

技术规范说明书

本技术规范基于需求规格说明书：[需求规格说明书](#) 进行编制，所涉及内容仅满足该文档所述需求。

第一章：整体技术分层架构（Layered Architecture）

采用 DDD 经典四层架构，确保业务逻辑与技术细节的深度解耦。

层次	职责描述	核心技术组件
接入层 (Interfaces)	负责与外界交互，处理请求鉴权、入参校验与结果封装。	Spring MVC, OpenAPI/Swagger, JWT
应用层 (Application)	负责业务流程编排（Orchestration），不含业务逻辑。	Spring Service (Application), DTO
领域层 (Domain)	系统的灵魂。包含聚合根、实体、领域事件、业务策略。	POJO, Domain Services, Specification
基础设施层 (Infrastructure)	提供技术能力支持：数据库持久化、缓存、消息推送、外部集成。	MyBatis-Plus, PostgreSQL, Redis, RabbitMQ

第二章：项目拆分方案（Project Decomposition）

为了支持并行开发并降低部署复杂度，建议采用以下拆分方式：

2.1 后端工程（Back-End）

- `pjl-backend-core`：核心单体应用。包含空间、运营、财务、权限四大领域。采用 Package 级别隔离。
- `pjl-backend-hub`：数据集成网关。专门负责对接停车、962121、支付网关等异构系统，通过 MQ 与 `core` 通信。

2.2 前端工程（Front-End）

- `pjl-admin-web`：PC 端管理后台（Vue 3 + AntD）。
- `pjl-mobile-all`：移动端全功能工程（Uni-app）。
 - 构建目标 A: 业主端（小程序）。
 - 构建目标 B: 员工端（Android APK, PDA 专用）。

第三章：整体开发规范与准则（Unified Standards）

3.1 代码风格与语言规范

3.1.1 后端（Java/Spring Boot）

- **命名风格**：遵循 Google Java Style Guide。类名 `UpperCamelCase`，方法/变量名 `lowerCamelCase`，常量 `UPPER_SNAKE_CASE`。
- **实体映射**：
 - 数据库实体后缀为 `Entity`（如 `UserEntity`）。
 - 数据传输对象后缀为 `DTO`（用于接口输入）。
 - 视图对象后缀为 `VO`（用于接口输出）。
- **Lombok 使用**：强制使用 `@Data`，`@Builder`，`@NoArgsConstructor`，`@AllArgsConstructor` 减少样板代码。
- **异常处理**：严禁吞掉异常。必须通过自定义异常 `BusinessException` 抛出，由全局异常处理器统一捕获。

3.1.2 前端（Vue 3/TS）

- **组合式 API**：必须使用 `<script setup>` 语法。
- **类型安全**：严禁使用 `any`。所有 API 请求的 Payload 和 Response 必须定义 `interface`。
- **组件命名**：页面组件 `PascalCase`，公共组件以 `Base` 或 `App` 开头。

3.1.3 数据库对象命名规范（Database Naming）

基于文档中对 3.3 PostgreSQL 专项规范及第四章各领域 DDL 的定义：

- **表名前缀（Domain Prefix）**：按业务领域进行逻辑隔离：
 - `base_`：空间与主数据领域（如 `base_space_node`）。
 - `op_`：运营调度领域（如 `op_work_order`）。
 - `ast_`：设施设备领域（如 `ast_equipment`）。
 - `fin_`：财务计费领域（如 `fin_bill`）。
 - `iam_`：权限与账户领域（如 `iam_user`）。
 - `hub_ / int_`：数据集成中心（如 `int_external_log`）。
- **审计辅助表**：状态变更记录表后缀为 `_step` 或 `_audit`。
- **主键命名**：所有主键字段名统一为 `id`，类型为 `uuid`。

- **索引命名:**

- 普通索引: `idx_{table}_{column}`。
- 唯一索引: `uk_{table}_{column}` (基于 DDL 唯一约束逻辑)。

3.1.4 状态机与枚举规范 (State Machine & Enums)

- **后端枚举:** 必须实现 `IEnum` 接口。
- **数据库存储:** 数据库中存储枚举的 **String 名称** (即枚举名), 而非数字索引。
- **状态常量:** 必须使用全大写蛇形命名, 如 `PENDING`、`IN_PROGRESS`、`PARTIAL_PAID`。

3.2 领域开发规范 (DDD Enforcer)

在开发具体功能模块时, Trae 必须遵循以下分层调用规则:

1. **接口层 (Controller):** 仅负责参数校验、转换和调用应用服务。不允许包含业务逻辑。
2. **应用层 (Application Service):** 负责事务管理、编排多个领域服务。
3. **领域层 (Domain Service/Entity):** **核心业务逻辑必须在此实现**。聚合根负责维护业务的一致性约束。
4. **基础设施层 (Infrastructure):** 处理 SQL、MQ 发送、第三方 API 调用适配。

3.3 PostgreSQL & JSONB 专项规范

- **字段定义:** 所有主键使用 `UUID`, 所有时间戳使用 `timestamp`。
- **JSONB 处理逻辑:**
 - **映射规则:** 在 Java 实体中使用 `Map<String, Object>` 或具体的 POJO, 并标记 MyBatis-Plus 的 `@TableField(typeHandler = JacksonTypeHandler.class)`。
 - **索引约束:** 若需对 JSONB 内部字段查询, 必须在 DDL 中要求建立 `GIN` 索引。
- **空间索引:** 所有经纬度字段必须使用 `geometry(Point, 4326)` 并建立 `GIST` 索引。

3.4 API 交互协议规范 (RESTful)

所有接口必须遵循以下格式, 确保前后端联调零沟通成本:

3.4.1 统一响应格式

所有接口响应必须封装在统一的结果对象中, 严禁直接返回业务 DTO 或 VO。

代码块

```
1  {
2    "code": 200,           // 业务状态码, 200 为成功
3    "message": "success", // 提示信息
```

```

4   "data": { ... },           // 结果对象
5   "timestamp": 1672531200000
6   }

```

- **格式要求**: 包含 `code` (200 为成功)、`message` (提示信息)、`data` (结果对象) 和 `timestamp` (13 位时间戳)。
- **异常处理**: 严禁吞掉异常, 必须通过自定义 `BusinessException` 抛出, 由全局异常处理器封装为上述格式。
- **审计留痕**: 当返回 `SUCCESS` 且涉及状态变更时, 必须确保已在同一个事务中插入了足迹记录 (Trace Log)。
- **幂等性校验**: 对于重复点击导致的请求, 应返回成功或特定的幂等性提示, 确保状态不发生异常跳变。

3.4.2 状态码规则

1. 基础系统状态码 (System Base Codes)

