智能体数据治理工具设计文档

1. 项目概述

1.1 背景

本项目旨在构建一个自动化数据治理工具，实现以下核心功能：

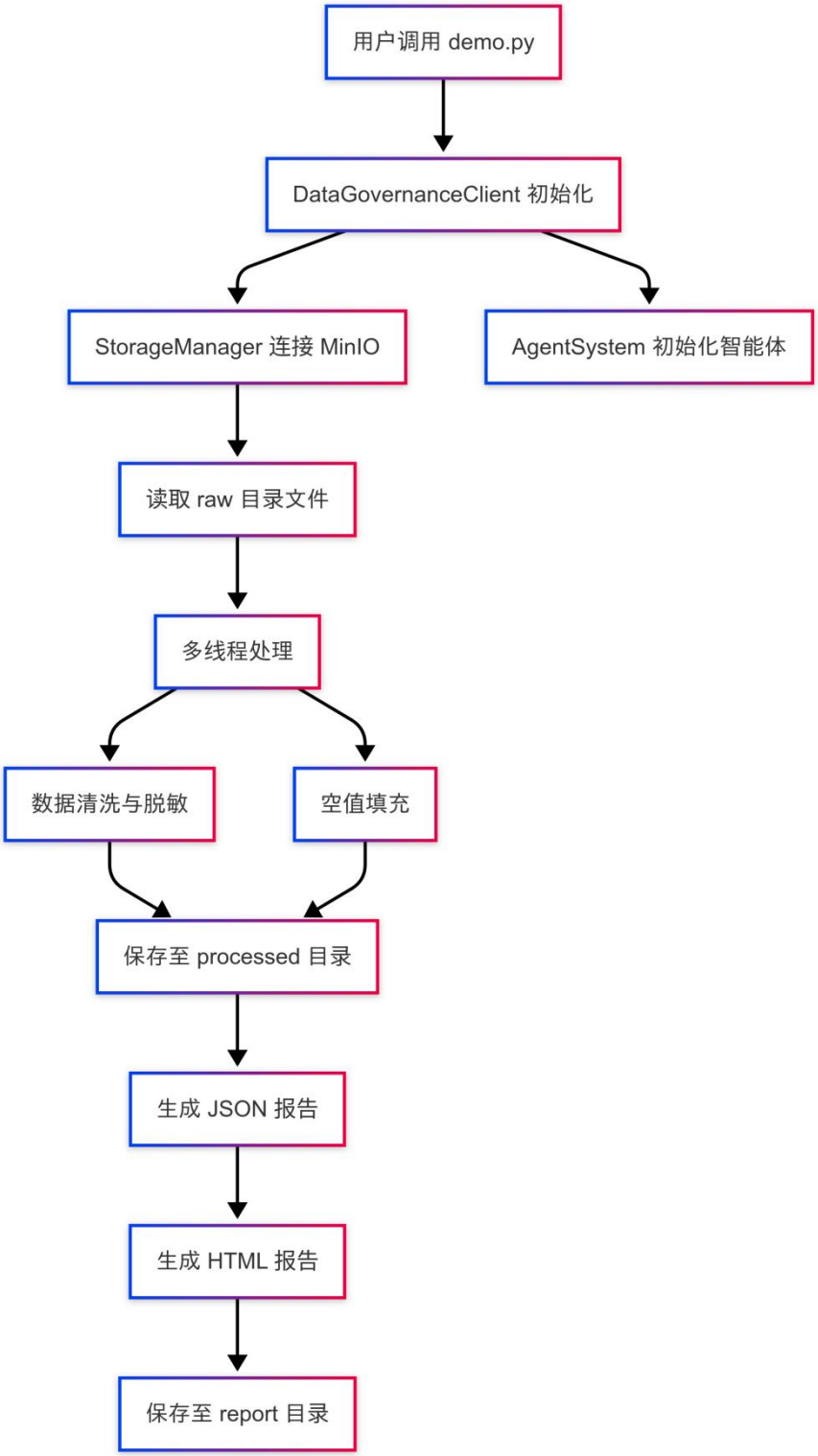
- ✓ 数据清洗与脱敏：对敏感字段进行哈希处理，优化空值填充。
- ✓ 数据质量验证：生成完整性报告和可视化图表。
- ✓ 分布式存储集成：与 MinIO 存储无缝对接，支持多格式文件处理。
- ✓ 智能体协作：通过多角色智能体（Orchestrator, DataCleaner 等）协调任务流程。

1.2 目标

- ✓ 自动化处理原始数据目录（`raw`）中的文件。
- ✓ 生成结构化报告（JSON）和交互式可视化报告（HTML）。
- ✓ 支持高并发处理（通过多线程优化性能）。

