一乱，最先会把哪些判断带偏？

最先带偏的通常是趋势判断和优先级判断。环比看短期变化，同比看周期位置，统计口径决定你拿来比较的到底是不是同一类东西。这三个只要有一个乱了，后面“增长”“下滑”“异常”“回暖”这些话都可能站不住。更麻烦的是，这类错误会很伪装，因为数字本身未必是假的，只是比较关系错了。你一旦在错误口径上继

续做资源分配、问题归因、行动建议，整个判断链都会很顺，但方向可能完全错。

对 Agent 来说，算出来不难，先确认你到底在比什么，才是真的难点。

42. 为什么你越说“重新做一个”，结果反而越容易越来越差？

因为这类话听上去像重置，实际上很容易变成把原来那些模糊点重新再模糊一遍。

它不知道你是要彻底推翻结构，还是只是不满意其中几段；不知道哪些内容能保留，哪些必须丢掉。于是每重做一次，它都可能在旧理解、新猜测和已有结果之间重新拼一版。次数一多，漂移就出来了。更好的提法，通常都不该是“重新做一个”，应该明确指出哪里不对、哪部分保留、哪部分重算、输出格式是否不变。

局部问题局部改，比一遍遍全盘重来更容易守住质量。

43. 怎样提局部修改，才不会把前面做对的部分一起毁掉？

关键是把“改哪里”和“别动哪里”同时说出来。很多人只会说“把这个改一下”，但没补后半句，系统就得自己判断改动范围，结果经常把相邻结构、版式甚至结论一起重写。更稳的提法通常会同时包含四件事：要改哪一页或哪一段，改成什么，哪些内容保持不变，输出仍然保存到哪里。你等于是给它画一个局部施工

区。这样它会更像在修补成品，而不会重新理解整份任务。对已经跑顺的结果，最怕的并不是改不动，真正怕的是改动边界没画出来。

44. 为什么连接器授权给得太宽，后面你就很难放心让它动手？

因为你一旦不知道它理论上能看到多少、能动多少，后面每次执行都会带着心理负担。连接器本来是为了少搬运、少切换，但权限范围如果比任务需要大很多，它带来的就不只是方便，还有不确定性。尤其是邮箱、知识库、在线文档这种系统，很多内容并不属于当前任务现场。你授权过宽，哪怕这次它没出错，下次让它自动处理时也很难彻底放心。最好的状态，是权限和场景大体对齐，能读什么、能写什么、能搜什么，你自己心里有数。能放心动手，本身也是效率的一部分。

45. 手机远程下指令时，你最容易漏掉哪一道确认门？

最容易漏掉的是“这句话落到电脑端以后，到底会变成什么动作”。在手机上发消息很轻，容易产生一种只是随口交代一下的错觉。可远程助理接住的，压根不是聊天氛围，它接住的是一条可执行任务。你不在电脑前，就更难及时看见它读了哪个目录、用了哪个连接器、结果写到了哪里。很多人远程下指令时只确认了

目标，没有确认现场和边界。真正该补的一道门，是在脑子里多问一句：如果它现在立刻开工，会碰哪些东西？这句问出来，很多过宽指令就会自动收住。

46. 为什么自动化第一次跑通，不代表定时执行时也会稳？

因为第一次跑通，往往发生在你还盯着它的时候。目录对不对、数据有没有更新、连接器有没有失效、输出格式有没有小问题，你都能当场看见。可一到定时执行，环境就变了。数据可能还没到齐，外部服务权限可能过期，文件命名可能撞车，甚至同一个目录里的旧文件都会影响结果。自动化的难点，从来都不在“能不能跑”，难在“无人值守的情况下还能不能稳定跑”。第一次成功，只能说明流程有希望，不代表它已经具备长期运行条件。

47. 哪些批量整理文件的任务，最容易出现误分类和误移动？

最容易出问题的，是规则看起来简单、实际判断很暧昧的那类任务。比如按主题分、按用途分、按“是否重要”分，这些词在你脑子里也许很清楚，但落到文件名、内容和目录时，边界并不硬。还有一种高风险场景是同名文件多、旧版本多、格式混杂，你以为它会看懂版本关系，实际上它只能根据能看到的线索做判断。

批量任务一旦判断标准不硬，就会把少量误判放大成整批混乱。最稳的分类规则，通常都能写成清楚的文本，不靠感觉。

48. 为什么生成了很多版本却不命名清楚，最后只会更乱？

因为版本一多，真正稀缺的就不在文件本身了，稀缺的是你对文件关系的把握。哪份是初稿，哪份是改完的，哪份只是临时导出，哪份能对外发，这些信息如果不进文件名，过几轮以后只能靠回忆猜。WorkBuddy 很会生成结果，也很容易在迭代过程中留下多个产物，所以命名反而比手工时代更重要。命名不清楚，后面继续改、归档、分享、自动化接力都会变难。版本管理看起来像小事，其实是在保护你对结果链条的控制感。文件越多，越不能偷懒。

49. 为什么任务步骤一乱，哪怕结果做出来了也很难放心复用？

因为复用靠的，从来都不是偶然做对一次这么虚的东西，你得知道它为什么对。步骤乱的时候，结果也许还能碰巧可用，但你很难判断下次换一批材料、换一天的数据、换一个同事来接，会不会立刻失效。真正能复用的流程，背后一定有相对稳定的顺序、明确的输入和可解释的中间环节。WorkBuddy 的强项之一是能