用于标识请求的物理处理状态及通用鉴权结果。

状态码 (Code)	业务含义	说明
200	SUCCESS	业务处理成功, 对应响应格式中的成功标识。
400	BAD_REQUEST	接口层入参校验失败, 属于接口层 (Interfaces) 职责。
401	UNAUTHORIZED	JWT 鉴权失效或未携带令牌, 由接入层处理。
500	INTERNAL_ERROR	系统内部未知异常, 由全局异常处理器统一捕获。

2. 框架与安全约束码 (Framework & Security Codes)

针对多租户隔离与通用开发准则的强制约束。

状态码 (Code)	业务含义	说明
1001	TRANSITION_DENIED	状态转换非法: 违反了状态转换的“白名单”校验规则。
1002	PROJECT_CONTEXT_ERROR	项目上下文异常: Header 缺失 X-Project-ID 或校验失败。
1003	OPTIMISTIC_LOCK_FAIL	并发更新冲突: 数据库 version 校验失败, 触发乐观锁防护。
1004	READ_ONLY_VIOLATION	只读状态保护: 对象进入终态后, 尝试修改禁止编辑的字段。

3. 业务领域特定状态码 (Domain Specific Codes)

针对运营、巡检、财务等核心业务场景的逻辑校验。

状态码 (Code)	业务含义	说明
2001	SEQUENCE_VIOLATION	巡检时序错误: 未按 route_config 规定的顺序提交点位结果。
2002	ALARM_MERGED	离线告警合并: 判定为 15 分钟内的重复告警, 策略转为合并为附件。
3001	BILL_LOCKED	账单锁定: 账单在审批流 (BPM) 期间被锁定, 禁止支付操作。
3002	ARREARS_RESTRICTED	欠费权限限制: 账户因逾期欠费, 被 is_restricted 标记限制功能权限。
4001	HUB_PARSE_FAILED	集成解析失败: 数据集成中心适配器转换 DTO 结构失败。

3.4.3 路由规则

接口路径必须符合 `/api/v1/{domain}/{resources}` 的结构, 且动词必须与 HTTP Method 对应 ()

- `GET /api/v1/{domain}/{resources}` - 获取资源
- `POST /api/v1/{domain}/{resources}` - 创建资源
- `PUT /api/v1/{domain}/{resources}/{id}` - 更新资源
- `DELETE /api/v1/{domain}/{resources}/{id}` - 逻辑删除

3.4.4 命名与映射规范 (Naming Strategy)

- **入参:** 后缀必须为 `DTO`。
- **出参:** 后缀必须为 `VO`。
- **多租户安全:** AI 生成的 API 必须在拦截器层实现 `ProjectContextInterceptor`, 从 Header 获取 `X-Project-ID` 并校验 `1002` 错误码 (Project Context Error)。

3.5 自测与测试覆盖率要求 (Test Driven)

Trae 执行指令约束: 在生成任何 Service 层逻辑后, 必须同步生成对应的测试代码。

1. 单元测试 (JUnit 5 + Mockito) :

- **领域模型测试:** 必须 100% 覆盖 Entity 内部的业务方法 (如计费公式、状态机转换)。
- **Mock 依赖:** 使用 `@Mock` 模拟数据库和外部 API, 确保测试可以在无数据库环境下运行。

2. 集成测试:

- 针对 Data Hub 的解析器, 必须提供 `sample.json` 的测试用例, 验证解析后的 `standard_data` 与预期一致。

3. 断言风格: 使用 `AssertJ` 编写易读的断言, 如

```
assertThat(result).isNotNull().hasFieldOrPropertyWithValue("status", "PAID");
```

3.6 并行开发冲突规避 (Trae Sync Strategy)

当多端并行开发时（如后端开发 API，前端开发页面）：

- **Contract First（契约优先）**：后端必须先生成接口定义和 DTO，并在 `interfaces` 模块中导出。
- **Mock Server**：前端在后端 API 未实现前，必须要求 Trae 生成基于 `vite-plugin-mock` 的模拟数据，且数据结构必须与后端 DTO 严格一致。
- **Database Migrations**：所有数据库变更必须通过脚本管理，严禁直接在数据库手动改表。

3.7 UUID v7 主键开发规则（RFC 9562 标准）

1. 数据库层（PostgreSQL）：

- 所有主键字段类型必须设为 `UUID`。
- 表 DDL 默认值统一调用自定义函数 `uuid_generate_v7()`。
- 必须确保初始化脚本中包含该函数的定义逻辑。

代码块

```
1 CREATE OR REPLACE FUNCTION uuid_generate_v7()
2 RETURNS uuid AS $$
3 DECLARE
4     v_time bigint;
5     v_seq_random_a int;
6     v_seq_random_b bigint;
7     v_res text;
8 BEGIN
9     -- 1. 获取当前毫秒级时间戳 (48位)
10    v_time := (extract(epoch from now()) * 1000)::bigint;
11
12    -- 2. 生成随机部分 A (12位, 用于填充 version 后的空间)
13    v_seq_random_a := floor(random() * 4096)::int;
14
15    -- 3. 生成随机部分 B (62位)
16    v_seq_random_b := (floor(random() * 4611686018427387904)::bigint);
17
18    -- 4. 拼接并强制设置版本号 (7) 和变体位 (Variant 2)
19    v_res := lpad(to_hex(v_time), 12, '0') ||
20              '7' || lpad(to_hex(v_seq_random_a), 3, '0') ||
21              case when (v_seq_random_b >> 62) = 0 then '8'
22                   when (v_seq_random_b >> 62) = 1 then '9'
23                   when (v_seq_random_b >> 62) = 2 then 'a'
24                   else 'b' end ||
25              lpad(to_hex(v_seq_random_b & x'3fffffffffffffff'::bigint), 15,
26                  '0');
27    RETURN v_res::uuid;
```

```
28 END;  
29 $$ LANGUAGE plpgsql VOLATILE;
```

- **UUID v7 的内部结构拆解，128 位数据的分布：**

- **前 48 位 (Timestamp)：**精确到毫秒的时间戳。这是保证**“有序性”**的关键。
- **4 位 (Version)：**固定为 `0111`（即十六进制的 `7`），标识这是 v7 版本。
- **后续位 (Random/Sequence)：**由随机数填充，保证在同一毫秒内生成多个 ID 时依然保持**“唯一性”**。

2. 后端开发层 (Java)：

- **类型约束：**严禁使用 `String` 存储 UUID，实体类和业务逻辑中必须统一使用 `java.util.UUID` 类型。
- **自动填充：**集成 MyBatis-Plus 的 `IdentifierGenerator`，使用符合规范的生成器（如 JUG 库或手写工具类）在应用层生成有序的 UUID v7。
- **映射处理：**确保 Jackson 和 MyBatis 的 TypeHandler 正确处理 `java.util.UUID` 到数据库 `uuid` 类型的映射。