2. 系统架构

2.1 架构图

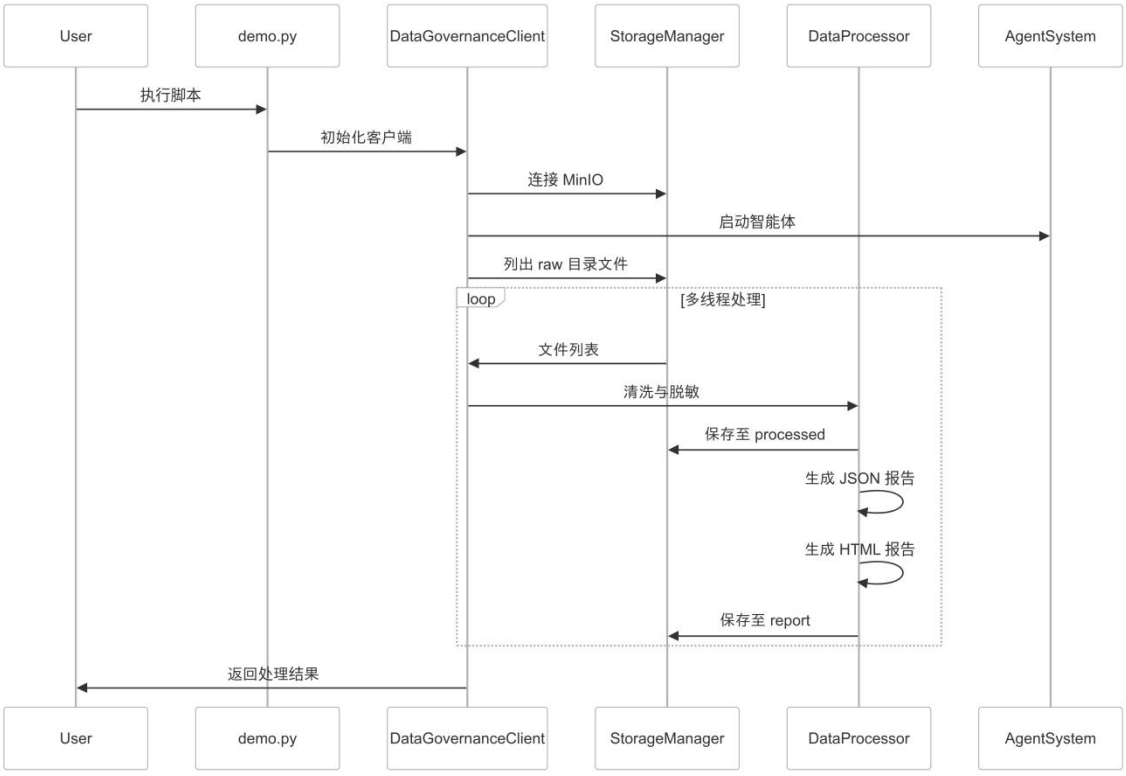

用户调用 demo.py
DataGovernanceClient 初始化
StorageManager 连接 MinIO
AgentSystem 初始化智能体
读取 raw 目录文件
多线程处理
数据清洗与脱敏
空值填充
保存至 processed 目录
生成 JSON 报告
生成 HTML 报告
保存至 report 目录

2.2 核心组件

组件	功能描述
DataGovernanceClient	用户接口类，协调存储管理、智能体协作和任务分发。
StorageManager封装	MinIO 操作，支持文件读写、目录管理。
DataProcessor	数据清洗、脱敏、空值填充，生成数据报告。
AgentSystem	管理五角色智能体，通过群组聊天协调任务流程。

3. 执行流程

3.1 流程图



3.2 关键步骤

3.2.1 初始化连接

- ✓ 通过 `DataGovernanceClient` 连接 MinIO 并验证权限。
- ✓ 启动智能体系统，初始化五个角色（Orchestrator, DataCleaner 等）。

3.2.2 文件处理

- ✓ 使用多线程（`ThreadPoolExecutor`）并发处理 `raw` 目录下的文件。
- ✓ 每个文件独立执行以下操作：
  - ■ 数据加载：根据文件格式(CSV/JSON/Parquet)读取为 DataFrame。
  - ■ 字段脱敏：对敏感字段（如 `name`）进行截断和 SHA-256 哈希。
  - ■ 空值填充：对数值型字段使用 `IterativeImputer` 进行优化填充。

3.2.3 报告生成

- ✓ JSON 报告：统计字段完整性、高频值和数据维度。
- ✓ HTML 报告：使用 Plotly 生成交互式柱状图，展示字段完整性比例。

3.2.4 结果存储

- ✓ 处理后的数据保存至 `processed` 目录。
- ✓ 报告文件保存至 `report` 目录，错误日志写入 `logs/errors`。

4. 实现原理

4.1 数据脱敏

```python
# 示例：敏感字段处理
df[col] = df[col].apply(
    lambda x: str(x)[:6] + hashlib.sha256(str(x).encode()).hexdigest()[:6]
)
```

