One Person Lab

OPL Framework 白皮书

为长期运行的 AI 专业工作提供可组合、可验证、可演进的底座

优秀的智能体底座应让每项能力都能被独立理解、验证、替换和组合，同时让事实与责任始终清楚可见。

Composable Capabilities / Reliable Work / Continuous Evolution

2026-08-15

Public whitepaper

目录

发布日期：2026-08-15

最近修订：2026-08-16

适用对象：希望理解 OPL Framework 为什么这样设计、它如何支撑可靠的 AI 专业工作，以及这套架构为什么能够长期演进的用户、合作者和技术决策者。

核心判断：优秀的智能体底座应让每项能力都能被独立理解、验证、替换和组合，同时让事实与责任始终清楚可见。

1 从一次运行到长期连续工作

让一个智能体完成一次任务并不困难。真正困难的是让一组不断变化的模型、工具、专业能力和用户界面，在数周甚至数月的工作中仍然保持可理解、可恢复和可追责。

复杂知识工作会不断遇到变化：更好的模型出现了，新的资料源需要接入，执行方式需要替换，评估方法需要升级，界面也会迭代。如果这些能力都依赖同一套隐式启动顺序、全局状态和手工连接，任何一次升级都可能牵动整条运行链。系统越有能力，开发、测试和运维反而越困难。

OPL Framework 关注的是专业智能体如何持续变好：每一部分都能独立演进，整个系统仍然保持可信。

因此，Framework 把运行底座理解为一个可组合的能力系统。规则、发现、工作空间、阶段、执行、证据、呈现、评估和连接，都以清晰的能力域和责任面参与同一条专业工作线。它们彼此协作，也各自守护事实与判断。

2 为什么让能力动态装配

OPL Framework 不把所有能力固化在同一条运行链里，而是让每项能力通过明确关系进入组合，在清楚的生命周期中启动和退出，并能被检查、替换和重新组合。

这带来五个直接结果。

第一，**改变一个能力，不必先理解整个系统**。每项能力只需要理解自己的责任和明确依赖，升级执行、发现或观测能力时，改动可以停留在合理边界内。

第二，**测试可以聚焦到单项能力**。每项能力可以独立装载、进入受控环境、面对缺失依赖并证明退出后没有遗留影响；完整组合再验证各项能力是否协作正确。

第三，**故障可以被解释**。系统能够回答实际装载了什么、哪项能力由谁提供、哪个版本参与了本次工作，以及不兼容发生在哪里。

第四，**升级和回退有了稳定对象**。一次工作可以绑定到明确的能力组合。新能力可以先在受控组合中验证，出现问题时也能回到上一组已知配置，而不必依赖难以复现的全局状态。

第五，**持续改进有了真实的操作单位**。候选生成、独立评估、受控替换和效果比较可以围绕一项能力进行，不必每次修改整个智能体运行底座。

运行底座提供组合秩序，专业智能体与负责人提供领域判断。运行层记录能力如何协作，领域层决定成果是否成立；这种分工让两种专业性各得其所。

3 四层生态中的可组合底座

用户面对的是 OPL Base、OPL App、OPL Packages 和 OPL Cloud。Framework 是 Base 的核心，负责可靠推进工作；Packages 提供可以安装和升级的专业能力；App 提供本地工作台；Cloud 将同一条 工作线延伸到在线空间、托管资源与协作服务。四层产品各有分工，用户始终使用同一种任务、成果、证据和责任语言。

从建立工作边界到持续运营，十一项品牌能力共同支撑一项专业工作的完整旅程。

工作旅程	品牌	为用户解决的问题
准备	**OPL Charter**	明确目标、规则、权限和不可逾越的边界
准备	**OPL Workspace**	让材料、过程和成果始终处在正确位置
组建	**OPL Atlas**	找到当前工作真正可用的智能体与能力

工作旅程	品牌	为用户解决的问题
组建	**OPL Pack**	让专业能力可以安装、升级和复用
组建	**OPL Stagecraft**	把复杂目标组织成可推进、可交接的专业阶段
推进	**OPL Runway**	让工作能够启动、持续、恢复并完成收口
推进	**OPL Ledger**	保存进展、证据和关键决定，支持复查与追溯
推进	**OPL Connect**	接入完成任务所需的数据、工具和外部服务
运营	**OPL Console**	看清当前状态、问题归属和下一步行动
运营	**OPL Foundry**	用真实效果持续评估和改进专业智能体
运营	**OPL Fabric**	为在线工作供给并治理计算、存储与托管资源

这些能力不是十一套彼此割裂的产品，而是一条连续工作链。用户从目标出发，OPL 负责在合适的位置 提供所需能力；无论工作发生在本机还是云端，任务、文件、进度、证据和下一步都保持连贯。

4 能力独立升级，体验始终一致

动态装配的价值，不是让用户管理更多零件，而是让系统可以局部升级、准确定位问题并可靠回退。OPL Packages 让专业能力按需要安装和更新，运行底座让各项能力在清楚的生命周期中协作，经过验证的 产品组合则把这些内部选择收敛为简单、稳定的使用体验。

当执行、连接、观察或评估能力获得更好的实现时，OPL 可以只替换相应部分。用户已有的工作位置、阶段、文件、证据和操作方式继续有效；出现问题时，系统也能说明发生变化的是哪项能力，并回到上一 个可靠组合。内部升级的自由，因此不会转化为用户的学习和维护负担。