3. 前端/移动端层 (Uni-app)：

- 如需在端上离线生成 ID，必须使用支持 UUID v7 的 JS 库，严禁生成随机的 v4 版本，以保证索引的顺序性。

4. 性能准则：

- 利用 UUID v7 的时间有序性，在进行时间范围查询或排序时，应优先考虑利用主键索引。

3.8 全局状态机与枚举定义 (Global State Machine & Enums)

本规范定义了平台内所有核心业务对象的生命周期状态。开发要求：

- **后端：**Java 实体必须使用 `IEnum` 接口，数据库存储 `String`（枚举名）。
- **变更记录：**状态每发生一次变更，必须记录到对应的 `_step` 或 `_audit` 表中。

3.8.1 运营调度领域 (Operation Domain)

1. 综合工单状态 (op_work_order.status)

状态常量 (Enum)	业务名称	前置状态 (Allowed From)	业务触发动作/说明
PENDING	待分派	(START)	工单初始创建, 进入待处理池
ASSIGNED	已指派	PENDING	管理员或系统自动选择了执行人
ACCEPTED	已接单	ASSIGNED	执行人在移动端点击“确认接单”
PROCESSING	处理中	ACCEPTED	执行人到达现场, 点击“开始处理”
PENDING_ACCEPT	待验收	PROCESSING	处理完成, 上传存证照片后提交验收
CLOSED	已关闭	PENDING_ACCEPT	报修人或管理员点击“验收通过”
CANCELED	已取消	PENDING, ASSIGNED	工单被撤回或判定为无效

2. 巡检任务状态 (op_inspection.status)

状态常量 (Enum)	业务名称	说明
NOT_STARTED	未开始	巡检计划已生成, 等待到达计划开始时间
IN_PROGRESS	巡检中	巡检员已扫描第一个点位, 任务激活
COMPLETED	正常完成	所有规定点位均已打卡, 且检查结果全为“正常”
EXCEPTION	异常完成	巡检已结束, 但其中包含至少一个“故障”项
MISSED	漏检	规定时间结束时仍有未打卡点位, 系统强制关闭

3.8.2 财务计费领域 (Finance Domain)

1. 账单状态 (fin_bill.status)

状态常量 (Enum)	业务名称	业务规则约束
UNPAID	未支付	默认状态。应收金额 > 0, 实收金额 = 0
PARTIAL_PAID	部分支付	$0 < \text{实收金额} < (\text{应收金额} + \text{滞纳金})$
PAID	已结清	实收金额 \geq (应收金额 + 滞纳金), 账单闭环
OVERDUE	已逾期	current_date > deadline_date 且状态为非 PAID
ADJUSTED	已调账	账单被红冲、作废或金额被手动修正为无效

3.8.3 空间资产领域 (Space Domain)

1. 房屋/铺位状态 (base_room_detail.house_status)

状态常量 (Enum)	业务名称	计费联动影响
VACANT	空置	仅产生基础物业费，不产生水电气等使用费
SOLD	已售待收	财务开始计费，但业主尚未正式入伙
DECORATING	装修中	需缴纳装修管理费、装修押金
OCCUPIED	已入住	正常产生所有类别费用
RENTED	已出租	计费通知单可能需要区分发送给业主或租客

2. 身份认证状态 (base_ownership.audit_status)

状态常量 (Enum)	业务名称	功能权限
SUBMITTED	已提交	仅能查看进度，无空间操作权限
AUDITING	审核中	后台客服正在核对资料
VERIFIED	已认证	获得开门、缴费、报修等完整权限
REJECTED	已驳回	权限被收回，需重新提交资料

3. 设施设备状态 (base_equipment.status)

状态常量 (Enum)	业务名称	自动化联动规则
NORMAL	正常运行	设备处于健康运行状态
FAULT	故障待修	自动创建一条维修工单
REPAIRING	维修中	关联维修工单正在处理中
OUT_OF_SERVICE	停用	设备暂时退出服务（如季节性停用），触发任务挂起/作废。
SCRAPPED	报废	设备永久移除，触发所有关联未完成任务强制终止。

3.8.4 流程审批 (BPM Domain)

1. 审批件状态 (proc_instance.status)

状态常量 (Enum)	业务名称	后续动作
DRAFT	草稿	仅发起人可见，未进入审批流
IN_REVIEW	审核中	任务在节点间流转，业务对象被锁定
APPROVED	已通过	触发业务回调（如：工单状态自动变为 CLOSED）
REJECTED	已驳回	流程终止，发起人需修改后重新发起
CANCELED	已撤回	发起人主动取消，解除业务对象锁定

3.8.5 数据集成中心 (Data Hub)

1. 原始报文处理状态 (hub_raw_message.process_status)

状态常量 (Enum)	业务名称	说明
RECEIVED	已接收	异构系统原始报文落库暂存
PARSED	已解析	适配器转换 DTO 结构成功
CONSUMED	已消费	业务逻辑处理完成并归档
FAILED	处理失败	数据清洗或逻辑校验不通过

3.9 状态机开发与校验规范 (Technical Constraints)

3.9.1 状态转换的“白名单”校验 (Transition Whitelist)

规则: 严禁在业务代码中直接 `setStatus()`。必须通过一个统一的校验器, 检查当前状态是否允许跳转到目标状态。

- **实现要求:** 在 `Service` 层变更状态前, 调用 `validateTransition(current, target)`。
- **API 行为约束:** 如果 AI 生成的代码中直接修改状态而未进行前置校验, 应判定为 Bug。

3.9.2 状态变更的原子性与审计 (Atomic Audit)

规则: 每一次状态变更必须伴随一条“足迹记录 (Trace Log)”。

- **技术要求:** 变更状态的 SQL 必须与插入 `op_work_order_step` 或 `fin_transaction_audit` 的 SQL 放在同一个 `@Transactional` 事务中。
- **记录内容:** 必须包含 `operator_id` (操作人)、`previous_status` (原状态)、`target_status` (新状态) 以及 `change_reason` (变更原因/备注)。

3.9.3 并发冲突防护 (Optimistic Locking)

规则: 针对高并发场景 (如多个客服同时抢占一个派单), 必须使用乐观锁。

- **实现要求:** 所有涉及状态机的表必须包含 `version` 字段。
- **SQL 示例:**

```
UPDATE table SET status = 'ACCEPTED', version = version + 1 WHERE id = ? AND version = ? AND status = 'ASSIGNED'。
```

3.9.4 领域事件触发机制 (Event-Driven)

