智能家居管理系统设计文档

1. 项目背景

智能家居管理系统的目标是为用户提供一个简洁、高效、可扩展的平台，使他们能够方便地管理家中的各种智能设备。

系统需要处理用户管理、设备控制、日志记录、配置管理、并发任务调度等多个方面的功能，并能够在多线程环境下高效运行。

2. 项目目标

开发一个功能完备的智能家居管理系统，具有以下主要功能：

1. **用户管理**：支持用户的注册、登录、权限控制等。
2. **设备管理**：支持智能设备的添加、删除、控制和状态查看。
3. **日志管理**：支持日志的记录、查询和存档。
4. **配置管理**：系统配置的读取、修改与持久化。
5. **数据库支持**：用户、设备、日志信息的存储与管理。
6. **多线程任务调度**：通过线程池管理多任务，确保并发任务的高效处理。
7. **异常处理**：系统能够捕获并处理异常，确保系统稳定运行。

3. 功能需求

3.1 用户管理功能

1. 用户注册与登录：
    - 用户可以通过用户名和密码进行注册与登录。
    - 使用**加密存储**密码（如通过 `SHA256` 或 `bcrypt` 加密）。
    - 登录时验证凭证，失败时返回错误信息，限制登录失败次数。
2. 用户角色与权限管理：
    - 管理员角色：
        - 可以查看所有设备的状态、控制所有设备、查看所有日志、管理用户（增、删、改）。
        - 支持设备添加、删除、更新及配置修改。
    - 普通用户角色：
        - 可以控制已授权的设备（设备控制权限可以在用户注册时配置）。
        - 普通用户不能管理其他用户。
        - 支持查询设备状态和控制设备。
3. 会话管理：
    - 系统记录每个用户的登录时间、`IP` 地址等信息，并提供会话管理功能。
    - 会话过期策略：用户未操作时，系统会自动退出。

3.2 设备管理功能

1. 设备支持：
    - 设备类型：
        - **灯具**：支持开关、亮度调节（0-100%）。
        - **温控器**：支持温度设置、当前温度显示。
        - **摄像头**：支持开启、关闭，支持分辨率调整。
        - **插座**：支持开关控制。
    - 新设备类型通过**工厂模式**（Factory Pattern）动态添加，支持设备类型的扩展。
2. 设备控制与状态查询：
    - 每个设备有状态属性（如是否开启，当前属性值等），并提供控制接口。
    - 支持设备开关操作（例如开/关灯）及其他控制操作，设备状态变更时更新数据库。
3. 设备管理：
    - 管理员可以添加、删除、修改设备。
    - 系统提供设备列表，管理员可以查看所有设备及其状态。
    - 支持设备批量操作，如批量启用、禁用设备。
4. 设备状态与历史记录：
    - 设备状态实时更新并保存到数据库。
    - 用户可以查看设备操作历史，包括设备的开关记录、温度调节记录等。

3.3 日志管理功能

1. 日志类型：
    - **用户登录/注销日志**：记录用户的登录、登出时间、IP 地址。
    - **设备操作日志**：记录设备的操作（如开/关设备、调节亮度、设置温度等）。
    - **系统异常日志**：记录系统运行中遇到的异常（如数据库操作失败、设备无法响应等）。
2. 日志存储与滚动输出：
    - 所有日志将存储在指定文件夹中，日志文件采用 `.txt` 格式。
    - 每个日志文件最大 10MB，当文件大小超过此限制时，日志文件将自动滚动：
        - 旧日志重命名为备份文件，例如：`log_1.txt`、`log_2.txt` 等。
        - 系统创建新的日志文件（如 `log.txt`）继续写入。
    - 滚动文件的最大数量为 5 个，超过 5 个备份文件时，最旧的备份将被删除。
3. 线程安全日志：
    - 为了确保多线程环境下日志记录的安全性，日志记录操作使用 `std::mutex` 进行互斥处理。
    - 为避免频繁的磁盘写入操作，日志将被缓存到队列中（如 `std::queue`），并定期或在队列达到一定大小时写入文件。
    - 使用文件流（`std::ofstream`）写入 `.txt` 格式日志文件。

