电梯调度系统文档

项目概述

这是一个基于 Python Flask 和 WebSocket 的多电梯调度系统模拟器。系统实现了 5 部电梯的并行运行，支持楼层内外呼叫以及电梯的基本运行控制。

技术栈

- 后端：Python Flask + Flask-SocketIO
- 前端：原生 JavaScript + WebSocket
- UI：HTML + CSS

系统架构

后端架构

1. **核心组件**
    - Flask 应用服务器
    - WebSocket 实时通信
    - 多线程电梯控制系统
2. **状态管理**

```
elevator_goal = [set() for _ in range(5)]  # 每个电梯的目标楼层
should_sleep = [0] * 10  # 电梯开门状态
state = [0] * 5  # 电梯运行状态: 0停止, 1向上, -1向下
pause = [1] * 5  # 电梯暂停状态
floor = [1] * 5  # 当前楼层
people_up = set()  # 向上请求
people_down = set()  # 向下请求
```

3. **主要功能模块**
    - `check_elevator()`: 电梯状态检查与更新
    - `update_elevator_state()`: 电梯运行状态更新
    - `ElevatorThread`: 电梯独立运行线程
    - WebSocket 事件处理器：处理各类请求

前端架构

4. **界面布局**
    - 电梯组显示区

    - 走廊呼叫按钮区
    - 实时状态显示
5. **交互功能**
    - 电梯内部按钮控制
    - 楼层外部上下行呼叫
    - 电梯运行状态显示
    - 电梯门状态显示

功能特性

1. 电梯调度算法
- 采用最近电梯优先策略
- 支持动态调整运行方向
- 智能处理同向楼层请求

2. 实时控制
- 电梯暂停/继续功能
- 实时状态反馈
- 电梯门控制模拟

3. 用户界面
- 响应式布局设计
- 直观的电梯状态显示
- 简洁的操作界面

系统架构

电梯调度算法详解

1. 基本调度策略
- **最近电梯优先**
    - 计算所有电梯与请求楼层的距离：`distances = [abs(floor[i] - floor_num) for i in range(5)]`
    - 选择距离最小的电梯响应请求：`nearest_elevator = distances.index(min(distances))`
    - 避免电梯频繁改变方向，提高运行效率

2. 状态管理机制

- **电梯状态定义**

```
state = [0] * 5  # 0:停止, 1:向上, -1:向下
```

- **目标楼层集合**
    - 使用集合数据结构存储目标楼层
    - 自动去重，避免重复请求
    - 分别维护内部请求和外部请求

3. 运行状态更新逻辑

```
def update_elevator_state(elevator_id):
    idx = elevator_id - 1
    goals = list(elevator_goal[idx])

    if not goals:
        state[idx] = 0  # 无请求时停止
        return

    # 向下运行时的状态更新
    if state[idx] == -1:
        if min(goals) > floor[idx]:
            state[idx] = 1  # 所有目标楼层都在上方时改变方向

    # 向上运行时的状态更新
    elif state[idx] == 1:
        if max(goals) < floor[idx]:
            state[idx] = -1  # 所有目标楼层都在下方时改变方向

    # 停止状态时的方向选择
    else:
        if max(goals) > floor[idx]:
            state[idx] = 1  # 有更高楼层的请求时向上
        elif min(goals) < floor[idx]:
            state[idx] = -1  # 有更低楼层的请求时向下
```

4. 电梯响应机制

- **内部请求处理**
    - 直接将目标楼层加入对应电梯的目标集合
    - 实时更新电梯运行状态
- **外部请求处理**
    - 分别维护向上(`people_up`)和向下(`people_down`)请求集合
    - 根据距离选择最近的电梯响应

- 考虑电梯当前运行方向，优化分配策略

5. 开门控制逻辑

```
# 到达目标楼层时的开门逻辑
if current_state == 1 and (current_floor in elevator_goal[elevator_id - 1] or
current_floor in people_up):
    should_sleep[elevator_id - 1] = 1  # 设置开门状态
    # 清理已完成的请求
    people_up.discard(current_floor)
    elevator_goal[elevator_id - 1].discard(current_floor)
```

6. 性能优化策略

- **请求合并**：同方向的请求优先由同一电梯处理
- **状态持续性**：避免电梯频繁改变运行方向
- **实时响应**：使用 WebSocket 保证状态更新的实时性
- **并发控制**：采用多线程确保电梯独立运行

7. 安全保护机制

- 楼层边界检查
- 电梯运行状态实时监控
- 异常状态自动处理
- 支持手动暂停/继续操作

安装与运行

环境要求

- Python 3.x
- Flask
- Flask-SocketIO

安装步骤

1. 安装依赖：

```
pip install flask flask-socketio
```

2. 运行应用：

```
python main.py
```

3. 访问系统：
   在浏览器中打开 `http://localhost:5000`

API 接口说明

WebSocket 事件

1. `set_goal`
    - 功能：处理电梯内部按钮请求
    - 参数：
        - elevator: 电梯编号
        - floor: 目标楼层
2. `set_up_request`
    - 功能：处理向上请求
    - 参数：
        - floor: 请求楼层
3. `set_down_request`
    - 功能：处理向下请求
    - 参数：
        - floor: 请求楼层
4. `toggle_pause`
    - 功能：处理暂停/启动请求
    - 参数：
        - elevator: 电梯编号

项目结构

```
ElevatorSystem/
├── main.py           # 主程序文件
└── templates/        # 前端模板
    └── index.html    # 主页面
```

注意事项

5. 运行时请确保 5000 端口未被占用
6. 调试模式下运行可能会导致 WebSocket 连接不稳定
7. 系统当前支持最高 20 层楼的调度