规则: 核心状态到达“终态”或“关键节点”时, 必须抛出领域事件。

- **联动场景:**
 - 账单变为 `PAID` \rightarrow 触发财务月结统计并发送 Push 通知。

- 设备变为 `FAULT` \rightarrow 自动触发运营域报修工单创建。
- **技术选型**：内部解耦使用 Spring `ApplicationEvent`，跨服务解耦使用 RabbitMQ。

3.9.5 “只读”状态保护 (Read-Only Guard)

规则：当对象进入“终态”后，其核心业务字段应变为只读。

- **具体表现**：
 - 工单处于 `CLOSED` 状态时，禁止修改维修内容或执行人。
 - 账单处于 `PAID` 或 `ADJUSTED` 状态时，禁止修改 `receivable_amount`（应收金额）。

3.9.6 状态机规则速查表 (供 AI 检索)

约束项	强制程度	检查点 (Check Point)
禁止跳级	必须 (Must)	校验 Allowed From 路径
审计留痕	必须 (Must)	检查是否关联插入 <code>_step</code> 或 <code>_log</code> 表
乐观锁	建议 (Should)	检查 update 语句是否带 version
幂等性	必须 (Must)	即使重复点击“支付”，状态也不应异常跳变

3.9.7 离线数据同步与冲突合并策略 (Offline Sync Strategy)

规则：前端 (PDA) 在网络恢复后上传的数据，必须经过以下“三步清洗法”方可入库，严禁直接 Insert。

- **临时 ID 映射 (ID Mapping)**
 - **前端**：离线创建的数据使用本地生成的 `UUID v7`，并标记 `is_local=true`。
 - **后端**：接收到数据后，必须重新生成服务端的正式 UUID，并建立 `client_temp_id` \rightarrow `server_real_id` 的映射关系返回给前端，前端据此更新本地缓存状态。
- **时间窗口去重 (Time Window Deduplication)**
 - **场景**：离线期间 (如 10:00-10:10)，IoT 设备可能已自动上报了故障。
 - **算法**：

代码块

```

1 // 伪代码逻辑
2 if (uploadData.type == "ALARM") {
3     // 检查数据库中是否存在 同一设备 在 发生时间前后 15分钟 的活跃工单boolean
4     exists = workOrderRepo.existsByDeviceIdAndCreatedTimeBetween(
5         uploadData.deviceId,
6         uploadData.occureTime.minusMinutes(15),
7         uploadData.occureTime.plusMinutes(15)
8     );

```

```
8
9     if (exists) {
10         return Strategy.MERGE_AS_ATTACHMENT; // 策略转为：合并为附件
11     }
12 }
13 return Strategy.CREATE_NEW; // 策略：新建工单
```

- **合并处理 (Merge Logic)**


- 若判定为重复，系统不创建新工单，而是将离线记录中的**照片**和**描述**，作为一条“追加记录 (Append Log)”插入到已存在的工单历史中，并标记来源为“离线补传”。

3.10 鉴权拦截器约束

实现要求：

- AI 必须实现一个 `ProjectContextInterceptor`。
- **拦截逻辑：**
 - 从 Header 获取 `X-Project-ID`。
 - 查询 `iam_user_project_role` 表，获取 `List<Role>`。
 - 根据角色展开获取 `List<PermissionID>`。
 - 校验当前接口的 `@RequiresPermission` 注解值是否在列表中。

异常处理：若 `X-Project-ID` 缺失或校验失败，统一返回错误码 `1002 (Project Context Error)`。

 **操作建议：**将上述内容保存为 `RULES.md`。每当你开启一个新任务时，向 Trae 发送：“请阅读 `RULES.md` 中的规范。现在开始开发 [某功能模块]，请严格遵循 **DDD 分层架构**、**UUID 主键** 和 **JUnit 5 单元测试覆盖率** 要求。”

第四章：核心业务域设计明细 (Domain Design Summary)

4.1 空间与主数据领域 (Space & Master Data Domain)

- **职责：**构建物理数字孪生底座，管理所有静态资源及其拓扑关系。
- **聚合根：** `SpaceNode` (空间节点)
- **领域服务：**
 - `SpaceHierarchyService`：负责节点移动（如将单元 A 从楼栋 1 移至楼栋 2）的层级校验与路径重建 (Ltree/Path 维护)。

- `CustomerAssetBindingService`：处理“人-房-车”绑定，需在事务中同步更新 `base_customer` 扩展属性。
- 领域事件：
 - `SpaceNodeArchivedEvent`：空间注销。下游动作：通知财务领域立即触发该空间的“断点结算”。
- 核心逻辑：通过 `node_path` 实现 $O(1)$ 复杂度的层级检索。
- UML 关键关联：`SpaceNode` 为聚合根，`RoomDetail` 和 `CarSpace` 为 1:1 扩展实体。
- 数据结构 DDL：

代码块

```

1  -- 初始化扩展
2  CREATE EXTENSION IF NOT EXISTS "ltree";
3  CREATE EXTENSION IF NOT EXISTS "postgis";
4
5  -- 空间节点表：承载物理拓扑
6  CREATE TABLE "base_space_node" (
7    "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
8    "parent_id" uuid REFERENCES "base_space_node"("id"),
9    "name" varchar(100) NOT NULL,
10   "node_path" ltree NOT NULL,          -- 示例: 'P1.B1.U1.R101'
11   "node_type" varchar(20) NOT NULL,   -- PROJECT, BUILDING, UNIT, ROOM,
12   "geo_location" geometry(Point, 4326),
13   "ext_props" jsonb DEFAULT '{}',     -- 存储层高、朝向等
14   "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
15  );
16  CREATE INDEX "idx_base_space_node_path" ON "base_space_node" USING GIST
17  ("node_path");
18  -- 房产明细表
19  CREATE TABLE "base_room_detail" (
20    "space_id" uuid PRIMARY KEY REFERENCES "base_space_node"("id"),
21    "room_no" varchar(50),
22    "build_area" numeric(12,4),
23    "fee_area" numeric(12,4),
24    "house_status" varchar(20) DEFAULT 'VACANT', -- VACANT, SOLD, OCCUPIED
25    "is_rented" boolean DEFAULT false
26  );
27
28  -- 权属关联表 (核心纽带)
29  CREATE TABLE "base_ownership" (
30    "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
31    "space_id" uuid NOT NULL,          -- 关联 Room 或 CarSpace

```

```

32     "person_id" uuid NOT NULL,           -- 关联 iam_user.id
33     "owner_type" varchar(20),          -- OWNER, FAMILY, TENANT
34     "is_primary" boolean DEFAULT false,
35     UNIQUE("space_id", "person_id", "owner_type")
36 );

```

4.2 运营调度领域 (Operation & Dispatch Domain)