把任务继续推进，可如果推进过程本身就混乱，那你复用下来的压根不是方法，只是运气。结果做出来只是第一关，能不能放心重跑，才决定它是不是一个流程。

50. 专家团什么时候会从帮手，变成让流程更乱的来源？

当任务主线本来就很单一，却硬拉多角色进来时，专家团就容易从加速器变成噪音源。因为它的价值建立在分工之上，任务如果不需要分工，多出来的角色只会带来更多视角、更多话语方式、更多整合成本。另一个常见问题是，用户自己也没讲清楚核心目标，却希望专家团替他一起想，结果团长拆得很热闹，最后交回来的东西反而更散。专家团最怕用在“我也不知道我要什么，但你们多想想”的场景。角色越多，越需要主线清楚。

51. 为什么技能装得太杂，反而更容易把任务带偏？

因为技能一多，系统在执行时可走的路也更多，而这些路不一定都适合当前任务。你以为自己是在增强能力，实际是在增加岔路。尤其当几个技能都和“写文档”“整理资料”“生成内容”沾边时，它更容易在边界相近的能力之间摇摆。文档里建议只启用当前任务需要的技能，这可不光是为了整洁，更实在的用意是降低

误调用概率。对新手来说，先把技能池收窄，比盲目扩张更重要。能明确知道这次为什么开它、不开另一个，才算真的进入了会用的阶段。

52. 自动化里明明有目标空间却选不到时，你第一步到底该查哪里？

第一步通常既不该去怀疑模型，也不该急着重建任务，应该先回到工作空间和数据状态本身去查。自动化对目录的要求比手动任务更稳定，它需要明确可见、可写、可持续存在的位置。如果目标空间选不到，常见原因往往是工作空间没有正确保存、目录状态变了、当前入口看到的并不是你以为那个项目，或者系统默认给你分配了新的 `automation` 工作空间。先查空间是否真的存在、是否在当前列表里、是否还在正确位置，这比盲目重试更有效。很多所谓“选不到”，其实是现场没有对齐。

53. 为什么外发邮件、发布文档这类动作，最后一遍必须人工过？

因为这类动作一旦发出去，错误的成本就不再只是一份内部返工。邮件可能发错对象，文档可能暴露未清理内容，措辞里的小问题也可能变成对外风险。

WorkBuddy 很适合帮你准备这些东西，甚至可以把大部分机械步骤提前做完，

但最后那一下最好还是人来过。这倒不一定会错，问题在于对外动作的责任边界和内部整理完全不同。把最后一道门留给人，就是把高后果动作和高效率流程拆开。前面可以让它跑，最后一下不要省。

四、结果验证与稳定性

54. 怎样给 WorkBuddy 设一个真正能验收的完成标准？

真正能验收的标准，至少得说清三件事：要生成什么、用什么格式、放到哪里。

比如“把这周三个数据源汇成一份周报，Word 格式，存到周报目录”。结果区正好给你对照依据——产物视图看文件有没有生成，变更视图看它动了哪些东西，全部文件视图看结果落没落进对的目录。标准越具体，验收越省力。模糊的标准最后都会变成你打开文件后还要再猜一次“这到底算不算做好了”。

55. 为什么没有输出路径约束，你就很难确认它到底交付到了哪里？

交付最后一定要落成一个具体文件。你不告诉它存到哪，它就会自己挑一个地方。等你回头想验收，并不知道那个文件在哪，看到产物也很难判断它落没落对。输出路径约束的价值，就是让你和它对“结果住在哪里”有一致预期。发任务时顺手指定一个目录，验收就从“翻文件”变成“对路径”。

56. 如何要求它按阶段汇报进度，才不会一边做事一边刷一堆废话？

把汇报节奏和执行步骤绑在一起。与其让它“随时汇报”，不如明确说“分三步做：先盘点数据，再生成图表，最后写结论，每做完一步先用一句话告诉我再继续”。对话区本来就会显示“正在分析数据”这种轻量提示，已经够看进度。你真正要补的，是关键的检查点：哪一步之后你想停下来看一眼中间结果。节点越清楚，汇报越有用；没有节点的汇报，多半只是噪音。

57. 怎样检查生成报告里的每个结论，都能回到原始数据？

最直接的办法，是要求它写结论时带上出处，比如“每个关键结论后面标注对应的数据来源、时间范围或原始字段”。这样你拿到报告就能顺着每条结论回查。结果区的变更视图和全部文件视图也能帮你对照原始数据和加工结果有没有走样。文档专门提示过，对数据可信度有要求时要提前写明是否需要公开来源和出处链接。一份结论回不到原始数据的报告，再好看也只能算半成品。

58. 为什么把相关性写成因果关系，是最危险的办公幻觉？

相关性只是说两件事一起出现，因果关系却说一个导致了另一个。模型很擅长发现“A 涨了 B 也涨了”，但它并不天然知道两者有没有因果链。于是报告里会出现“因为新增用户上升，所以转化率提高”这种话，听着合理，可能只是同一个外部因素同时推高了两个指标。最危险的地方在于，这种结论看着特别像能指导行动，你照着分配资源，方向可能就错了。遇到因果类表述，先反问一句：这是数据证出来的，还是句子顺出来的。

