SERIES 03 / PALANTIR DEEP DIVE

3000亿美金的企业AI操作系统

Palantir Foundry × AIP

—引擎盖下的真实架构

承接 Series 01 本体论基础 / Series 02 语义栈三层扩展

$300B+	20+	3
市值 (2026.07)	年数据积累	核心产品层

Palantir 不是一家 AI 公司。它卖的不是模型，不是 API -- 是一整套让 AI 在企业里安全落地的操作系统。

这期拆的是：这个操作系统到底长什么样，里面每一层在干什么。

PRODUCT ARCHITECTURE

产品结构 -- 三层关系

Palantir 的产品不是一个工具，而是三层堆叠的系统。点击每层查看详情。

AIP AI Platform

-- AI 层，依附 Foundry ▼

AIP 不是一个独立产品，它是 Foundry 的 AI 扩展层。所有 AI 能力（LLM 调用、Agent 编排、Chatbot）都跑在 Foundry 的数据基础之上。

AIP 的核心理念：LLM 永远不直接碰你的数据和工具。它通过 Ontology 定义的 Action Type 来操作业务，每一步都可审计、可回滚、可限权。

关键组件：`AIP Logic / AIP Chatbot / AIP Assist / Agent Studio`

FOUNDRY Data Operating System

-- 数据操作系统 + Ontology ▼

Foundry 是 Palantir 的核心。把它理解为"企业级数据操作系统" -- 数据接入、清洗、建模、Ontology 构建、权限管理、应用开发，全在这一层完成。

Ontology 是 Foundry 最重要的概念。它不是数据库，是一层覆盖在所有数据源之上的"业务语义层"。每个 Object Type 都有属性、关系、动作的严格定义。

关键组件：`Data Connection / Pipeline Builder / Ontology Manager / Workshop / Quiver`

APOLLO Deployment Platform

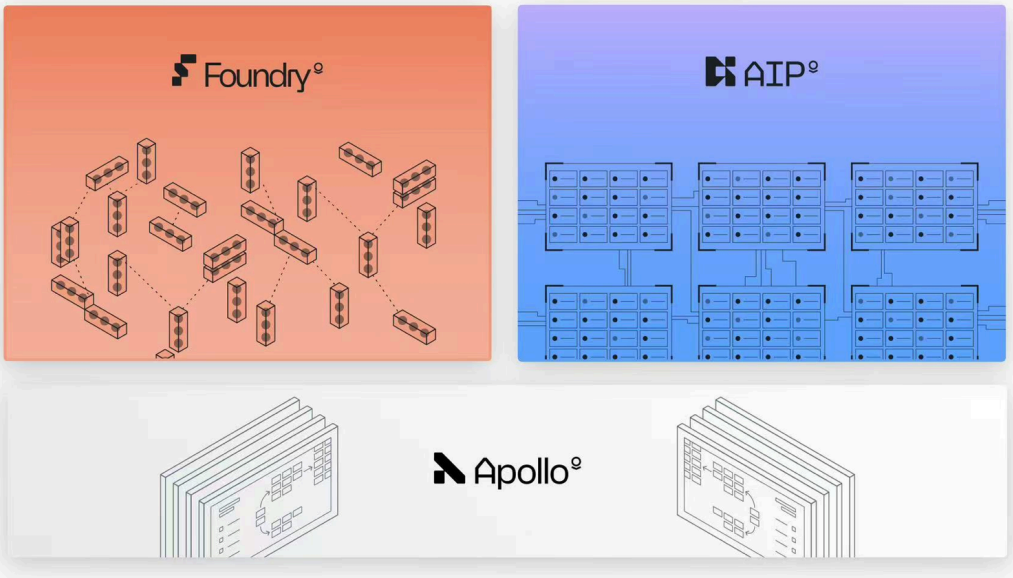
-- CI/CD，支持气隙网络 ▼

Apollo 是基础设施层。负责 Palantir 所有软件的持续部署和生命周期管理。关键能力：支持在完全离线（air-gapped）的网络环境下运行。

这是 Palantir 能做国防、情报客户的技术前提。Apollo 可以把整个 Foundry + AIP 部署到没有互联网的保密网络中，同时保持软件自动更新和版本管理。

关键能力：`Air-gapped deployment / Continuous delivery / Multi-cloud / On-prem`

AIP, Foundry, and Apollo The Enterprise Operating System



Palantir

palantir.com / official architecture

> AIP sits on top of Foundry, which sits on top of Apollo. You cannot have AIP without Foundry. You cannot have Foundry without Apollo. This is a full operating system, not a feature.
>
> -- Palantir internal engineering documentation