- ✓ 截断前 6 位：保留部分原始信息供人工核对。
- ✓ SHA-256 哈希：对剩余内容哈希化，取前 6 位作为唯一标识。

4.2 空值填充

```python
imputer = IterativeImputer()
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])
```

- ✓ 迭代填充：基于数值字段的分布，通过回归模型预测缺失值。

4.3 报告生成

```python
# HTML 报告生成
fig = px.bar(x=fields, y=completeness, title="数据完整性报告")
html_content = fig.to_html(include_plotlyjs='cdn')
buffer.write(html_content.encode('utf-8'))
```

- ✓ Plotly 交互图表：支持缩放、悬停查看数值等操作。
- ✓ CDN 加载资源：减少 HTML 文件体积，提升加载速度。

5. 部署与运行

5.1 依赖安装

```bash
pip install pandas minio plotly scikit-learn autogen
```

5.2 配置文件示例

```python
client = DataGovernanceClient(
    minio_endpoint="192.168.1.100:9000",
    minio_access="admin",
    minio_secret="AlexHou@szim",
    ollama_config={
        "model": "deepseek-r1:70b",
        "base_url": "http://192.168.1.101:11434/v1",
        "api_key": "************************"
    },
    bucket="daas-user",
    base_path="alex"
)
```

5.3 运行命令

```bash
python demo.py
```

6. 扩展性与优化

6.1 扩展方向

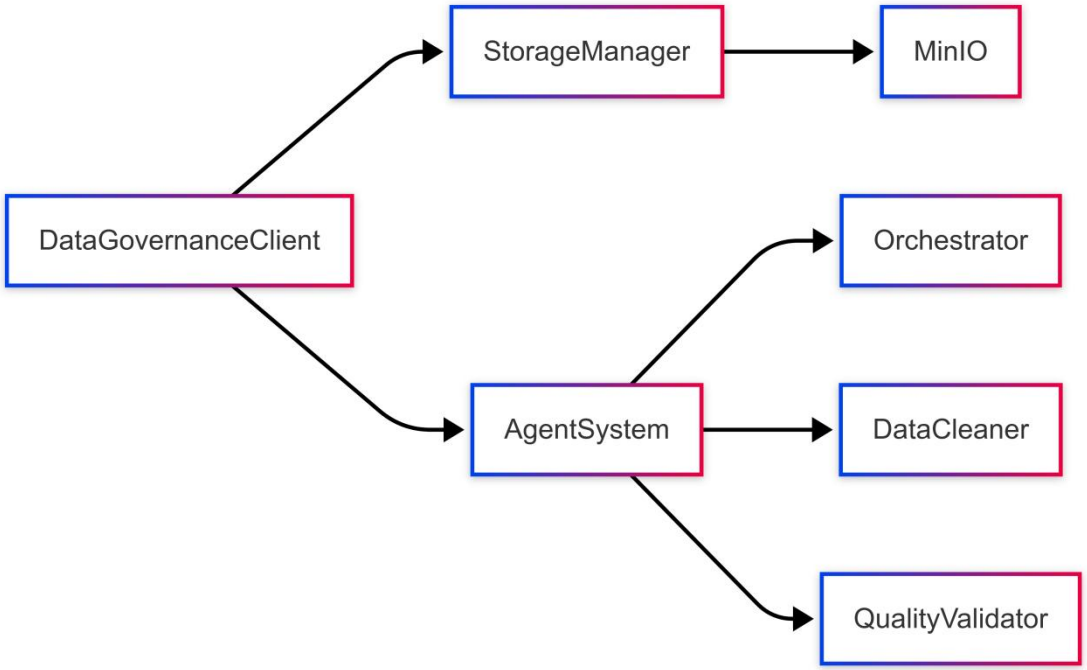
- ✓ 支持更多文件格式：通过扩展 `StorageManager` 的读写方法。
- ✓ 自定义脱敏规则：通过配置文件动态指定敏感字段和处理逻辑。
- ✓ 分布式处理：集成 Spark 或 Dask 处理大规模数据。

6.2 性能优化

- ✓ 缓存机制：对常用数据预处理结果缓存至内存或本地磁盘。
- ✓ 异步 IO：使用 `asyncio` 提升 MinIO 操作的并发效率。

附录

A. 系统模块关系图



B. 错误处理机制

- ✓ 异常捕获：所有文件操作封装在 `try-except` 块中。
- ✓ 日志记录：错误信息写入 MinIO 的 `logs/errors` 目录，格式为 JSON。