- **职责:** 处理“人、事、时、地”的动态匹配，实现工单、报修、维护、访客申请与排班等的闭环。
- **聚合根:** `WorkOrder` (工单)、`InspectionTask` (巡检任务)
- **领域服务:**
 - `SequenceValidationService`: **核心逻辑**。校验当前提交的点位是否符合 `route_config` 中的 `seq` 顺序，非法则抛出 `BusinessException`。
 - `AutoDispatchService`: 基于员工负载 (Load) 和地理位置 (PostGIS 距离计算) 的自动分派算法。
- **领域事件:**
 - `InspectionAbnormalDetectedEvent`: 发现异常。**下游动作:** 自动创建关联维修工单。
- **核心逻辑:**
 - 基于状态机驱动工单流转，基于实时在岗状态匹配派单。
 - **巡检点位状态流转约束:** 点位状态必须从 `PENDING` -> `ARRIVED` (扫码+LBS) -> `COMPLETED`。未 `ARRIVED` 的点位无法提交结果数据。
- **数据结构 DDL:**

代码块

```

1  -- 综合工单主表
2  CREATE TABLE "op_work_order" (
3      "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
4      "order_no" varchar(32) UNIQUE NOT NULL,
5      "category" varchar(20) NOT NULL,          -- REPAIR, COMPLAINT, INSPECTION
6      "source" varchar(20) NOT NULL,           -- C_END, GOV_962121
7      "location_id" uuid REFERENCES "base_space_node"("id"),
8      "current_handler_id" uuid,
9      "status" varchar(20) DEFAULT 'PENDING',
10     "sla_deadline" timestamp,
11     "ext_info" jsonb DEFAULT '{}',
12     "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
13 );
14
15 -- 人员排班与考勤

```

```

16 CREATE TABLE "op_shift_plan" (
17     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
18     "staff_id" uuid NOT NULL,
19     "shift_date" date NOT NULL,
20     "start_time" time NOT NULL,
21     "end_time" time NOT NULL,
22     UNIQUE("staff_id", "shift_date")
23 );
24
25 CREATE TABLE "op_attendance" (
26     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
27     "staff_id" uuid NOT NULL,
28     "current_status" varchar(20),          -- ON_DUTY, OFF_DUTY
29     "last_geo" geometry(Point, 4326),
30     "update_time" timestamp DEFAULT CURRENT_TIMESTAMP
31 );
32
33 -- 统一巡检任务表 (支撑 3.2.3 设备巡检 & 3.3.1 品质巡更)
34 CREATE TABLE "op_inspection_task" (
35     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
36     "plan_id" uuid,                        -- 关联的计划ID
37     "task_batch_no" varchar(32) NOT NULL,  -- 批次号
38     "target_type" varchar(20) NOT NULL,    -- DEVICE (设备) 或 SPACE (区域)
39     "target_id" uuid NOT NULL,            -- 具体的设备ID或空间ID
40     "route_config" jsonb NOT NULL,        -- 存储具体的巡检路线和检查项快照
41     "executor_id" uuid,                  -- 执行人
42     "actual_geo" geometry(Point, 4326),   -- 实际打卡坐标
43     "status" varchar(20) DEFAULT 'PENDING', -- 对应 3.8.1 定义的状态
44     "result_data" jsonb,                 -- 存储巡检结果 (温度、压力、照片URL)
45     "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
46 );
47 -- 索引建议: 针对批次号和执行人建索引, 加速移动端查询
48 -- 在 op_inspection_task 表的 route_config 字段增加约束说明
49 COMMENT ON COLUMN "op_inspection_task"."route_config" IS
50 '巡检路线静态快照。格式示例:
51 [
52   {
53     "seq": 1,                          -- 执行顺序索引
54     "point_id": "uuid",                 -- 物理点位/设备ID
55     "point_name": "泵房A",
56     "is_required": true,                 -- 是否必点
57     "check_items": [...],               -- 该点位的检查项快照
58     "status": "PENDING",                 -- PENDING, COMPLETED, SKIPPED
59     "completed_at": null                 -- 完成时间戳
60   },
61   ...
62 ]';

```

```

63
64 -- 访客预约申请表 (C端业务入口)
65 CREATE TABLE "op_visitor_invite" (
66     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
67     "project_id" uuid NOT NULL,           -- 所属项目
68     "owner_id" uuid NOT NULL,           -- 发起预约的业主ID
69     "visitor_name" varchar(64) NOT NULL, -- 访客姓名
70     "visitor_phone" varchar(20),        -- 访客手机号
71     "plate_no" varchar(15),             -- 访客车牌号
72     "visit_date" date NOT NULL,        -- 预计到访日期
73     "valid_hours" int DEFAULT 4,        -- 凭证有效时长(小时)
74     "status" varchar(20) DEFAULT 'ACTIVE', -- ACTIVE(生效), CANCELLED(撤销),
        COMPLETED(已到访)
75     "create_time" timestamp DEFAULT now()
76 );
77
78 -- 索引: 加速闸机识别和业主查询
79 CREATE INDEX "idx_visitor_plate" ON "op_visitor_invite" ("plate_no");
80 CREATE INDEX "idx_visitor_owner" ON "op_visitor_invite" ("owner_id");

```

• 核心算法：时序校验逻辑

当移动端（PDA/App）提交某个点位的检查结果时，后端必须执行“前置依赖检查”。

逻辑描述：

- 输入：task_id, current_point_id, result_data
- 处理逻辑：
 - 从数据库加载该任务的 route_config。
 - 定位 current_point_id 在 route_config 数组中的索引 N。
 - 时序判定：检索索引 0 到 N-1 的所有点位，若存在 is_required = true 且 status != 'COMPLETED' 的点位，则判定为“越序作业”。
- 输出：抛出异常码 403-SEQUENCE-ERROR，提示：“请先完成前置点位：[点位名称] 的巡检”。

• 数据库原子更新函数（UDF）

为了防止并发提交导致的顺序错乱，使用存储过程处理点位状态更新：

代码块

```

1 CREATE OR REPLACE FUNCTION func_submit_inspection_point(
2     p_task_id uuid,
3     p_point_id uuid,
4     p_result jsonb