59. 遇到读不出来的文件时，你该先查材料，还是先换模型？

按文档给的顺序，先换模型。文件读不出来，最常见的根因是当前模型不支持这种文件类型，而不是文件本身坏了。要读图片、扫描版 PDF、复杂 Excel，就得切换到支持图片理解或多模态的模型。换了还不行，再回头查材料有没有加密、损坏、改过后缀名。还有一个容易漏的点，是没装对应的 Skill，有些文档处理能力是通过技能市场补进来的。顺序大致是：先换模型，再查插件，最后查材料。

60. 怎样用“待确认”把风险留在台面上，而不是藏进成品里？

做法很简单，就是明确告诉它：凡是拿不准、找不到依据、只能猜的地方，都标出来，不要自己补一个看着像真的答案。比如要求它“对没有明确来源的数据或判断，用【待确认】标注，并列出你还需要我提供什么”。这样成品里的风险点就是看得见的，而不是被一段流畅文字盖过去。权限系统也提供了类似保护——默认权限下敏感操作和工作空间外的执行都需要你确认。每一层你都可以设一个“待确认”的口子。

61. 为什么你必须检查文件能不能继续编辑，而不只看预览图？

预览只证明“这东西能看”，不证明“这东西能改”。结果区的预览很好用，表格、文档都能直接看一眼版式。但预览通过，不代表文件本身是可编辑格式。有时候它生成一个截图、一个网页、一个被锁的 PDF，都能预览，你拿去改就费劲了。真正能往下走的工作流，需要可编辑的源文件——Word 能改字、Excel 能改公式、PPT 能换页面。验收时除了看预览，最好打开文件本身确认格式对不对、内容能不能接着动。

62. 怎样判断一个自动化任务，已经可以基本脱手运行？

至少满足三个条件。第一，流程已经在手动模式下跑顺过不止一次，输入输出都稳定。第二，结果保存路径明确，默认那个 `automation` 工作空间是不是你要的，先确认一下。第三，外部依赖都稳——数据按时到位、连接器授权没过期、模型可用。文档把自动化定位在“周期性、重复性任务”，本身就说明它适合接管已验证过的事。最后还要看你的电脑在它该执行的时候是不是真开着。第一次跑通只能说明流程有希望，无人值守下稳定跑才算真脱手。

63. 为什么同一个任务换了技能、连接器或模型后，你最好重新做一次验收？

这三样东西都直接参与执行链路，换一个，结果就可能变。模型换了，理解能力和输出风格会变；技能换了，工作流程会变；连接器换了，它读到的数据和能写的位置都可能变。之前那次验收是在旧组合下成立的，换完就作废了。最稳的做法是换完跑一次最小化验证——拿一个你已经知道正确答案的小任务，看新组合下还能不能给出一致结果。重新验收一次，比事后返工便宜得多。

64. 升级或重启以后，如果原工作空间看起来“消失了”，你第一步到底该查什么？

先别怀疑系统坏了，也别急着重建任务，先去本地大目录里找。工作空间说到底就是本地的文件夹，升级重启后界面里看不到，多半是入口没对上，不是文件被删了。Windows 去 `C:/Users/用户名/Tencent WorkBuddy`，Mac 去 `/Users/用户名/Tencent WorkBuddy`，按日期命名的文件夹通常还在，重新打开就能找回。还有一种情况是更新时安装目录被清空重写了一这也是为什么文档反复提醒重要文件要同步保存到独立目录。大多数“消失”，其实只是没找对位置。

65. 新接入一个模型后，你最该先比较什么，而不是盲目全量切换？

最该先比较它在你固定任务上的具体表现，而不是它宣传页上的能力清单。能力标记（工具调用、图片输入、推理模式）只告诉你它理论上能做什么，真正决定能不能用的是它在你场景里稳不稳。接入新模型后，挑一两个已验证过结果的任务让它跑一遍，和旧结果对照。重点看三件事：输出格式有没有变、关键结论有没有偏、工具调用有没有失败。文档也提到不同模型各有侧重，这些差异只有在你的真实任务里才会显形。小范围比一遍，再决定要不要扩大使用。

66. 怎样判断一次失败，到底该怪模型，还是该回头重写流程？

按错误来源拆。如果伴随错误码，先看它属于哪一类。文档把常见错误码做了归类：像 14003、11133、1001 这类，多和模型侧状态有关，通常先切换模型重试；像 3002、3003、400、401、504 这类，多和网络环境有关，先查你用的是公司网络还是家庭网络。错误码指向模型，这次大概率是模型的问题；如果反复卡在同一步，换模型也救不回来，那多半是流程有毛病——任务太重、目录没设对、需求里有它理解不了的模糊点。先按错误码归类，再看是不是反复卡同一处，基本就能分清该换模型还是改流程。

67. 为什么结果区里的产物，往往比对话框里的“已完成”更可信？

对话里的话是它自己说的，结果区里的文件是它真正做出来的。一个“已完成”可以很轻，一个落地的文件必须真的被生成、写入、保存，可信度不在一个量级。结果区的四个视图给你从不同角度核对交付：产物视图看生成了什么，全部文件视图看落在哪里，变更视图看它动了什么，预览视图让你不用下载就快速看一眼。

