电梯调度系统设计文档

本系统是一个基于`Java`的电梯调度系统，模拟了20层楼5部互联电梯的运行。系统采用LOOK算法实现高效的电梯调度，并提供直观的图形界面展示电梯运行状态。一些基础介绍可见`README.md`文档。

一、系统整体架构设计

架构分层模型

本系统采用分层架构设计，将功能模块划分为四个核心层次：

1. **用户交互层（GUI）**

- **功能定位**：提供可视化操作界面，实时展示电梯运行状态
- **核心组件**：
  - 电梯井道动态可视化面板
  - 楼层外部呼叫按钮矩阵
  - 电梯内部控制面板（含报警装置）
  - 实时运行日志窗口
- **技术实现**：基于Swing框架构建，通过自定义ElevatorVisualizer组件实现电梯运动动画，采用双缓冲技术优化渲染性能

2. **调度控制层**

- **功能定位**：处理用户输入事件，协调多电梯协作
- **核心机制**：
  - 请求队列管理：维护楼层外部请求和电梯内部请求
  - 动态权重分配：基于LOOK算法计算电梯调度优先级
  - 异常处理中枢：监控电梯报警状态，实现故障隔离
- **技术特性**：采用生产者-消费者模式处理请求，保证线程安全

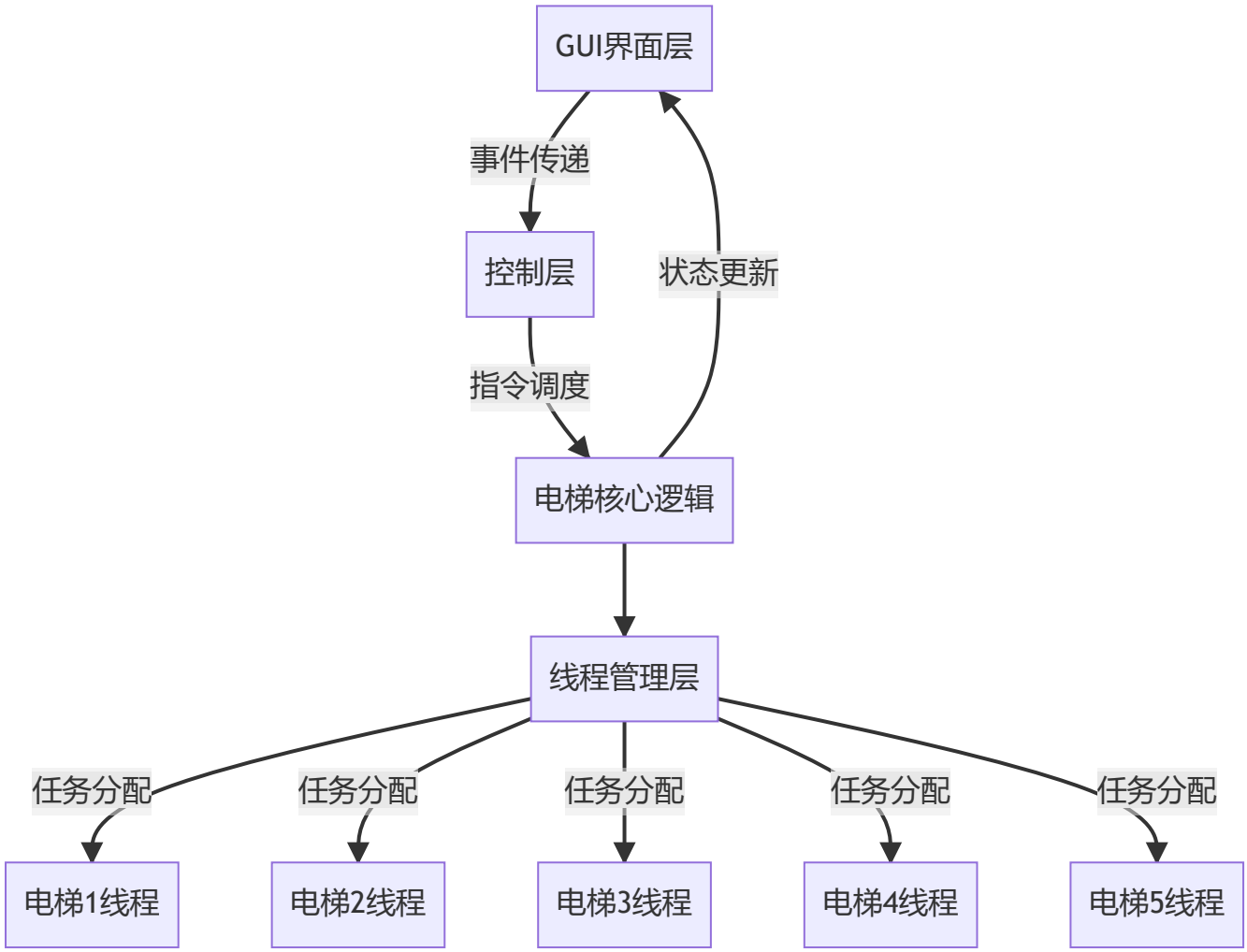
3. **电梯实体层**

- **功能定位**：实现单部电梯的物理行为模拟
- **状态模型**：
  - 移动状态（上行/下行/停止）

    - 门控状态（开门中/已开启/关闭中）
    - 报警状态（正常/紧急停止）