```

```

5 ) RETURNS jsonb AS $$
6 DECLARE
7     v_route jsonb;
8     v_idx int;
9     v_pre_check boolean;
10 BEGIN -- 1. 获取任务路线配置SELECT route_config INTO v_route FROM
      op_inspection_task WHERE id = p_task_id FOR UPDATE;
11
12     -- 2. 检查前置必填点位是否已完成-- 逻辑: 查找索引小于当前点位、且为必填但未完成的
      点位SELECT EXISTS (
13         SELECT 1 FROM jsonb_array_elements(v_route) WITH ORDINALITY as
      elem(val, ord)
14         WHERE (val->>'is_required')::boolean = true
15         AND val->>'status' != 'COMPLETED'AND ord < (
16         SELECT ord FROM jsonb_array_elements(v_route) WITH ORDINALITY
      as target(val, ord)
17         WHERE (val->>'point_id')::uuid = p_point_id
18         )
19     ) INTO v_pre_check;
20
21     IF v_pre_check THENRETURN jsonb_build_object('success', false, 'msg',
      'Sequence violation: previous points not completed');
22     END IF;
23
24     -- 3. 原子更新该点位状态及结果-- 使用 jsonb_set 定位并更新数组中的特定对象状态
25     UPDATE op_inspection_task
26     SET route_config = (
27         SELECT jsonb_agg(
28             CASE WHEN (elem->>'point_id')::uuid = p_point_id THEN
29                 elem || jsonb_build_object('status', 'COMPLETED',
      'completed_at', now(), 'result', p_result)
30             ELSE elem END
31         )
32         FROM jsonb_array_elements(route_config) AS elem
33     )
34     WHERE id = p_task_id;
35
36     RETURN jsonb_build_object('success', true);
37 END;
38 $$ LANGUAGE plpgsql;

```

● 特殊场景处理

- **跳点申请**: 若现场因客观原因 (如泵房钥匙缺失) 无法按顺序巡检, 人员需在 App 发起“跳点申请”, 由 7.2 BPM 模块审批。审批通过后, 后端将该点位的 `is_required` 临时更新为 `false` 或 `status` 更新为 `SKIPPED`, 以解锁后续点位。

- **离线容错**：在 3.9.7 离线同步逻辑中，若离线提交的多个点位时序正确（按本地时间戳），则后端同步时应予以认可；若离线提交的点位本身就存在乱序，则整批次退回。

4.3 设施设备领域（Asset Domain）

- **职责**：对应需规 3.2 模块，负责资产台账、维保计划（3.2.2）与巡检任务（3.2.3）。
- **聚合根**：Equipment（设备实体）
- **领域服务**：
 - MaintenanceScheduler：解析 Cron 表达式，负责任务的“准时生成”。
 - AssetHealthEvaluator：根据报修频率与巡检记录自动计算健康分（0-100）。
 - HardwareAccessController：
 - **白名单下发**：接收到访客预约指令后，通过 pjl-backend-hub 将车牌号实时下发至车行道闸相机的白名单库，或将二维码特征值下发至人行闸机。
 - **离线容错**：若硬件网关断网，系统通过心跳机制记录未下发任务，待网络恢复后自动补发。
- **领域事件**：
 - AssetScrappedEvent：设备报废。**下游动作**：触发 5.5 节定义的熔断逻辑，注销所有 PENDING 任务。
 - AssetFaultEvent：设备感应到故障。**下游动作**：联动 4.2 领域自动创建报修工单。
- **数据结构 DDL**：

代码块

```

1  -- 4.3.1 Equipment Master Data (Core Aggregate)
2  CREATE TABLE "ast_equipment" (
3      "id" UUID PRIMARY KEY DEFAULT gen_random_uuid(),
4      "asset_no" VARCHAR(50) UNIQUE NOT NULL, -- Asset Number
5      "name" VARCHAR(100) NOT NULL,
6      "category" VARCHAR(30) NOT NULL, -- ELEVATOR, FIRE_SYSTEM, HVAC, etc.
7      "model_spec" VARCHAR(100),
8      "space_id" UUID NOT NULL, -- Anchored to 4.1 Space Domain
9      "status" VARCHAR(20) DEFAULT 'OPERATIONAL', -- OPERATIONAL, FAULT, SCRAPPED
10     "purchase_date" DATE,
11     "expiry_date" DATE,
12     "ext_props" JSONB DEFAULT '{}', -- Flexible attributes for different
      equipment types
13     "created_at" TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
14     "updated_at" TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
15 );
16
17 -- 4.3.2 Maintenance Plan (Supporting Entity)
18 CREATE TABLE "ast_maintenance_plan" (

```

```

19     "id" UUID PRIMARY KEY DEFAULT gen_random_uuid(),
20     "equipment_id" UUID REFERENCES "ast_equipment"("id"),
21     "plan_name" VARCHAR(100) NOT NULL,
22     "cron_expression" VARCHAR(50) NOT NULL, -- Scheduling logic (e.g., "0 0 1
    * * ?")
23     "last_run_time" TIMESTAMPTZ,
24     "next_run_time" TIMESTAMPTZ,
25     "is_active" BOOLEAN DEFAULT TRUE,
26     "config" JSONB DEFAULT '{}' -- Specific steps for maintenance
27 );
28
29 -- 4.3.3 Inspection Points (Value Objects for 3.2.3)
30 CREATE TABLE "ast_inspection_point" (
31     "id" UUID PRIMARY KEY DEFAULT gen_random_uuid(),
32     "equipment_id" UUID REFERENCES "ast_equipment"("id"),
33     "point_name" VARCHAR(100) NOT NULL,
34     "check_method" VARCHAR(20), -- QR_CODE, NFC, MANUAL
35     "standard_guide" TEXT -- Operating instructions for staff
36 );
37
38 -- Indexes for performance
39 CREATE INDEX "idx_ast_equipment_space" ON "ast_equipment" ("space_id");
40 CREATE INDEX "idx_ast_equipment_jsonb" ON "ast_equipment" USING GIN
    ("ext_props");

```

4.4 财务计费领域（Finance Domain）

- **职责：**保障资金安全，处理高并发计费与复杂的核销策略。
- **聚合根：** `FinancialBill`（账单）
- **聚合根实体：**
 - `RefundOrder`（退款申请单）：记录退款生命周期。
 - `PaymentTransaction`（支付流水）：记录与支付网关的交互结果。
- **领域服务：**
 - `PricingEngineService`：计费引擎。支持周期性费用（物业费）、表计费用（水电）及公摊算法。
 - `ArrearsValidationService`：**核心逻辑**。执行“欠费抵扣校验”，在发起退款前检索该空间下是否存在未结清账单。
 - `RefundWorkflowService`：管理从申请、BPM 审批到支付网关执行退款的完整链路。
- **领域事件：**

- `BillPaidEvent`：缴费成功。**下游订阅者**：4.5 权限领域（自动解除欠费限制）、移动端（推送收据）。
- `RefundProcessedEvent`：退款入账成功。**下游订阅者**：通知模块（告知业主退款到账）。
- **核心逻辑**：策略版本化（公式存入 JSONB），防止调价导致历史账单变动。
- **数据结构 DDL**：