很多翻车都是因为只看对话没看文件——它说做完了，打开一看要么没存对地方、要么格式不对、要么缺一半。相信产物，别只相信话。

68. 怎样给连接器任务补一层“发送前确认”，才不容易闯祸？

连接器最容易闯祸的地方，是它能直接对外部系统动手——发邮件、发消息、改在线文档、写网盘，这些动作撤回成本很高。所以发送类任务，最好在指令里就加一道确认门，比如“邮件准备好后先存草稿，等我确认内容和收件人再发”。

文档也提到，连接器授权后可以随时通过开关禁用或断开，不想要它自动外发时就把对应连接器关掉，只让它读取和准备。把它从“自动执行”降级成“准备好等你点头”，多这一下确认，能省掉很多对外发错的麻烦。

69. 为什么客户端不在线时，再好的自动化也根本跑不起来？

WorkBuddy 的自动化是在你的本地电脑上执行的，不是在云端。定时规则一触发，真正去读文件、调模型、生成结果的是你那台机器上的客户端。电脑关了、休眠了、客户端没在运行，这条链路就断了，再精确的定时也没用。文档专门提了一个防休眠设置，开启后系统不会休眠，就是给远程操控和自动化留出持续运

行的条件。搭自动化的前提，不光是流程跑通，还要保证它在该执行的那个时间点，你的电脑是醒着、在线、跑着客户端的。

70. 想让结果更稳时，你最该优先补强哪三个环节？

第一个是输入端，怎么提需求。三要素写清楚——做什么、有什么材料、要什么结果，再加一个参考样本，比写十行抽象要求管用。第二个是能力端，模型、技能、连接器。读不出文件就换支持对应能力的模型，固定流程跑顺了再沉淀成技能，外部数据够用就别乱接连接器。第三个是验收端，结果区和版本管理。结果区四类视图挨个核对，重要文件提前备份，变更视图接收前先自检一遍。文档推荐的上手顺序也正好印证这个递进：先会提需求，再会分步调整，再熟悉专家和样例，最后涉及真实文件先备份再尝试自动化。

五、真实办公流与场景实战

71. 第一次上手，为什么最推荐从 Excel 到周报再到 PPT 这条闭环？

因为这条链路正好把它最核心的能力都串起来了，每一步都能验收。起点是 Excel，你手上有现成数据，让它按月统计、生成柱状图折线图，考验的是读取和可视化能力。中间是周报，把数据加工成结论，考验归纳和写作能力。终点是 PPT，把内容组织成演示文稿，考验结构化和排版能力。三步递进，前一步的产物正好是后一步的输入，而且都是你能直接打开拿去用的文件。这条闭环虚的空间几乎没有，跑完一遍，你对它的真实水平心里就有数了。

72. 怎样把一堆会议记录，整理成真正可追责的待办清单？

可追责的关键，在于每条待办都绑得上人和时间。很多人整理纪要，最后得到的是一段“大家讨论了什么”的摘要，没法追责，因为它没有行动项。真正能追责的清单至少要三块：每个议题的结论、对应的责任人、截止日期。直接把要求写进指令，比如“整理成清单，包含每个议题的结论、责任人和截止日期，标记有

争议和未决的事项，用表格输出，不需要开场白”。把输出限定成表格，是逼它把每条拆成结构化字段。别忘了把“有争议、未决”单独标出来，这些恰恰是最容易事后扯皮的点。

73. 为什么“资料到网页”这类任务，很适合检验

WorkBuddy 到底能不能真正交付？

因为网页是一个完整的、能打开、能交互的成品，没法靠一段文字糊弄过去。你让它把资料做成网页，它就得真去组织结构、排页面、处理样式，最后还得能运行。结果区的预览视图能直接看效果，不用下载就能确认结构、样式和交互对不对。这一关过了，说明它交付的是真东西；过不了，你一眼看得出来。相比 Word 或 Excel，网页对“能不能用”的检验更硬——Word 容错空间大，少一段也能交，网页打不开或结构乱了就是不行。想快速摸清它的交付底线，拿一个资料到网页的任务试一次，比反复问“你能做什么”准得多。

74. 如何让它做网页时，既能预览结果，又不把事情搞复杂？

先让它把最小可运行的版本跑起来，再逐轮加东西。第一次别要求复杂交互、多页面、精美样式，先生成一个核心功能能跑的版本，比如一个支持新增、搜索、

编辑的简单工具。生成完直接在结果区预览里看效果，确认基本结构没问题，然后用对话继续迭代，一次只加一类能力。文档也是这个建议：先做最小可用版本，把升级需求拆开提，一次只增加一类能力。预览视图让你不用每次开外部工具就能看页面改没改对。小步快跑，比一次堆满更稳。

75. 怎样用连接器，把腾讯文档里的内容直接变成可交付结果？

先接连接器，再发任务，整个过程不用把文件下载到本地。腾讯文档是 WorkBuddy 内置支持的连接器之一，接上之后它就能直接读你在线文档里的内容，按你的要求加工。比如让它“读取腾讯文档里这份项目周报，整理出关键进展、风险和下周计划，生成一份 Word 存到周报目录”。连接器把下载、整理、再上传那几步省掉了，你直接围绕在线材料发任务就行。文档提醒过，连接器需要相应权限，授权要正确，授权后可随时禁用或断开。接之前先确认权限够用，接完先用一个简单读取任务验证链路通不通，再往下做复杂加工。