- **行为特征**：
    - 每层移动耗时500ms模拟
    - 开关门动画效果实现
    - 紧急制动响应机制

4. **线程管理层**

- **功能定位**：管理系统线程资源，确保并发安全
- **核心组成**：
    - 固定大小线程池（5个电梯线程）
    - 事件分发线程（EDT）管理UI更新
    - 定时任务线程（状态轮询）
- **同步机制**：
    - ReentrantLock保护关键资源
    - Atomic变量保证状态可见性
    - BlockingQueue实现请求缓冲



核心模块组成

1. **电梯调度系统（ElevatorSystem）**

- **架构角色**：系统大脑，协调多电梯运作
- **核心功能**：
  - 接收并分配楼层呼叫请求
  - 监控各电梯运行状态
  - 实现电梯间请求状态同步
  - 处理全局异常状态
- **工作流程**：
  1. 接收楼层按钮或电梯内部按钮信号
  2. 过滤无效请求（如重复按层）
  3. 根据LOOK算法选择最优电梯
  4. 更新目标电梯的请求队列
  5. 监控请求完成状态

2. **电梯实体（Elevator）**

- **状态管理**：
  - 当前楼层实时追踪
  - 运行方向动态调整
  - 门控状态机转换
- **行为逻辑**：
  - 自动响应最近请求
  - 方向改变决策机制
  - 紧急制动处理流程
- **线程模型**：
  - 独立运行线程处理移动逻辑
  - 异步任务处理开关门动画
  - 定时轮询请求队列（100ms间隔）