3.4 配置管理功能

1. 配置文件读取与保存：
    - 系统配置包括数据库连接信息、设备初始化数据等。
    - 配置文件支持 JSON 格式，便于用户修改和扩展。
    - 配置文件可指定设备初始状态、用户角色等。
2. 配置动态更新：

- 支持在系统运行时动态更新配置（如设备的控制权限、日志存储路径等），无需重启系统。

3. 设备状态保存：
   - 系统关闭时，保存所有设备的当前状态（如开关状态、设置的温度、亮度等）到配置文件。

3.5 数据库管理功能

1. 数据库设计：
   - 使用 SQLite3 数据库。
   - 数据表设计：
     - **用户表**：包含 `id`（主键）、`username`、`password_hash`、`role`。
     - **设备表**：包含 `id`（主键）、`device_type`、`status`（如开/关、亮度等属性）、`last_modified`（更新时间戳）。
     - **日志表**：包含 `id`（主键）、`log_type`、`timestamp`、`user_id`、`device_id`、`log_content`。
2. 数据库操作：
   - 提供增、删、改、查操作，支持高效查询（如按时间查询日志、按设备查询历史操作）。

3.6 多线程任务调度功能

1. 线程池设计：
   - 使用线程池管理任务。通过`std::thread`和`std::mutex`管理线程，确保线程池能够复用线程，减少线程创建销毁的开销。
   - 任务加入线程池后，线程池将选择空闲线程执行任务，避免频繁的线程切换。
2. 任务调度：
   - 支持异步执行设备控制操作（如开关设备、调节亮度等），确保主线程不被阻塞。
   - 支持日志记录、设备状态更新等任务的异步执行，保证系统响应能力。
3. 线程安全的日志记录：
   - 在多线程环境下，通过 `std::mutex` 或 `std::lock_guard` 确保日志的线程安全。

3.7 异常处理功能

1. 全局异常捕获：
   - 使用异常处理机制（`try-catch`）捕获异常并记录详细的错误信息到日志中。
   - 捕获所有关键操作（如数据库操作、设备控制、配置文件读取等）中的异常。

4. 系统架构设计

4.1 模块划分

- **用户管理模块**：负责用户的注册、登录、会话管理和权限控制。
- **设备管理模块**：负责设备的增、删、改、查，设备控制与状态更新。
- **日志管理模块**：负责记录、存储和查询日志，确保日志的线程安全。
- **配置管理模块**：负责配置文件的读取、写入、更新与设备状态的保存。
- **数据库管理模块**：负责数据库的连接、操作和事务管理。
- **任务调度模块**：通过线程池管理任务调度与并发执行。
- **异常处理模块**：负责捕获和记录系统运行中的异常。

4.2 数据库设计

- 用户表：
  - `id` INT PRIMARY KEY
  - `username` VARCHAR(255)
  - `password_hash` VARCHAR(255)
  - `role` VARCHAR(50)
- 设备表：
  - `id` INT PRIMARY KEY
  - `device_type` VARCHAR(100)
  - `status` TEXT
  - `last_modified` TIMESTAMP
- 日志表：
  - `id` INT PRIMARY KEY
  - `log_type` VARCHAR(50)
  - `timestamp` TIMESTAMP
  - `user_id` INT
  - `device_id` INT
  - `log_content` TEXT

5. 技术栈

- **编程语言**：C++ (C++14, C++17 或 C++20)
- **数据库**：SQLite3
- **线程池管理**：使用 `std::thread` 和 `std::mutex`
- **配置文件解析**：`nlohmann/json` 库
- **日志管理**：`std::ofstream`, `std::mutex`
- **版本控制**：Git

6. 使用说明

1. 启动系统时，首先检查配置文件 `config.json` 中的设备初始化数据。
2. 用户登录后，可以管理已授权的设备、查看日志、修改系统配置等。
3. 系统异常会被自动捕捉，并记录详细日志。