76. 为什么“邮件汇总”比“自动发邮件”更适合作为第一条闭环？

邮件汇总风险低，自动发邮件风险高，第一条闭环应该挑最稳的。邮件汇总只是让它读取、整理、提炼邮件内容给你看，全程不对外产生动作，就算理解偏了，最坏也就汇总得不够好，重做一次就行。自动发邮件一旦发出去，对象错了、内容错了、措辞不当，撤回都来不及。文档在讲每日简报时也明确建议，先手动执行一次，确认内容、流程和接收都正常，再开启自动化。这个顺序背后的逻辑就是：先验证链路，再放开权限。汇总这条线走稳了，再考虑往外发的那一步。

77. 怎样把批量重命名做成既省时又不容易丢文件的流程？

核心是先预览，再执行，中间卡一道确认门。批量重命名最容易出事的地方，是它一次性改了一堆文件名，等你发现规则不对，已经改完了，很难一个个还原。所以第一步让它先展示重命名预览结果——旧名是什么、新名是什么，列一张清单给你看，确认没问题再真正执行。文档给的示例指令正好是这个思路：“读取文件或属性，按日期加主题加类型的规则批量重命名，并先展示重命名预览结果。”还有个稳妥做法，是先拿一小批样本试一遍，验证规则符合预期，再扩大到全量。规则要写成硬条件，而不是“按重要性分类”这种它只能靠猜的软规则。

78. 如何用它把乱文件夹先盘点再归档，而不是直接一把搬家？

把“盘点”和“归档”拆成两个独立动作。第一步只让它盘点，比如“列出这个文件夹里所有文件，按类型分组显示”，这一步只读不动，结果是一张清单，你能看清楚里面到底有什么。第二步再让它归档，而且要带确认门，比如“把 PDF 移到归档目录，表格移到表格目录，其他不动，执行前先列出待移动清单让我确认”。乱文件夹最大的风险在于内容混杂、命名又不规范，一把搬家很容易把不该动的东西也带走。文档反复强调过，处理重要文件前先备份，执行后优先检查目标目录和回收站。先盘点、再确认、再归档，乱摊子才能变整齐而不丢东西。

79. 远程助理最适合救哪三种“人不在电脑前，但事必须马上办”的场景？

第一种是会议中临时要东西。你人在开会，手里只有手机，但马上需要一份会议纪要或补充材料，在手机上发一句话，电脑端的它就开始干活，做完结果回到你手机上。第二种是出门在外要处理文件，有人催你要一份桌面资料，远程助理能让你直接让电脑去读、去整理、去发送。第三种是通勤路上提前安排，你在路上想到一件待办，先丢过去让它提前准备材料，到公司直接接着用。文档把远程助

理的场景就归纳成这几类：出门在外、会议中、通勤路上、临时需求。它的价值在于把“必须坐在电脑前才能推进工作”这件事松开了。前提是电脑要保持开机并运行客户端，网络也要正常。

80. 为什么“周五自动出周报”这种任务，最值得优先自动化？

因为它同时满足三个条件：周期性强、规则明确、不需要实时人工干预。每周五到了那个点，要做的事几乎一样——汇总本周进展、按模块分类、统计关键数据、生成一份简洁周报。文档也把自动化明确划在“周期性、重复性任务”里，周报正是最典型的代表。而且周报的交付物清楚，是一份能打开的文件，验收起来简单。把它做成定时任务，比如设定每周五 17 点自动运行，它就会按时汇总、生成、保存到指定目录。你只需要回到结果区看一眼产物对不对，不用每周再从头手工磨。先从这种最稳的任务开始自动化，跑顺了再往更复杂的场景扩，风险最小。

81. 怎样把 Google Calendar 和 Drive 这类外部服务接进来，才不会一开始就卡住？

分两步走，先把授权做对，再把任务描述写具体。授权这一步，进入连接器入口，选 Google Calendar 或 Google Drive，按提示完成账号授权，这一步过不了后面什么都跑不起来。常见卡点就是授权没成功，或者你要读的日程、文件不在授权的那个账号下。任务描述这一步，要给清楚时间范围和搜索边界，别只说“看看我的会议”，要说“查看今天的会议安排”，资料多时还要指定文件夹、关键词或会议名称缩小范围。文档的建议也是这样：授权时只授予所需权限，时间范围说清楚，资料多就缩小范围。先让它只查今天的日程验证链路通不通，再往整理资料、生成待办这类复杂任务走。

82. 为什么“每次都要重新想指令”的任务，最值得先沉淀成固定流程或 Skills?