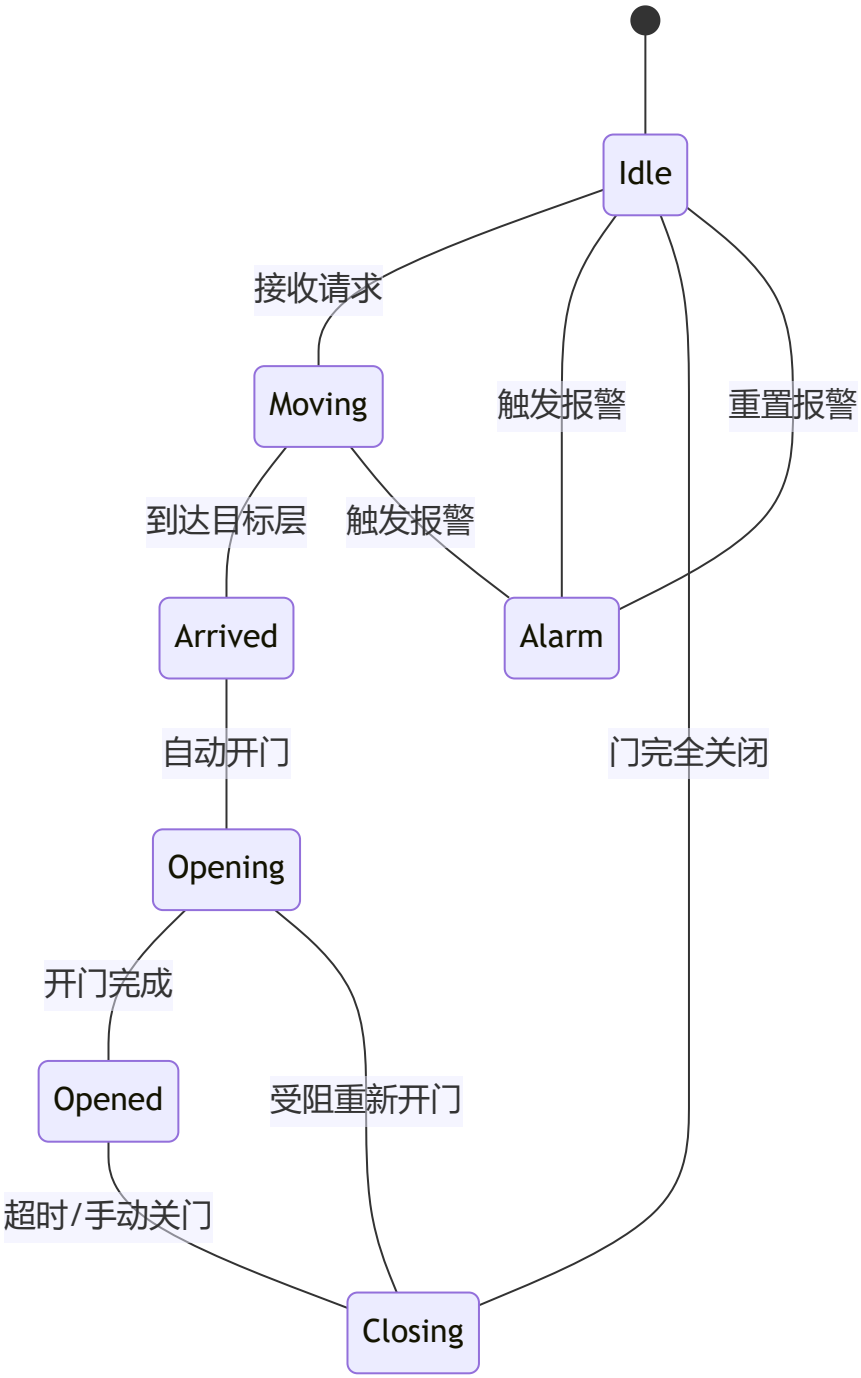
3. **可视化界面（ElevatorGUI）**

- **交互设计**：
  - 20层楼外部呼叫面板（左侧）
  - 5部电梯三维可视化面板（中部）
  - 实时日志显示区域（右侧）
- **动态绑定**：
  - 电梯位置与楼层刻度同步
  - 按钮状态与请求队列联动
  - 报警状态颜色警示机制
- **性能优化**：
  - Swing Timer控制刷新频率（100ms）
  - 局部重绘降低渲染开销
  - 事件合并处理避免界面卡顿

模块名称	对应类	主要职责
电梯实体模块	Elevator	单部电梯状态管理、物理运动模拟
调度中枢模块	ElevatorSystem	全局请求分配、电梯协同调度
楼层系统模块	Floor	楼层按钮状态管理、外部请求生成
可视化模块	ElevatorGUI	图形界面展示、用户交互处理
线程管理模块	ExecutorService	电梯线程池管理、资源分配
紧急处理模块	Elevator(alarm相关)	报警状态管理、紧急停止机制

