空间资产数据管理平台优化方案设计文档

1. 项目背景

空间资产数据管理平台目前主要实现了建筑信息管理、空间分区管理、设备设施管理和运维管理等核心功能。为了提升系统性能、可用性和用户体验，需要在现有基础上进行技术优化。

2. 现状分析

2.1 当前系统特点

- 采用 Spring Boot + JPA 的技术架构
- 实现了基础的 CRUD 功能
- 具备基本的权限控制
- 支持空间资产的多层级管理

2.2 存在的问题

- 频繁访问数据库导致性能瓶颈
- 缺乏实时通知机制
- 系统可扩展性有限
- 缺少性能监控和问题追踪能力

3. 优化方案

3.1 缓存层优化

业务场景

1. **建筑信息缓存**
    - 建筑基础信息变更频率低但查询频繁
    - 多个业务模块都需要访问建筑信息
    - 涉及楼层、房间等多级数据查询
2. **权限数据缓存**

- 用户权限数据在每次操作时都需要验证
- 权限数据相对稳定，变更不频繁
- 影响系统整体响应速度

实现策略

- 采用 Redis 实现分布式缓存
- 设置合理的缓存过期策略
- 实现缓存预热机制
- 建立缓存更新机制

3.2 消息队列集成

业务场景

1. **设备维护通知**
   - 设备需要维护时通知相关人员
   - 需要通过多个渠道发送通知
   - 通知发送不应影响主业务流程

2.现场审核通知

若房间实际用途和设计用途不一致需要通知相关人员（可选择通知哪一部门）

3.设备状态监控

- 需要进行数据分析和异常检测
- 设备到期需要进行通知
- 支持高并发数据处理

实现策略

- 使用 RabbitMQ 实现消息队列
- 设计消息优先级机制
- 实现消息重试和死信队列
- 建立消息追踪机制

3.3 实时通知机制

业务场景

1. **运维任务通知**
    - 维护人员需要实时接收新任务
    - 设备异常需要及时通知
    - 任务状态变更实时同步
2. **设备告警推送**
    - 设备异常实时推送
    - 告警级别分类处理
    - 支持告警确认和处理

实现策略

- 采用 WebSocket 实现实时通信
- 设计消息推送机制
- 实现消息分发策略
- 建立连接保活机制

3.4 分布式锁实现

业务场景

1. **设备维护状态管理**
    - 防止设备重复维护
    - 确保维护状态一致性
    - 避免资源争用
2. **空间资产状态变更**
    - 确保资产状态变更的原子性
    - 防止并发操作导致的数据不一致
    - 支持事务性操作

实现策略

- 使用 Redis 实现分布式锁
- 设计锁超时机制

- 实现锁重试策略
- 建立死锁预防机制

3.5 链路追踪实现

业务场景

1. **资产变更追踪**
    - 追踪空间资产的变更操作
    - 记录操作人和变更内容
    - 支持变更历史查询
2. **运维操作追踪**
    - 记录设备维护过程
    - 追踪任务执行状态
    - 支持问题定位和分析

实现策略

- 集成 Spring Cloud Sleuth
- 实现统一的链路 ID
- 设计追踪日志格式
- 建立追踪数据存储机制

3.6 性能监控体系

业务场景

1. **接口性能监控**
    - 监控资产查询性能
    - 监控设备状态查询
    - 识别性能瓶颈
2. **系统资源监控**
    - 监控系统负载
    - 监控资源使用情况

- 预警阈值设置

实现策略

- 集成 Spring Boot Actuator
- 使用 Prometheus 收集指标
- 通过 Grafana 展示监控数据
- 建立告警机制

4. 实施建议

4.1 优先级排序

1. 缓存层优化（提升系统响应速度）
2. 实时通知机制（提升用户体验）
3. 消息队列集成（提升系统可扩展性）
4. 分布式锁实现（确保数据一致性）
5. 性能监控体系（支持运维管理）
6. 链路追踪实现（便于问题定位）

4.2 实施步骤

1. **第一阶段**：缓存层和实时通知
2. **第二阶段**：消息队列和分布式锁
3. **第三阶段**：性能监控和链路追踪

4.3 风险控制

- 进行充分的测试验证
- 制定回滚策略
- 准备应急预案
- 做好监控告警

5. 预期效果

- 系统响应时间提升 50%
- 实时通知延迟控制在 1 秒内
- 系统可用性提升到 99.9%
- 问题定位时间缩短 60%
- 运维效率提升 40%

6. 后续规划

- 持续优化缓存策略
- 完善监控指标
- 优化告警规则
- 扩展业务场景