ScheduleManager项目设计文档

1. 小组成员与分工

姓名	邮箱	分工
王桢锐	blizzard-wang@sjtu.edu.cn	负责命令行版本的实现
何昊聪	haocong-he@sjtu.edu.cn	负责Web版本的实现

2. 命令行版本设计

2.1 功能概述与创新点

本项目除了实现了所有要求的功能外，还具有以下功能与创新点：

1. 实现了密码盲输，即用户输入密码时，密码不会显示在屏幕上，以提升安全性。
2. 实现了用户修改密码的功能。
3. 对用户的输入判断全面，鲁棒性强。

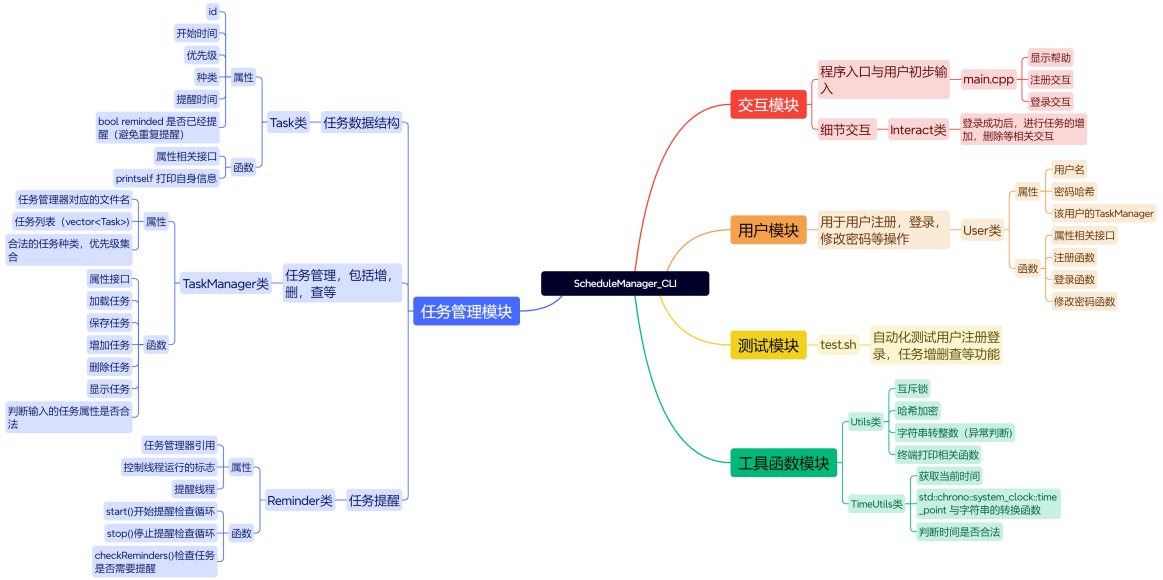
2.2 模块与类的设计

命令行版本的总体设计如下：

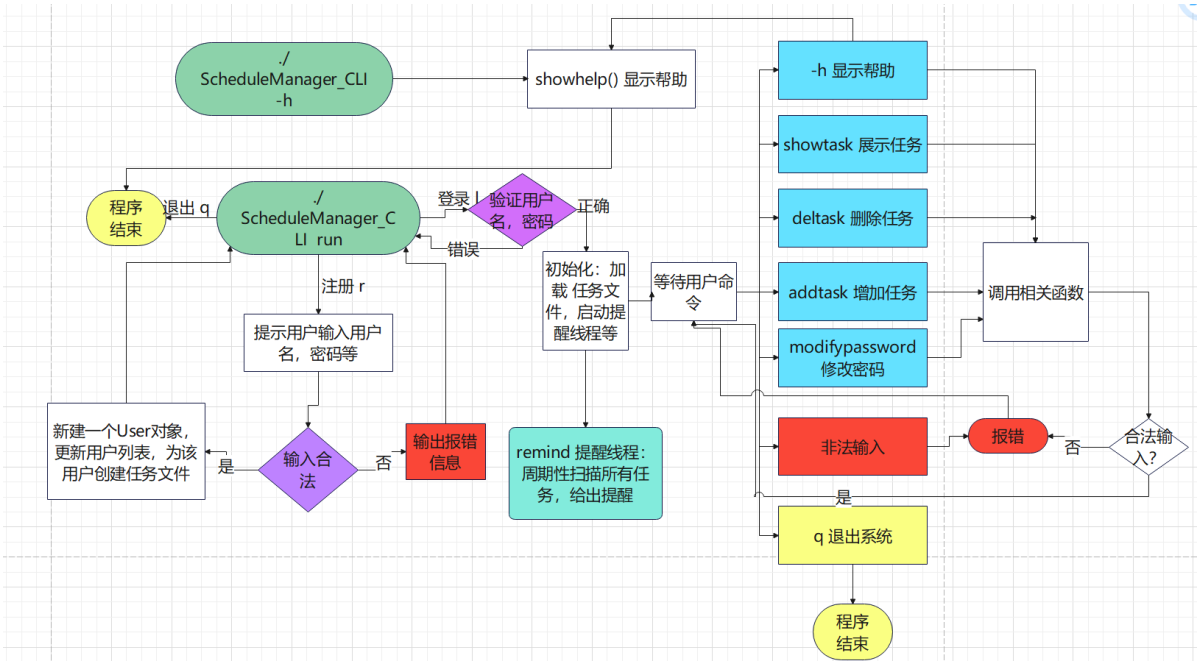
本项目严格遵从C++面向对象编程的思想，采用模块化编程。对于所有类均给出头文件和源文件。

```
schedule/
├── CLI/
|   ├── build/                  # 编译生成的目录，包含生成的可执行文件
|   ├── include/                # 头文件目录
|   |   ├── Interact.h          # 与用户交互相关的函数声明
|   |   ├── Reminder.h          # 提醒功能相关的类声明
|   |   ├── Task.h              # 任务类的声明
|   |   ├── TaskManager.h       # 任务管理类的声明
|   |   ├── TimeUtils.h         # 时间处理工具函数的声明
|   |   ├── User.h              # 用户类的声明
|   |   ├── Utils.h             # 工具函数的声明
|   ├── src/                    # 源文件目录
|   |   ├── main.cpp            # 项目的主程序入口
|   |   ├── Interact.cpp        # 与用户交互相关的函数实现
|   |   ├── Reminder.cpp        # 提醒功能相关的类实现
|   |   ├── Task.cpp            # 任务类的实现
|   |   ├── TaskManager.cpp     # 任务管理类的实现
|   |   ├── TimeUtils.cpp       # 时间处理工具函数的实现
|   |   ├── User.cpp            # 用户类的实现
|   |   ├── Utils.cpp           # 工具函数的实现
|   ├── test.sh                 # 自动化测试脚本，用于测试命令行版本的功能
|   ├── CMakeLists.txt          # CMake 配置文件，用于生成项目的构建系统