因为你每重想一次指令，就在重复消耗一次注意力，而且每次写出来的还不一样，结果自然飘。这类任务有个共同特征：你做过不止几次，步骤基本固定，但每次都得从头组织一遍怎么说。把它做成 Skill 之后，下次只要触发一个意图，它就按你固化的流程走，不用你再重复交代背景、格式、输出位置。文档给的建议也是从高频动作开始，越高频的重复操作越值得沉淀。前提是这个流程得先在手动模式下跑顺，手工流程还不稳定就急着封装，最后只是把混乱固定下来。顺序是先手动跑稳，再沉淀成技能。

83. 零代码做本地应用这件事，最适合拿什么小题先试？

最适合拿一个功能单一、边界清楚的小工具先试，比如一个本地知识管理工具。

它的需求很克制：支持新增、搜索、编辑、分类，界面简洁，适合个人日常记录。

为什么挑这种？因为核心动作少，你能很快看出它生成的代码能不能跑、跑起来对不对，结果区的预览还能直接看页面效果，不用装一堆环境。文档也是这个建议：先做最小可用版本，先完成核心功能，再逐步加复杂能力。新手最怕一上来就想做个功能很多的应用，结果每一步都在排查报错，走不完一个完整闭环。小工具跑通了，再试模糊搜索、自动推送这些更复杂的功能，每一步都有上一个稳定版本兜底。

84. AI 自驱动模式最容易在哪些任务里帮上忙，又最容易在哪些任务里失控？

帮上忙的，通常是目标清楚、步骤能拆、中间能自检的任务。比如“搜集近一周某领域的动态，整理成简报，附来源链接，输出前自检有没有重复信息”。你把目标、约束、输出格式、完成标准一次说清，它就能自己规划顺序，先搜集、再筛选、最后整理，你不用每步盯着。失控的，多半是目标模糊、边界没给、又涉

及高风险动作的任务。比如只说“帮我把这些资料整理好”，没说整理成什么、放到哪、能不能删，它就会自己补理解，加上自驱动模式下它更敢动手，很容易多删多改多覆盖。文档提醒得很到位：目标越清楚自驱效果越好，复杂任务先给边界，涉及删除、覆盖、迁移先预览方案再执行。

85. 什么场景下它能真省时间，什么场景下只是把手忙脚乱换成眼忙心乱？

能真省时间的场景有几个共同点：任务边界清楚、你已经做过类似的事、交付物能直接验收。比如固定格式数据做周报、一批文件按规则重命名、会议记录提成待办清单。它做完你打开看一眼就知道行不行，省下来的是实打实的机械操作时间。变成眼忙心乱的场景，通常是你给它一个目标很虚、判断标准很软的任务，又忍不住每步盯着它、反复纠正它，看着很忙，大部分精力花在纠偏和重新解释上，最后可能比手工还累。判断标准很简单：这个任务你自己能不能说清“做完长什么样”。能说清，交给它大概率省时间；说不清，先把它定义清楚，别急着交。

86. 怎样判断你的业务里，哪个流程最值得成为下一个 WorkBuddy 闭环？

从四个维度筛。第一是频率，这个流程是不是反复出现，每周甚至每天都得做。

第二是稳定性，输入输出是不是相对固定，每次步骤差不多。第三是交付物，做完是不是有一个能直接验收的结果。第四是风险，里面有没有不可逆的高后果动作，比如对外发送、覆盖原文件。同时满足前三条、第四条风险又可控的，就是最值得优先做成闭环的。文档推荐的自动化适用判断也是这个逻辑：重复性高、规则明确、无需实时人工干预。挑出来之后先手动跑通一遍，验证结果稳定，再考虑沉淀成 Skill 或做成定时自动化。一个闭环跑稳了再挑下一个，比一次性铺开好几个半成品流程更划算。

六、边界、替代与未来判断

87. 为什么复杂后台系统和正式生产开发，不该因为它“也能做网页和代码”就贸然交给它？

能生成代码，跟能承担生产级责任，是两回事。做网页、写脚本、搭本地小工具，它确实能上手，尤其适合验证想法、做原型、处理个人效率工具。但复杂后台系统涉及的是另一套要求：高并发下的稳定性、数据一致性、权限审计、故障回滚、长期可维护性。这些场景里，错一行代码的代价可能是一次线上事故。文档对零代码本地应用的定位也很克制，建议先做最小可用版本、先完成核心功能再扩展，本身就说明它适合探索和验证，不是顶替正式工程链路。用它快速试出原型，再交给正式开发加固上线，是更合理的分工。把它当原型加速器很值，当生产替身风险就大了。

88. 什么任务适合留在 WorkBuddy 里做，什么任务应该换更专门的方式？

留在 WorkBuddy 里做的，是那些需要把读取、加工、生成串成一条链的办公任务。它的强项在于接住一整段 workflow：读文件、理解内容、调用工具、产出可

交付结果。周报生成、会议纪要提炼、数据可视化、批量文件整理、资料到网页，这些都很适合。应该换专门方式的，是边界极明确、已有成熟专业工具的环节，比如复杂的财务建模、正式的代码工程、精密的图表设计，专门工具的深度和精度远超通用 Agent。判断标准很简单：这件事需不需要跨多个步骤、跨多种格式、跨读和写的协同。需要，就留在这里；只是单一环节的深度操作，交给专门工具更顺手。

89. 什么场景继续手工处理，反而会更直接也更稳？

那些判断高度依赖现场经验、又需要随时灵活调整的场景，手工反而更稳。比如和客户谈判时的临场应对、对一份设计稿的审美判断、对复杂人事问题的权衡，这些事的价值在于人当下的感知和取舍，而非执行速度。手工做的好处是，你能随时根据对方反应调整，能把只可意会的判断揉进每个决定里。WorkBuddy 擅长把已定义清楚的事高效执行，但定义本身就模糊、需要边做边定的任务，交给它反而要花更多时间反复对齐。还有一类也适合手工：任务量极小、一次性、不值得搭流程的事。为做一次的事去配连接器、写提示词、设自动化，收不回成本。