5 受控组合，保持体验清楚

OPL Framework 在内部保留能力组合的自由，在产品中提供少量经过完整验证的组合形态。

基础运行、完整桌面工作台、研究工作、基金工作、视觉交付和 Foundry 开发，需要的能力组合各有侧重。Framework 用受控配置表达这些差异：每种配置都选择明确的能力，冻结实际组合，并接受完整验证。

用户只需选择“我要完成什么工作”。稳定的能力域降低维护成本，受控配置降低使用成本；二者结合，让系统内部快速演进，产品体验仍然稳定、清楚、可预测。

6 Framework 与 App：运行能力和产品体验各司其职

OPL Framework 负责组织和运行能力，确保任务可以可靠启动、持续、恢复和收口。One Person Lab App 负责把这些能力变成连贯的窗口、导航、任务视图、文件工作面、进度和操作入口。

两者采用一致的可组合设计，但承担不同责任：Framework 保证能力真实可用，App 保证用户看到的是 一套稳定产品。运行底座和界面因此可以独立演进，同时共享同一份任务、状态和操作事实。

7 界面持续演进，产品体验保持连续

One Person Lab App 对外提供一套统一的任务、能力、文件、状态和操作体验。界面设计可以持续改进，但用户不会因此进入两套产品，也不需要重新建立工作、迁移能力或学习相互冲突的操作方式。

只有经过完整兼容性、安装、安全和发布验证的界面才会进入正式产品。内部实现如何演进，不改变用户 已有工作的连续性，也不改变产品对状态、操作和结果的承诺。

8 自进化：围绕能力持续改进

这套底座的长期价值，在于让系统能够观察自身、提出改进并在证据约束下采用更好的实现。

当执行、发现、观测、评估和连接都可以独立升级时，Foundry 就能围绕一项清楚的能力工作：保留当前版本，形成候选，执行针对性测试，在真实工作中比较效果，再由明确的负责人决定是否采用。成熟的改进只替换相应部分，其他能力保持稳定。

每次实验都保留完整的版本和组合记录，Ledger 保存证据，Foundry 提供独立比较，最终采用决定归属明确。自进化由此获得可以验证、可以回退、可以持续积累的工程基础。

自进化因此成为一条专业的改进链：提出候选、获得证据、接受评估、受控激活、保留回退。

9 可组合的能力，清楚的责任

可组合层负责运行能力之间的协作，各类事实则由真正的拥有者维护。

领域智能体决定专业成果是否合格；Workspace 和文件系统保存实际材料与产物；持久运行环境保存长期执行历史；Ledger 保存证据与事件；App 持有桌面产品和发布事实；能力包负责人维护来源、版本与能力声明。Framework 把这些事实连接起来，让每项判断都能回到正确的责任方。

这种分工让新的提供者、连接器、观察器或执行器能够进入组合，同时保持事实归属稳定。创新可以发生在局部，信任则贯穿整个系统。

10 一次升级如何发生

假设任务恢复能力出现了一个表现更好的候选版本。

在传统整体式运行器中，这次升级可能同时触碰启动逻辑、状态页、工具注册、任务恢复和发布脚本。团队需要重新验证整个系统，而且很难判断改进究竟来自哪里。

在 OPL Framework 中，候选版本先进入受控组合，沿用相同的工作阶段、文件位置和证据要求。Foundry 比较恢复成功率、证据完整性和用户可见结果。只有证据支持且对应负责人确认后，新组合才会用于后续工作。

升级前后的工作都能回答：使用了哪一版能力，组合中还有哪些能力，评估依据是什么，出现问题时回到哪里。一次局部改进因此不会变成一次全系统赌博。

11 这套设计带来的可验证预期

这套设计是否有效，应由用户可感知的运行结果和可以复查的证据判断，而不是由内部结构或技术名称判断。

- **品牌与责任一致。**每项用户可见能力都有清楚的产品承诺和负责人，不用概念掩盖实际边界。
- **组合可以被看见。**每次运行都能解释实际使用了哪些能力及其来源。
- **升级可以被比较。**新能力先获得独立证据，再进入受控组合。
- **故障可以被隔离。**能力生命周期和责任边界让问题停留在合理范围，并保留清楚的回退对象。
- **界面演进保持产品连续。**正式产品始终使用同一套任务、文件、状态和操作来源，并接受一致的产品与发布验证。
- **责任始终归属明确。** Framework 负责组合与运行，领域质量、文件、发布和最终决定归对应责任方。

这套设计让 OPL 可以同时获得两种通常难以兼得的能力：内部持续变化，外部长期稳定。

12 结语

OPL Framework 的目标，是成为一套结构清楚、持续进化的专业能力系统。

四层生态让用户理解 Base、App、Packages 和 Cloud 的分工；十一项品牌能力让一项工作从准备、组建、推进到运营都有清楚去向；动态装配让运行能力可以组合、检查和替换；经过验证的产品组合让内部自由不会成为用户负担；Framework 与 App 让运行和界面共享同一种设计语言；Foundry 则让每项能力可以在证据约束下持续改进。

这些设计最终应体现为更稳定的工作、更清楚的进展、更容易恢复的任务，以及一个能够不断升级却不丢失责任与事实的 AI 专业平台。

了解更多：

- One Person Lab 家族白皮书
- One Person Lab
- One Person Lab App
- OPL Cloud