二、核心功能设计与实现

1. **电梯运行状态机**



2. LOOK调度算法实现

1. 算法核心思想

- **扫描方向保持**：电梯沿当前移动方向持续服务请求
- **智能转向机制**：当前方向无请求时立即反转
- **动态权重评估**：综合距离、方向、负载等多维度评分

2. 决策流程

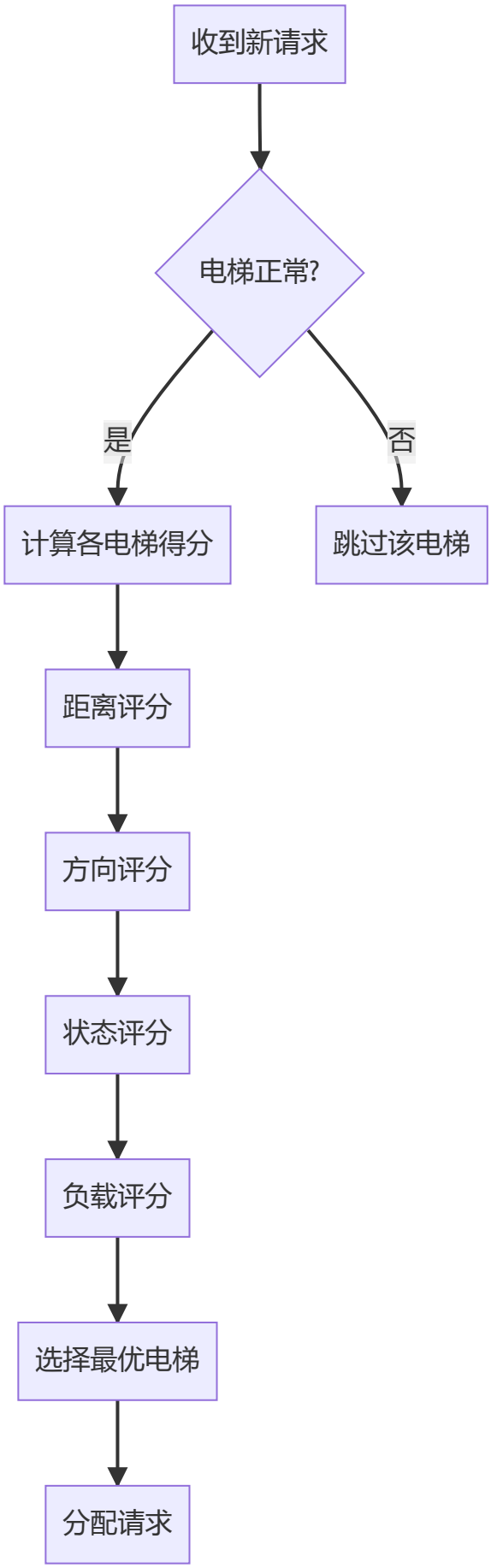
1. **请求接收**：捕获楼层外部呼叫或电梯内部选层

2. **候选筛选**：排除报警电梯，建立可用电梯列表
3. **多维评分**：
    - **基础距离分**（50%权重）：|当前层-目标层|
    - **方向匹配分**（30%权重）：同向请求优先
    - **运行状态分**（15%权重）：停止电梯响应更快
    - **负载压力分**（5%权重）：请求数较少优先
4. **最优选择**：总分最低电梯获得任务分配