90. 为什么不是所有资料和目录，都值得一股脑交给 WorkBuddy？

因为有些东西交出去，换来的往往是风险和混乱，谈不上效率。一类是敏感资料，涉及客户隐私、合同细节、个人信息的文件，放进一个会读取、会加工、可能还通过连接器外发的 Agent 里，边界没控好就是事故。一类是结构极乱、你自己都搞不清内容的目录，比如堆满临时文件的桌面，交给它等于让它在噪音里猜什么相关，猜错就多读多动。文档也专门提醒过，别把整个桌面交给它，给它一个专门目录更安全。判断要不要交，可以问一句：这件事交出去，节省的时间值不值得承担它读错、改错、发错的风险。值，就交；不值，就留在手里。

91. 连接器和平台接入越来越多时，为什么边界管理会变得更重要？

每多接一个连接器，它能看到的资料、能动的系统就多一层，这些能力是叠加的。接一两个的时候，你心里还清楚它能碰什么；接上七八个邮箱、文档、网盘、消息平台之后，你自己都未必记得全。边界管理在这个阶段就变得关键，因为它直接决定你能不能放心让它动手。文档给的安全提示都指向同一件事：连接器需要相应权限，授权后可随时禁用或断开，建议定期检查连接器状态。好的边界管理有几个动作：只授权当前任务需要的权限、用完就关、定期清点还连着哪些服务、把高后果动作留在人工确认那一层。接入是能力扩张，边界管理是让这种扩张不失控的刹车。

92. 本地工作台、远程助理和平台接入放在一起后，真正变重的管理成本是什么？

是边界维护和状态确认这件事本身。单看每一项都不复杂：本地工作台执行，远程助理让你离开电脑也能下指令，平台接入把外部服务接进来。但三者叠加之后，你要管的东西变多了一哪些连接器还连着、哪台电脑在线、哪个工作空间是当前的、哪个自动化该跑、推送有没有真的发到小程序。这些都不是一次性配置就能完事，得持续盯着。文档里反复出现的提醒，比如电脑要保持开机运行、网络要正常、连接器要定期检查状态、自动化要在工作空间里管，其实都在说同一件事：这套系统能跑起来，但需要你维护它的运行环境。能力越全，维护面越宽。真正变重的成本，不在学功能，而在持续保证这一切还在正确状态里。

93. 模型越来越多时，你该怎么避免把“换模型”当成唯一解法？

换模型确实是个常用手段，文档里遇到读不出文件、任务卡住、输出异常，第一步建议往往是先切换模型测试。但如果每次出问题都只靠换模型，你会漏掉真正该修的东西。有些问题换模型能缓解，比如某个模型不支持某种文件类型；有些问题换模型治不了，比如任务本身太宽、目录没设对、需求里有它理解不了的模

糊点，后者换多少个模型都还是会卡在同一处。更稳的习惯是：遇到问题先切换模型试一次，没解决就回头查流程和需求。把错误码归类、把卡住的步骤定位出来、把模糊的需求写清楚，这些动作比无脑换模型更有用。模型是工具变量，不是万能开关。

94. 技能、MCP、连接器都在扩张时，普通用户最该先守住哪条授权边界？

最该守住“它能对外做什么”这条线。技能更多影响内部执行方式，关掉顶多是某个能力不调用，影响有限。MCP 和连接器不一样，它们直接决定 WorkBuddy 的手能伸到哪些外部系统——能不能读你的邮件、写你的网盘、发消息到工作群、动你的在线文档。这些动作一旦执行，后果是向外的、难撤销的。文档把权限分成默认和完全访问两档，默认权限下敏感操作和工作空间外的执行都需要你确认，这个设置就是给普通用户兜底用的。扩张能力时先把这条边界守住：连接器只授权当前任务需要的范围，用完就关，定期清点还连着哪些，外发类动作一律留人工确认。技能可以多装几个试，MCP 和连接器要谨慎接。内部可以折腾，对外要克制。

95. 为什么真正拉开差距的，常常是任务结构和验收方式，而不是提示词？

因为提示词再精巧，也只是单次任务的输入；任务结构和验收方式决定的是你能不能稳定复用、能不能持续拿到靠谱结果。很多人把精力花在琢磨一句话怎么写得更好，却没想清楚这个任务到底分几步、每步的输入输出是什么、最后拿什么标准验收，结果是偶尔写得好，换个材料就飘。真正拉开差距的人，会在意更深的两层：任务结构清不清楚，能不能拆成可验收的环节；验收方式硬不硬，是看文件本身还是看它说“已完成”。文档里反复强调的“可验收结果”“先验证再扩大”“结果区四类视图核对”，说的都是验收这一层。提示词是术，任务结构和验收是道。把后两层想清楚，提示词稍微朴素一点，结果反而更稳。

96. 什么情况下，WorkBuddy 会从一次性尝鲜，变成你固定会开的工作台？

发生在你跨过那道坎之后：找到一个高频复用、已经跑顺、下次还能原样再来的场景。一次性尝鲜的状态是，你偶尔想起它，扔个任务试试，做完就关掉，下次想用还得重新组织。固定会开的状态是，你开工前会顺手打开它，因为你知道有几个活交给它更省事，而且这些活的流程已经稳了。文档描述的正是这个转变—