代码块

```

1  -- 财务账户表
2  CREATE TABLE "fin_billing_account" (
3      "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
4      "space_id" uuid UNIQUE REFERENCES "base_space_node"("id"),
5      "balance" numeric(15, 2) DEFAULT 0.00,
6      "total_debt" numeric(15, 2) DEFAULT 0.00,
7      "update_time" timestamp DEFAULT CURRENT_TIMESTAMP
8  );
9
10 -- 计费策略表
11 CREATE TABLE "fin_fee_policy" (
12     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
13     "item_name" varchar(50) NOT NULL,           -- 物业费, 水费
14     "calc_type" varchar(20) NOT NULL,         -- FIXED, AREA_BASED, TIERED
15     "formula_config" jsonb NOT NULL,         -- 存储阶梯或系数算法
16     "version" int DEFAULT 1
17 );
18
19 -- 账单表
20 CREATE TABLE "fin_bill" (
21     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
22     "account_id" uuid REFERENCES "fin_billing_account"("id"),
23     "policy_id" uuid REFERENCES "fin_fee_policy"("id"),
24     "billing_period" varchar(10) NOT NULL,    -- 2024-01
25     "receivable_amount" numeric(12, 2),
26     "actual_paid_amount" numeric(12, 2) DEFAULT 0,
27     "refund_status" varchar(20) DEFAULT 'NONE', -- 枚举值: NONE(无), REFUNDING(退款中), REFUNDED(已退款), REFUSED(拒退)
28     "status" varchar(20) DEFAULT 'UNPAID',
29     "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
30 );
31
32 -- 退款申请单表 (支持部分退款与全额退款)
33 CREATE TABLE "fin_refund_order" (
34     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(), -- 建议使用
35     "original_bill_id" uuid NOT NULL REFERENCES "fin_bill"("id"),

```

```

36     "refund_no" varchar(32) UNIQUE NOT NULL,           -- 退款单号 (RF + yyyyMMdd +
    随机码)
37     "applicant_id" uuid NOT NULL,                     -- 申请人 (客服/业主)
38     "refund_amount" numeric(12, 2) NOT NULL,         -- 本次申请退款金额
39     "reason" varchar(255) NOT NULL,                  -- 退款原因
40     "bpm_instance_id" varchar(64),                   -- 关联 7.2 的审批流ID
41     "status" varchar(20) DEFAULT 'PENDING',         -- PENDING, APPROVED,
    REJECTED, SUCCESS
42     "refund_time" timestamp,                          -- 实际到账时间
43     "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
44 );
45
46 -- 财务流水审计表 (补充定义, 确保包含逆向流水)
47 CREATE TABLE "fin_transaction_audit" (
48     "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
49     "bill_id" uuid REFERENCES "fin_bill"("id"),
50     "refund_id" uuid REFERENCES "fin_refund_order"("id"), -- 可空, 仅退款时有值
51     "trade_no" varchar(64),                          -- 外部支付/退款流水号 (微信/支
    付宝)
52     "amount" numeric(12, 2) NOT NULL,                -- 正数为收, 负数为退
53     "direction" varchar(10) NOT NULL,                -- INCOME (进), OUTLAY (出)
54     "transaction_type" varchar(20) NOT NULL,         -- PAYMENT, REFUND,
    ADJUST_DEC (调减)
55     "audit_time" timestamp DEFAULT CURRENT_TIMESTAMP
56 );

```

4.5 集成中心领域 (Data Hub Domain)

- **职责:** 系统的“防腐层”，解析第三方（停车、IoT、政府）异构报文。
- **聚合根:** `IntegrationLog` (集成审计日志)
- **领域服务:**
 - `OfflineSyncService`: **核心逻辑**。处理移动端 (3.9) 离线数据的“补传与合并”。执行时间戳仲裁，防止旧数据覆盖新数据。
 - `ProtocolTransformer`: 协议转换器。将外部系统的原始报文 (XML/自定义格式) 转化为内部标准领域模型。
 - `IdempotentConsumer`: 幂等消费服务。通过 Redis 确保同一外部通知 (如支付成功回调) 仅处理一次。
- **领域事件:**
 - `ExternalDataReceivedEvent`: 接收到外部数据 (如 962121 工单)。**下游订阅者:** 4.2 运营领域 (自动转化为内部报修工单)。
- **核心逻辑:** 先存后洗。保留原始 JSONB 报文以备追溯。

代码块

```
1  -- 集成中心审计与原始报文表 (整合物理实现与领域语义)
2  CREATE TABLE "int_external_log" (
3    "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
4    "system_source" varchar(50) NOT NULL,    -- 来源系统: PARKING, PAYMENT,
GOV_962121, IOT
5    "event_type" varchar(64),                -- 事件类型: PAY_CALLBACK,
DEVICE_ALARM, ORDER_SYNC
6    "trace_id" varchar(64),                  -- 全链路追踪ID, 用于与业务表关联
7    "raw_payload" jsonb NOT NULL,           -- 原始请求报文 (原 raw_payload)
8    "raw_response" jsonb,                   -- 平台给出的原始响应报文
9    "process_status" varchar(20) DEFAULT 'PENDING', -- PENDING, SUCCESS, FAILED
10   "error_msg" text,                         -- 失败时的堆栈或原因描述
11   "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
12 );
13
14 -- 高效索引设计
15 -- GIN 索引用于在海量原始 JSON 报文中快速检索特定的业务标识 (如: 车牌号、第三方单号)
16 CREATE INDEX "idx_int_log_raw_gin" ON "int_external_log" USING GIN
("raw_payload");
17 -- 追踪 ID 索引用于跨系统联调
18 CREATE INDEX "idx_int_log_trace" ON "int_external_log" ("trace_id");
```

4.6 权限与账户领域 (IAM Domain)

- **职责:** 身份认证与多项目权限隔离。
- **聚合根:** `Account` (账户)、`PermissionProfile` (权限档案)
- **领域服务:**
 - `ProjectContextFilter`: 动态拦截器, 确保所有 SQL 查询强制带上当前项目的隔离标识。
 - `RoleHierarchyService`: 管理集团、项目、岗位三级权限继承逻辑。
 - `VisitorCredentialService`:
 - **生成逻辑:** 基于 `invite_id` + `timestamp` + `salt` 生成高强度的加密混淆字符串 (用于生成二维码)。
 - **校验逻辑:** 后端接收到闸机上传的码值后, 解析时间戳和合法性, 确保二维码不可伪造且具备动态时效。
- **领域事件:**


- `AccountRestrictedEvent`：账户因欠费或违规被限制。**下游动作**：下发指令给门禁硬件，取消特定区域通行权。
- `VisitorAuthorizedEvent`：预约成功并生成凭证后抛出。**下游订阅者**：4.6 设备领域（执行下发逻辑）。
- **核心逻辑**：通过 `user_project_role` 实现一个用户在不同小区拥有不同身份的精细化权限。
- **数据结构 DDL**：

