

Asset Servant 儲存職責澄清

作者: Manus AI

日期: 2025年10月14日

版本: 1.0

核心原則

Asset Servant 不儲存任何資料






Asset Servant 是一個無狀態的映射服務,它不儲存: - ❌ USD 3D Model - ❌ 時序數據 - ❌ 資產屬性 - ❌ 文檔檔案 - ❌ 任何持久化數據

Asset Servant 只儲存: - ✅ Tag 映射表 (從 IADL 解析而來,運行時記憶體中)

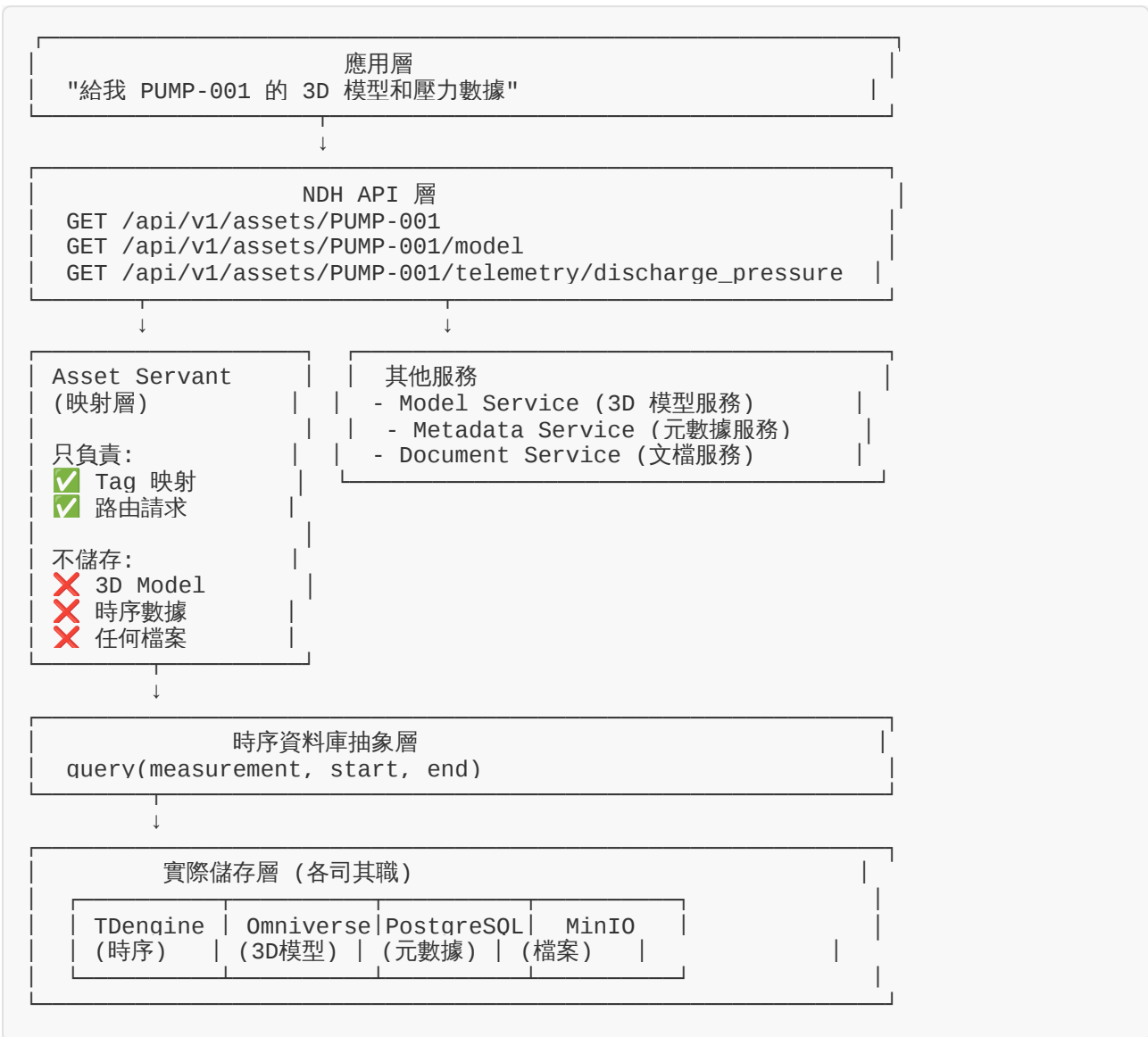
完整的資料儲存架構

1. 各類資料的實際儲存位置

資料類型與儲存位置

-  IADL 定義檔案 (資產藍圖)
 - └ 儲存位置: Git Repository / 檔案系統
 - └ 格式: YAML/JSON
-  USD 3D Model (幾何模型)
 - └ 儲存位置: Omniverse Nucleus / MinIO / S3
 - └ 格式: .usd, .usda, .usdc
-  時序數據 (Tag 數據)
 - └ 儲存位置: TDengine / PI System / InfluxDB
 - └ 格式: 時序資料庫原生格式
-  資產元數據 (屬性、配置)
 - └ 儲存位置: PostgreSQL (關係型資料庫)
 - └ 格式: 表格結構
-  文檔檔案 (PDF, 圖紙, 照片)
 - └ 儲存位置: MinIO / S3 / 分散式檔案系統
 - └ 格式: 原始檔案格式