-从偶尔用一下，到每天会开，中间卡住的就是有没有一个稳定复用的场景。这个场景通常有几个特征：频率高、规则固定、交付物能验收、你已经验证过不止一次。尝鲜靠新鲜感，工作台靠真实省下的时间。后者才是它能长期留在你日常工作里的理由。

97. 你到底该不该把对外发送、正式发布、直接改文件这类动作交给它？

可以交给它准备，但最后那一下最好还是人来过。这类动作的共同点是后果向外、难以撤销。邮件发错对象、文档发布了没清理的内容、直接覆盖了重要原文件，这些麻烦都不是内部返工能解决的。它很适合帮你把前期工作做完一起草邮件、整理文档、准备好要改的版本。但到了真正发送、发布、覆盖的那一刻，多一道人工确认成本很低，省下的风险很高。文档在讲连接器和助理时也暗含这个意思：默认权限下敏感操作要确认，授权可以随时关掉。把高后果动作和高效率流程分开，前面让它跑，效率归它；最后一下你点头，责任归你。

98. 为什么功能看起来很多，不代表你一开始就该全都用上？

功能多只是可能性多，不代表每个都适合你当下的场景。一上来就把模型、专家、技能、连接器、自动化、远程助理全铺开，结果往往是哪个都没用透，还增加互相干扰的概率。文档专门提醒过，只启用当前任务所需的技能，能减少无关干扰和误调用，这个原则放到整个功能面上同样成立。更稳的节奏是从一个具体场景入手，把它跑顺、跑稳、跑成习惯，再考虑加下一个能力，每加一层都验证它真的让你更省事，而不是更忙。功能是为你服务的，不是用来集邮的。用透三个功能，比浅尝十个更能让你建立对它的真实判断。少而准，永远比全而杂更有力量。

99. 如果只打算先学一部分，你最该先守住哪些基础能力？

最该守住三样。第一是会提需求，能把做什么、有什么材料、要什么结果说清楚，这是所有后续能力的基础。第二是会用结果区验收，知道打开产物、全部文件、变更、预览这四个视图去核对它到底交付了什么，而不是只看对话里那句“已完成”。第三是会控制边界，知道给它专门目录而不是整个桌面，知道敏感操作留人工确认，知道高风险动作先预览再执行。这三样凑在一起，你就能稳稳地用它处理真实工作。模型怎么选、专家怎么拉、连接器怎么接、自动化怎么搭，这些都是进阶，可以后面慢慢补。基础这三样不守住，进阶功能用得再花，结果也飘。

100. 这 100 个问题，最后到底该帮你避开什么误用和误判？

帮你避开三类最伤的误判。第一类是把它当聊天框用，只问不交，永远停在它最浅那层能力上，然后得出“也就那样”的结论。第二类是把它当万能替身用，一上来就全权放手，让它自动发送、覆盖原文件、整理整个桌面，结果在错误前提下非常认真地执行，闯了不该闯的祸。第三类是频繁换底座当唯一解法，模型一换、技能一加、连接器一接，却不回头查流程和验收，于是同样的坑反复踩。这 100 个问题反复说的其实是同一件事：用它的关键不在功能多，而在任务结构清楚、边界控得住、验收够硬。避开这三类误判，你就不会在它真正能帮上忙的场景里错过它，也不会在它不该放手的地方盲目信任它。工具的价值，最终取决于用它的人知不知道它的边界在哪。

七、参考资料

一、WorkBuddy 官方文档

<https://www.workbuddy.ai/docs/workbuddy>

二、腾讯云开发者社区 (cloud.tencent.com)

- 腾讯云·WorkBuddy 深度解析

<https://cloud.tencent.com/developer/article/2680651>

- 腾讯云·入门与高阶玩法

<https://cloud.tencent.com/developer/article/2699138>

- 腾讯云·从认识到上手

<https://cloud.tencent.com/developer/article/2668741>

- 腾讯云·核心机制拆解

<https://cloud.tencent.com/developer/article/2647226>

- 腾讯云·定位与上手建议

<https://cloud.tencent.com/developer/article/2669663>

- 腾讯云·场景与落地案例

<https://cloud.tencent.com/developer/article/2638618>

三、Tencent Cloud 英文技术站

([tencentcloud.com](https://www.tencentcloud.com))

- Tencent Cloud·WorkBuddy 英文产品介绍

<https://www.tencentcloud.com/techpedia/144114?lang=en>

- Tencent Cloud·WorkBuddy 中文能力解读

<https://www.tencentcloud.com/techpedia/144100?lang=zh>

四、社区评测与集成文档

- Eigent·WorkBuddy 第三方评测

<https://www.eigent.ai/blog/workbuddy-ai-review>

- CloudBase·WorkBuddy 接入文档

<https://docs.cloudbase.net/en/ai/ai-tools/workbuddy>

- Evolink·与 CodeBuddy 集成指南

<https://docs.evolink.ai/cn/integration-guide/codebuddy-workbuddy>

- 云巴巴·WorkBuddy 问答 <https://www.yun88.com/qa/3853>

- xmsumi·WorkBuddy 介绍 <https://www.xmsumi.com/detail/3341>