代码块

```

1  -- 用户统一账号表
2  CREATE TABLE "iam_user" (
3    "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
4    "username" varchar(50) UNIQUE NOT NULL,
5    "password_hash" varchar(255) NOT NULL,
6    "user_type" varchar(20) NOT NULL,    -- STAFF, OWNER
7    "is_enabled" boolean DEFAULT true,
8    "create_time" timestamp DEFAULT CURRENT_TIMESTAMP
9  );
10
11 -- 用户-项目-角色关联表 (核心权限纽带)
12 CREATE TABLE "iam_user_project_role" (
13   "id" uuid PRIMARY KEY DEFAULT gen_random_uuid(),
14   "user_id" uuid REFERENCES "iam_user"("id"),
15   "project_id" uuid REFERENCES "base_space_node"("id"), -- 指向
    node_type='PROJECT' 的节点
16   "role_id" uuid NOT NULL,
17   UNIQUE("user_id", "project_id", "role_id")
18  );

```

 **领域模型详细设计文档见：** [📖 领域模型详细设计](#)

第五章： 业务流程串联检查（Business Flow Audit）

本章定义了跨领域的深度联动场景。开发时，必须确保这些流程在事务、状态和领域事件上实现完全闭环。

5.1 运营-财务-空间：欠费管控闭环（Arrears Control Loop）

场景描述：当住户产生逾期欠费时，系统需自动联动空间权限限制，并在缴清后实时恢复。

- **逻辑链路**：
 - **触发**：`fin_bill` 状态变为 `OVERDUE`（逾期）。

- **判定：**系统检查该 `space_id` 下所有未结清账单，若金额超过阈值或时间超过 90 天。
- **限制：**调用 `IAM` 服务，将关联业主的 `is_restricted` 标记设为 `true`（限制其申请公共区域预约、在线开门等非基础服务）。
- **恢复：**当账单状态变为 `PAID`（已结清），触发 `BillPaidEvent` 领域事件，系统自动撤销权限限制。

5.2 物联-运营-资产：设备故障自愈闭环（IoT-Ops-Asset Loop）

场景描述：传感器监测到硬件故障，自动触发维修流程并更新资产健康档案。

- **逻辑链路：**
 - **感知：**`pjl-backend-hub` 接收设备异常报文，状态置为 `RECEIVED`。
 - **联动：**`base_equipment.status` 自动变更为 `FAULT`（故障待修）。
 - **派单：**系统根据设备所属空间和维护组，自动创建 `op_work_order`，初始状态为 `PENDING`。
 - **归档：**工单状态变为 `CLOSED` 后，设备状态同步恢复为 `NORMAL`，并自动在 `base_equipment.ext_props` 写入最后维修日期与执行人。

5.3 审批-财务：逆向调账审计闭环（BPM-Finance Adjustment Loop）

场景描述：针对已入账的错误账单，通过审批流进行受控的“红冲”或“减免”处理。

- **逻辑链路：**
 - **申请：**客服发起调账申请，关联具体的 `fin_bill_id`。
 - **锁定：**该账单在数据库中被标记为 `LOCKED`（禁止在此期间进行任何支付操作）。
 - **审批：**`proc_instance` 状态流转。若 `REJECTED`，账单解锁；若 `APPROVED`，触发回调。
 - **核销：**系统自动将原账单置为 `ADJUSTED`（已调账），并生成一条等额负值的 `fin_transaction_audit` 审计流水，确保财务平账。

5.4 空间-权限：入驻/退租全流程闭环（Space-IAM Life Cycle）

场景描述：房屋状态变更时，系统自动清理或授予对应的数字通行权限。

- **逻辑链路：**
 - **变更：**`base_room_detail.house_status` 变更为 `OCCUPIED`（已入住）。
 - **授权：**系统自动激活该业主在 `base_ownership` 中的认证状态，下发蓝牙门禁权限。
 - **注销：**当房屋状态变为 `VACANT`（空置）或办理退租时，系统必须强制调用 `IAM` 服务注销该空间下所有非业主类型的临时通行证，确保物理安全闭环。

5.5 资产-任务：生命周期强制阻断（Asset Termination Trigger）

技术实现：通过 PostgreSQL 触发器实现“报废即终止”的强一致性逻辑。

代码块

```
1  -- Implementation of Technical Spec Sec 5.5
2  CREATE OR REPLACE FUNCTION func_terminate_asset_tasks()
3  RETURNS TRIGGER AS $$
4  BEGIN
5      -- Only trigger when status changes to SCRAPPED or OUT_OF_SERVICE
6      IF (NEW.status <> OLD.status) AND (NEW.status IN ('SCRAPPED',
7      'OUT_OF_SERVICE')) THEN
8
9          -- 1. Terminate associated Inspection Tasks (Domain 4.2)
10         UPDATE "op_inspection_task"
11         SET "status" = 'TERMINATED',
12             "result_data" = jsonb_set(COALESCE("result_data", '{}'::jsonb),
13             '{termination_reason}', '"Asset Scrapped"')
14         WHERE "target_equipment_id" = NEW.id AND "status" = 'PENDING';
15
16         -- 2. Force Close associated Work Orders (Domain 4.2)
17         UPDATE "op_work_order"
18         SET "status" = 'CLOSED',
19             "close_note" = 'System auto-close: Asset scrapped'
20         WHERE "equipment_id" = NEW.id AND "status" NOT IN ('CLOSED',
21         'ARCHIVED');
22
23     END IF;
24     RETURN NEW;
25 END;
26 $$ LANGUAGE plpgsql;
27
28 CREATE TRIGGER trg_asset_scrapped_termination
29 AFTER UPDATE ON "ast_equipment"
30 FOR EACH ROW EXECUTE FUNCTION func_terminate_asset_tasks();
```

5.6 预约-权限：访客通行闭环流程

- **场景描述：**业主发起预约 -> 生成凭证 -> 硬件放行 -> 实时提醒。
- **逻辑链路：**
 - **触发：**业主在小程序提交预约信息（写入 `op_visitor_invite`）。
 - **授权：**4.5 权限领域 生成加密二维码，返回给 C 端展示。

- **分发：4.6 设备领域** 订阅 `VisitorAuthorizedEvent`，通过集成中心（Hub）向闸机下发临时白名单。
- **核验：**访客到场扫码，闸机回调 **4.4 集成中心**。
- **反馈：**集成中心确认通行后，更新 `op_visitor_invite` 状态为 `COMPLETED`，并触发微信订阅消息提醒业主。