2. Asset Servant 在架構中的位置



詳細說明

IADL 定義 vs. Asset Servant 儲存

IADL 檔案內容:

```

# pump001.iadl.yaml (儲存在 Git / 檔案系統)

asset:
  asset_id: PUMP-001

  # 3D 模型引用 (不是實際模型, 只是路徑)
  geometry:
    model_file: "omniverse://server/projects/plant1/models/pump001.usd"
    # ↑ 這只是一個 URL 引用, 實際 USD 檔案在 Omniverse Nucleus

  # 數據標籤定義
  data_tags:
    - tag_id: discharge_pressure
      source:
        type: tdengine
        measurement: pump001_pressure
      # ↑ 這只是映射配置, 實際數據在 TDengine

  # 靜態屬性
  properties:
    manufacturer: "Grundfos"
    rated_power: 15
    # ↑ 這些屬性會被載入到 PostgreSQL

  # 文檔引用
  documents:
    - type: manual
      file_path: "s3://bucket/docs/pump001_manual.pdf"
      # ↑ 這只是一個 S3 路徑, 實際 PDF 在 MinIO/S3

```

Asset Servant 在記憶體中的狀態:

```

class AssetServant:
    def __init__(self, asset_id: str, iadl_definition: dict):
        self.asset_id = "PUMP-001"

        # 只儲存 Tag 映射表 (運行時記憶體)
        self.tag_mappings = {
            "discharge pressure": {
                "db_type": "tdengine",
                "measurement": "pump001_pressure"
            }
        }

        # 不儲存 3D 模型
        # 不儲存時序數據
        # 不儲存文檔
        # 不儲存任何檔案

```

各類資料的訪問流程

場景 1: 獲取 3D 模型

應用層請求:

GET /api/v1/assets/PUMP-001/model

NDH API 層:

↓ (路由到 Model Service, 不經過 Asset Servant)

Model Service:

1. 查詢 PostgreSQL 獲取資產元數據
2. 從元數據中讀取 model_file 路徑
3. 從 Omniverse Nucleus 獲取 USD 檔案
4. 返回 USD 模型 URL 或串流

Asset Servant: 完全不參與此流程 ❌

場景 2: 獲取時序數據

應用層請求:

GET /api/v1/assets/PUMP-001/telemetry/discharge_pressure

NDH API 層:

↓ (路由到 Asset Servant)

Asset Servant:

1. 查找 Tag 映射: "discharge_pressure" -> TDengine.pump001_pressure
 2. 調用 TSDB 抽象層
- ↓

TSDB 抽象層:

↓ (路由到 TDengine Adapter)

TDengine:

返回實際時序數據

Asset Servant: 只負責映射和路由 ✅

Asset Servant: 不儲存時序數據 ❌

場景 3: 獲取資產元數據

應用層請求:

GET /api/v1/assets/PUMP-001

NDH API 層:

↓ (路由到 Metadata Service)

Metadata Service:

1. 查詢 PostgreSQL
2. 返回資產屬性 (manufacturer, rated_power, etc.)

Asset Servant: 完全不參與此流程 ❌

場景 4: 獲取文檔

應用層請求:

GET /api/v1/assets/PUMP-001/documents

NDH API 層:

↓ (路由到 Document Service)

Document Service:

1. 查詢 PostgreSQL 獲取文檔清單
2. 從 MinIO/S3 獲取實際檔案
3. 返回下載 URL 或串流

Asset Servant: 完全不參與此流程 ❌

NDH 服務架構 (完整版)

NDH 微服務架構



Asset Servant Service

職責: Tag 映射和路由

儲存: 無 (無狀態服務)

依賴: IADL 定義 (從檔案系統讀取)



Model Service

職責: 3D 模型管理

儲存: Omniverse Nucleus / MinIO

功能: 模型上傳、下載、版本控制



Telemetry Service

職責: 時序數據查詢

儲存: TDengine / PI / InfluxDB

功能: 查詢、聚合、訂閱



Metadata Service

職責: 資產元數據管理

儲存: PostgreSQL

功能: CRUD 操作、搜索、過濾



Document Service

職責: 文檔管理

儲存: MinIO / S3

功能: 上傳、下載、版本控制



Sync Service

職責: 數據同步 (PI → TDengine)

儲存: 無 (只做數據搬運)

完整的資料流範例

應用場景: Dashboard 顯示資產完整資訊

```
// 前端 Dashboard 代碼

// 1. 獲取資產基本資訊 (Metadata Service)
const assetInfo = await fetch('/api/v1/assets/PUMP-001');
// 返回: { asset_id, manufacturer, rated_power, ... }

// 2. 獲取 3D 模型 (Model Service)
const modelUrl = await fetch('/api/v1/assets/PUMP-001/model');
// 返回: { model_url: "omniverse://server/.../pump001.usd" }

// 3. 在 Omniverse 中載入模型
omniverse.loadModel(modelUrl);

// 4. 獲取即時數據 (透過 Asset Servant)
const pressure = await fetch(
  '/api/v1/assets/PUMP-001/telemetry/discharge_pressure?latest=true'
);
// Asset Servant 映射 -> TDengine 查詢 -> 返回數據

// 5. 獲取文檔清單 (Document Service)
const docs = await fetch('/api/v1/assets/PUMP-001/documents');
// 返回: [{ type: "manual", url: "s3://bucket/..." }, ...]
```