ONTOLOGY CONSTRUCTION

Ontology 怎么建

四步向导。点击每个步骤查看操作细节。

1 选数据源　2 填元数据　3 映射属性　4 设主键

在 Foundry 的 Ontology Manager 中，选择底层数据集作为 Object Type 的数据来源。支持 CSV、Parquet、数据库直连、API 接入等多种格式。一个 Object Type 可以关联多个数据集 -- 物理存储分散，逻辑上统一。

为 Object Type 填写名称、描述、所属领域（Domain）。这一步定义的是"业务含义" -- 这个对象在组织里代表什么。好的元数据是可搜索、可治理的前提。Foundry 会强制要求这些字段，不是可选的装饰。

把底层数据集的列（Column）映射为 Object Type 的属性（Property）。例如：数据表的 `cust_name` 映射为 `Customer.name`。映射过程中可以做类型转换、字段重命名、计算派生属性。

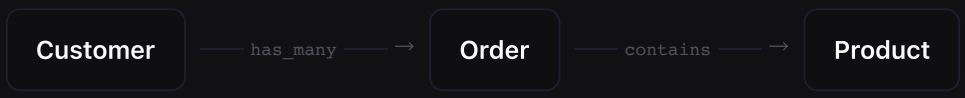
为 Object Type 设置主键（Primary Key）。主键是唯一标识每个 Object 实例的字段。系统会校验唯一性。主键设好后，这个 Object Type 就可以被其他 Object Type 引用，建立 Link Type（关联关系）。

Object Type 结构

OBJECT TYPE **Customer**

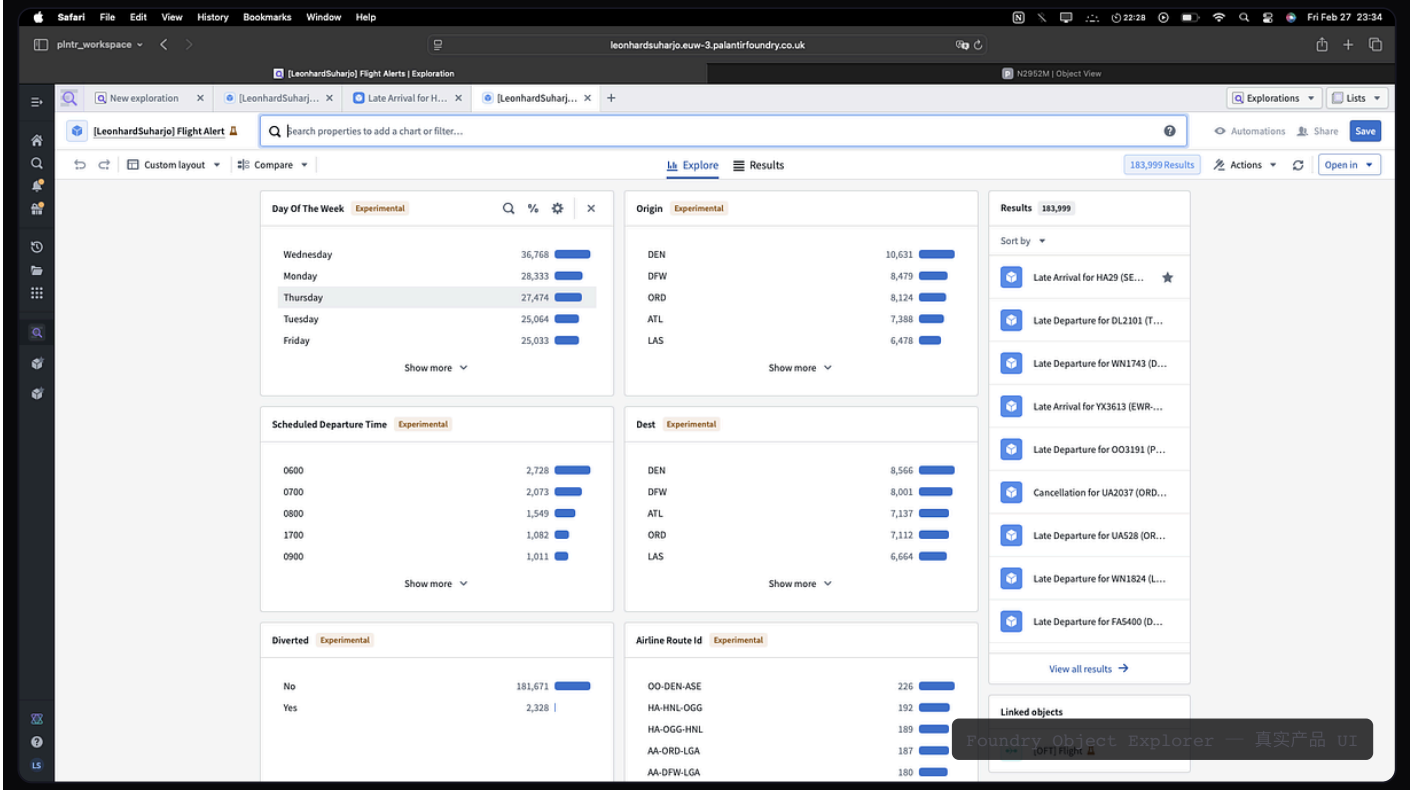
customer_id	string	PK
name	string	
region	string	
tier	enum [Gold, Silver, Bronze]	
orders	Order[]	LINK
upgrade_tier	Action	ACTION