3. **算法优势**

- **高效率**：减少空驶时间
- **公平性**：动态权重防止某部电梯过载
- **弹性扩展**：评分权重可配置适应不同场景

算法流程图：

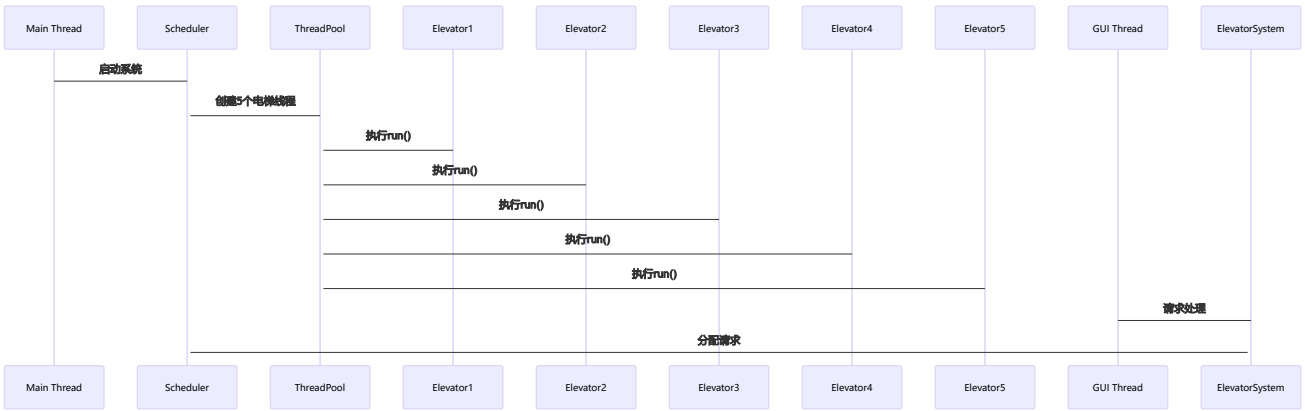


核心评分逻辑：

```
private int calculateLOOKScore(Elevator elevator,
                               int reqFloor,
                               Direction reqDir) {
    // 基础距离分（50%）
    int distanceScore = Math.abs(elevator.currentFloor - reqFloor) * 2;

    // 方向匹配分（30%）
    if (elevator.direction == reqDir) {
        distanceScore -= 10;
    }

    // 状态分（15%）
    switch(elevator.state) {
        case STOPPED: distanceScore -= 2; break;
        case MOVING: distanceScore += 2; break;
    }

    // 负载分（5%）
    distanceScore += elevator.requests.size() * 3;

    return distanceScore;
}
```

三、线程模型与并发控制

1. 线程架构



2. 并发架构模型

```
主线程
├── GUI事件分发线程（EDT）
├── 调度管理线程
└── 电梯线程池（5个）
    ├── 电梯1#移动线程
    ├── 电梯2#移动线程
    ├── ...
    └── 电梯5#移动线程`
```

3. 关键同步机制

- **电梯状态同步**：采用原子更新

```
// 原子状态更新
private void updateElevatorStatus() {
  synchronized(statusLock) {
      currentFloor = targetFloor;
      direction = newDirection;
  }
}
```

- **状态更新同步**：通过SwingUtilities保证UI线程安全

```
public static void logMessage(String message) {
  SwingUtilities.invokeLater(() -> {
      logTextArea.append(message + "\n");
  });
}
```

- **资源竞争控制**：电梯内部锁（ReentrantLock）保护请求队列

```
// 使用ReentrantLock保证线程安全
public void pressFloorButton(int floor) {
  lock.lock();
  try {
      if (!requestedFloors.contains(floor)) {
          requestedFloors.add(floor);
      }
  } finally {
      lock.unlock();
  }
}
```

4. 线程通信设计

- **事件驱动**：按钮动作触发Observer模式通知
- **状态推送**：电梯定期发布运行状态（100ms间隔）
- **异常传播**：报警状态通过EventBus广播

5. 死锁预防策略

- 锁获取设置超时时间（tryLock）
- 统一资源申请顺序（楼层->电梯）
- 采用无锁数据结构（CopyOnWriteArrayList）

四、GUI与业务逻辑交互

1. 界面元素绑定

2. 可视化组件

界面元素	对应类/方法	数据绑定机制
电梯位置显示	ElevatorVisualizer	paintComponent()重绘
状态标签	elevatorStatusLabels	Timer定时更新
楼层按钮	externalButtons[][]	ActionListener事件绑定
报警指示	alarmButton	背景色动态变化
运行日志	logTextArea	静态logMessage()方法