數據流經的服務: - 步驟 1: Metadata Service → PostgreSQL - 步驟 2: Model Service → Omniverse Nucleus - 步驟 3: 前端直接連接 Omniverse - 步驟 4: Asset Servant (映射) → Telemetry Service → TDengine - 步驟 5: Document Service → MinIO/S3

Asset Servant 只參與步驟 4,且只負責映射,不儲存任何數據。

為什麼 Asset Servant 不儲存資料?

設計原則: 單一職責

Asset Servant 的唯一職責:

將抽象的、語義化的 Tag ID 映射到實際的、技術性的資料點名稱

如果 Asset Servant 儲存資料會導致: 1. ❌ 職責不清晰 (既做映射又做儲存) 2. ❌ 數據冗餘 (3D 模型、時序數據都有專門的儲存系統) 3. ❌ 擴展性差 (Asset Servant 需要處理各種資料類型) 4. ❌ 維護困難 (一個服務承擔太多責任)

設計原則: 關注點分離

Asset Servant: 專注於 Tag 映射和路由
Model Service: 專注於 3D 模型管理
Telemetry Service: 專注於時序數據查詢
Metadata Service: 專注於元數據管理
Document Service: 專注於文檔管理

每個服務都有明確的職責邊界,互不干擾。

Asset Servant 的完整定義

```
class AssetServant:
    """
    資產 Tag 映射服務

    職責:
    - ✓ 解析 IADL 定義, 建立 Tag 映射表
    - ✓ 將抽象 Tag ID 映射到實際資料點
    - ✓ 路由時序數據查詢到正確的資料庫

    不負責:
    - ✗ 儲存 USD 3D Model
    - ✗ 儲存時序數據
    - ✗ 儲存資產元數據
    - ✗ 儲存文檔檔案
    - ✗ 管理資產生命週期
    - ✗ 執行業務邏輯
    """

    def __init__(self, asset_id: str, iadl_path: str):
        """
        初始化 Asset Servant

        Args:
            asset_id: 資產 ID
            iadl_path: IADL 定義檔案路徑 (從檔案系統讀取)
        """
        self.asset_id = asset_id

        # 從檔案系統讀取 IADL (不儲存在 Asset Servant 中)
        iadl_definition = self._load_iadl(iadl_path)

        # 建立 Tag 映射表 (運行時記憶體, 不持久化)
        self.tag_mappings = self._build_tag_mappings(iadl_definition)

    def _load_iadl(self, path: str) -> dict:
        """從檔案系統載入 IADL 定義"""
        # IADL 儲存在 Git / 檔案系統, 不在 Asset Servant
        import yaml
        with open(path, 'r') as f:
            return yaml.safe_load(f)

    def build_tag_mappings(self, iadl: dict) -> dict:
        """建立 Tag 映射表 (只在記憶體中)"""
        mappings = {}
        for tag in iadl.get('data_tags', []):
            mappings[tag['tag_id']] = tag['source']
        return mappings

    async def get_tag_data(self, tag_id: str, start: datetime, end: datetime):
        """
        獲取 Tag 數據 (透過映射和路由)

        流程:
        1. 查找映射
        2. 路由到正確的 TSDB
        3. 返回數據

        注意: 不儲存數據, 只做查詢路由
        """
        mapping = self.tag_mappings[tag_id]
```

```
# 路由到正確的資料庫
if mapping['type'] == 'tdengine':
    return await self.tdengine_adapter.query(
        measurement=mapping['measurement'],
        start_time=start,
        end_time=end
    )
elif mapping['type'] == 'pi_point':
    return await self.pi_adapter.query(
        measurement=mapping['point_name'],
        start_time=start,
        end_time=end
    )
```

總結

Asset Servant 的儲存職責

儲存 (運行時記憶體): - ☒ Tag 映射表 (從 IADL 解析而來)

不儲存 (由其他服務負責): - ☒ USD 3D Model → Omniverse Nucleus / MinIO - ☒ 時序數據 → TDengine / PI System / InfluxDB - ☒ 資產元數據 → PostgreSQL - ☒ 文檔檔案 → MinIO / S3 - ☒ IADL 定義 → Git / 檔案系統

核心設計原則

1. **單一職責**: Asset Servant 只做 Tag 映射和路由
2. **無狀態**: 不持久化任何資料,可水平擴展
3. **關注點分離**: 每種資料類型由專門的服務和儲存系統管理
4. **清晰邊界**: 職責明確,易於維護和擴展

Asset Servant 是一個純粹的映射和路由服務,它不是資料儲存層,而是資料訪問的抽象層。