Link Type 示例



传统数据表 vs Ontology Object Type

维度	数据表	OBJECT TYPE
语义	列名是代号，含义靠猜	每个属性有业务定义
关系	JOIN 写在 SQL 里，隐式	Link Type 显式声明
操作	任何 SQL 都能跑	只有 Action Type 定义的操作
权限	表级或行级	对象级 + 属性级 + 操作级
版本	靠 migration 脚本	内置版本控制和审计



Foundry 中浏览 Ontology 对象的真实界面：左侧维度过滤、中间数据分布、右侧对象详情和 Linked Objects

CORE INNOVATION

Action Type -- 最关键的创新

Action Type 是 Palantir 安全模型的基石。LLM 不能"自由发挥"，只能调用预定义好的操作。

配置流程

选操作类型

创建、修改、删除、还是自定义逻辑？每种操作类型对应不同的安全等级和审批流。创建类操作通常权限最宽，删除类最严。

定义参数

每个 Action 的输入参数都需要预定义。类型、是否必填、取值范围。例如"修改客户等级"的参数就只有 `target_tier: enum`，不能传任意值。

设前置条件

Precondition -- 什么情况下这个操作可以执行。例如：只有当客户近 12 个月消费超过 10 万，才允许升级到 Gold Tier。条件不满足，操作直接被拦截。

配权限

谁能触发这个 Action？人类用户、特定 Agent、还是自动化流程？权限可以精细到：只有华北区经理能批准华北客户的等级升级。Role-Based + Object-Scoped。

加 Side Effects

操作执行后触发的联动效果。升级客户等级后自动：发通知邮件、更新 CRM 记录、触发下游工作流。Side Effects 也是预定义的，不是 LLM 即兴发挥。

安全模型对比

传统模式

- AI 自由操作
- 操作后审核
- 出事了再回滚
- 先进门，后装摄像头

Palantir 模式

- 只有预定义的门存在
- 其他地方全是墙
- 每扇门都有锁和日志
- 建筑设计阶段就定好通道

> "LLMs do not have direct access to tools or data. They can only invoke pre-defined Action Types on the Ontology."
>
> -- Palantir AIP documentation, learn.palantir.com

OAG VS RAG

Ontology-Augmented Generation

Palantir 没有选择 RAG，而是发明了自己的范式：OAG。两者的差异不只是技术路径，而是哲学。

RAG

▶ 文档

↓ 切块 (Chunking)

↓ 向量搜索

↓ 拼给模型

↓ 生成回答

非结构化 / 无权限 / 难审计

OAG

▶ Ontology

↓ 定位业务对象

↓ 带定义 / 口径 / 权限

↓ 结构化上下文

↓ 精确回答 + 可审计

结构化 / 精确 / 可审计

Q Q3 华北家电渠道毛利率是多少?

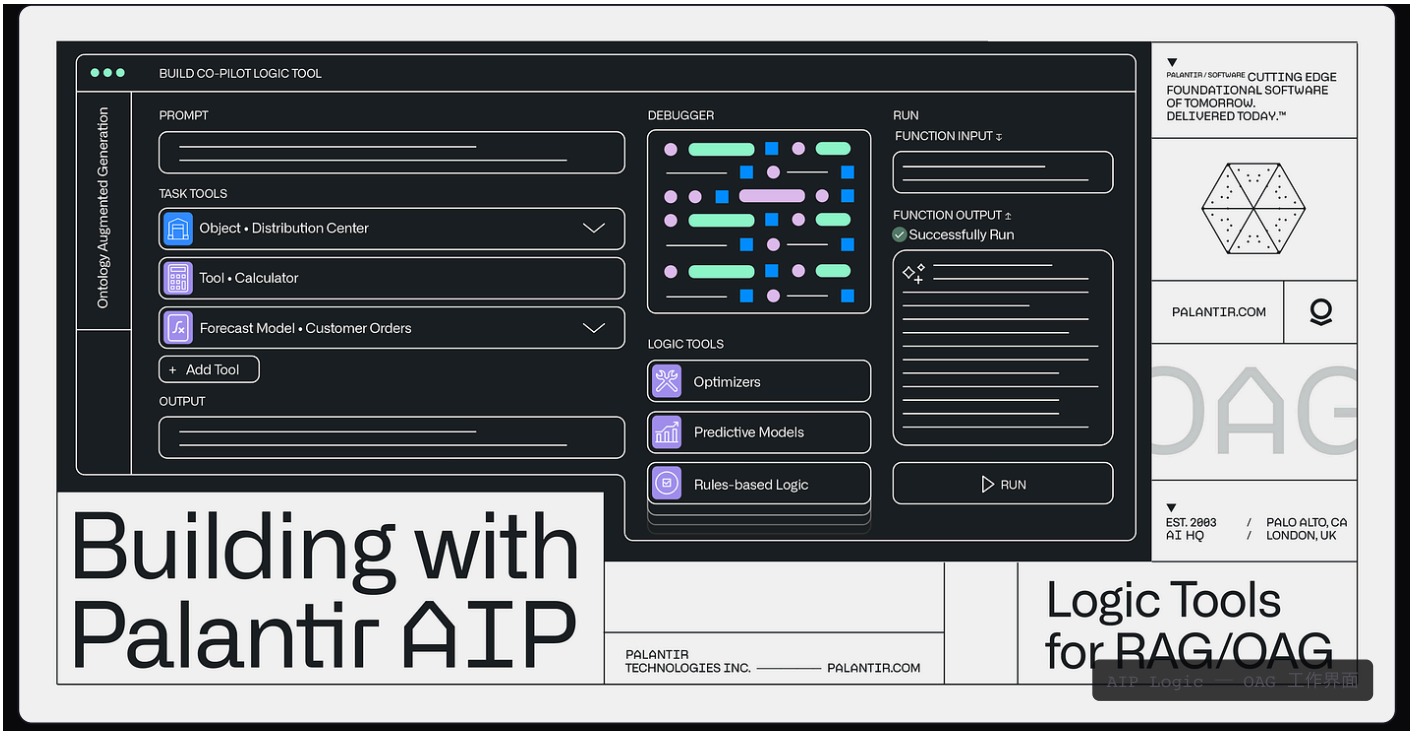
RAG 回答

- 搜到三份包含"毛利率"的文档
- 两份口径不同（含/不含退货）
- 取了置信度最高的 chunk
- 结果可能是过期数据

OAG 回答

- 定位 Object: 华北.家电.Q3
- 口径：含退货、含促销折让
- 数据源：ERP + CRM 实时
- 42.3%，↓3.1pp YoY

OAG 的隐藏能力：不只查数据，还能调确定性工具。Ontology 里的 Action Type 可以触发优化器、计算器、审批流 -- 这是 RAG 做不到的。



AIP Logic 的 Co-Pilot 工具配置界面：左侧 Prompt + Task Tools（Object、Calculator、Forecast Model），中间 Debugger，右侧 Run 面板。Logic Tools 包括 Optimizers、Predictive Models、Rules-based Logic

APPLICATION LAYER

Workshop + AIP Chatbot

Ontology 建好了，谁来消费它？Workshop 是操作界面，Chatbot 是自然语言入口。

Workshop -- Ontology 的消费层

Workshop	拖拽式应用构建器，消费 Ontology 的 Object/Action/Link	↑
Ontology	Object Types + Link Types + Action Types -- 统一语义层	↑
Data	多数据源接入 -- ERP / CRM / IoT / 第三方 API	

Workshop 不写代码。业务人员可以直接拖 Object Type 的属性到仪表盘，绑定 Action Type 到按钮。Ontology 变了，Workshop 应用自动更新 -- 因为引用的是语义层，不是底层表。

AIP Chatbot Studio -- 三要素配置

C

Context Sources

选哪些 Object Type 作为上下文。LLM 只能"看到"你选定的对象，看不到的就是不存在。

T

Tools

绑定哪些 Action Type 给 Chatbot 用。可调用的操作 = 可开的门。没绑定的 Action，Chatbot 无法触发。

M

Model

选用哪个 LLM 后端。Palantir 不绑模型 -- GPT-4o、Claude、Llama 都行。模型是可替换的零件。

> AIP Chatbot 是一扇透明的窗户，不是一道打开的门。用户能看到 Ontology 的信息，能通过 Action Type 做操作 -- 但看到什么、能操作什么，全由配置决定，不由模型决定。
>
> -- "透明窗户"比喻，来自 Ontologize YouTube 系列

FORMER FDE SPEAKS

前 FDE 的三个大实话

FDE = Forward Deployed Engineer，Palantir 派驻客户现场的工程师。点击展开真相。

01 极其昂贵，很多项目失败

Palantir 的合同动辄千万美金起。但高价并不保证成功。很多客户买了 Foundry 之后发现：自己的数据治理根本没到能用 Ontology 的程度。数据质量不过关，什么操作系统都是空中楼阁。

FDE 的存在就是为了弥补这个 gap -- 但这也意味着 Palantir 的交付极度依赖人，不是纯产品驱动。

02 大量重复浪费

每个客户的 Ontology 都是从零建的。医药行业 A 公司建了一套药品 Object Type，B 公司到了再建一遍。行业知识没有跨客户复用的机制。

这是 Palantir 商业模式的内在矛盾：定制化带来高客单价，但也让边际成本降不下来。"平台"的故事在收入端还没兑现。

03 产品是踩坑提炼出来的

Foundry 的很多最佳实践不是产品团队设计出来的，是 FDE 在客户现场踩了无数坑之后抽象出来的。Action Type 的安全模型、Ontology 的版本控制、Workshop 的构建范式 -- 全是血泪经验。

这也是为什么 Palantir 的护城河不只是代码，而是 20 年积累的"企业数据怎么治理"的 know-how。

> **大模型公司组建 FDE 团队深入企业一线，表面帮客户找场景，实际是找高消耗场景来锁死迁移成本。服务行业必须 vendor agnostic。大模型厂商搞咨询，就像药厂自己组建医生团队 -- 医生卖药，商业伦理天然矛盾。**
>
> -- Alex Karp 对大模型公司 FDE 模式的批评（公开演讲，多次重复）

DISCLOSURE —— 利益冲突披露：本期内容基于公开资料和前 FDE 分享，作者与 Palantir 无利益关系。批判和赞赏并存，不是推销。

COGNITIVE UPGRADE

认知升级 + Takeaways

三期系列进化线

Series 01

本体 = 知识规范

从亚里士多德到知识图谱，本体论是什么、为什么重要

Series 02

加三层

语义栈 = 本体 + 指标 + 事件 + 治理。Data Agent 的完整架构

Series 03

终态是融合

Palantir 用 20 年踩坑证明：分层建设，融合消费，安全兜底

四个可执行 Takeaway

01

先治数据，再谈 Ontology

Palantir 项目失败率高的核心原因：客户数据质量不到位。在考虑任何语义层之前，先确保你的核心业务数据是干净的、有唯一主键的、有明确 owner 的。

02

Action Type 思维比 Ontology 更紧迫

大多数企业在 AI 落地时最缺的不是知识图谱，而是"AI 能干什么"的边界定义。先定义 10 个最高频的业务操作，把参数、前置条件、权限想清楚 -- 这比建 Ontology 更能立竿见影。

03

OAG > RAG，前提是你有结构化语义

如果你的业务知识已经结构化了（哪怕是 Excel + 数据字典），就不要走 RAG 的弯路。把它建成 Object Type，带上口径和权限，效果远超向量搜索 + chunk 拼接。

04

模型可换，语义层不可换

Palantir 不绑模型，GPT / Claude / Llama 随时切换。这说明：模型是消耗品，语义层才是资产。把精力花在定义你的业务 Ontology 上，而不是纠结用哪个模型。

分层是起步，融合是终态。

Data + Ontology + AI -- 分开建设，融合消费，安全兜底。这是 Palantir 用 3000 亿美金验证的路径。

Sources: learn.palantir.com / Ontologize YouTube / LearnWithBeibei

by Tricia