├── web/                        # Web 版本的代码和相关文件
|...
```

本项目主要包含五大模块：**交互模块，用户管理模块，任务管理模块，工具函数模块，以及测试模块。**



2.3 流程图



2.4 关键技术问题实现

2.4.1 多线程

在项目中，多线程机制主要体现在 `Reminder` 类的实现上。`Reminder` 类通过一个独立的线程不断检查是否有任务需要提醒。

- `std::thread`：用于创建一个新的线程，在本项目中，启动一个线程来定期检查任务提醒。
- `std::this_thread::sleep_for`：使线程睡眠指定时间段，这里是10秒，表示每10秒检查一次任务提醒。

为了防止多个线程同时对Task列表或者任务文件进行写操作，或者当一个线程进行写操作时，另一个文件进行读操作，需要引入互斥锁，以实现多个线程对临界资源的互斥访问。

`std::mutex mtx`：在 `Utils` 命名空间中定义一个互斥锁实例 `mtx`。

- `std::lock_guard<std::mutex>`：RAII 方式管理互斥锁，**在构造时自动获取锁，在析构时自动释放锁**，避免手动调用 `lock` 和 `unlock` 带来的错误。

当然，在添加任务，显示任务，提醒线程遍历任务等涉及到读写文件和修改Task列表的操作时，也要进行加锁操作。另外，由于提醒线程每10s才检查一次，检查完立刻释放锁，所以不会引起死锁的问题。

2.4.2 时间的获取与处理

本项目中时间的获取与处理使用了C++标准库中的`<chrono>`、`<ctime>`、`<iomanip>`、`<sstream>`和`<regex>`，这些库和函数提供了强大的时间管理和字符串处理能力。下面详细解释时间获取与处理的具体实现。

1. 获取当前时间
    - 使用`std::chrono`库获取当前系统时间。
    - 使用`std::time_t`和`std::localtime`函数将系统时间转换为易于操作的时间结构。
2. 格式化时间
    - 使用`std::ostringstream`和`std::put_time`函数将时间结构格式化为字符串。
3. 解析时间
    - 使用`std::istringstream`和`std::get_time`函数将时间字符串解析为时间结构。
4. 验证时间格式
    - 使用正则表达式`<regex>`验证时间字符串格式的合法性。

2.4.3 哈希加密

在本项目中，密码的加密存储是通过使用SHA-256哈希函数实现的。SHA-256是一种加密哈希函数，用于将输入数据转换为固定长度的散列值（哈希值）。该方法确保了密码的安全存储，即使用户信息泄露，攻击者也难以从散列值反推出原始密码。

在项目中，我们使用了OpenSSL库中的EVP（高层加密抽象）接口来实现SHA-256哈希算法。EVP接口提供了一种通用的方式来调用各种加密和哈希算法。

核心代码

```
EVP_DigestFinal_ex(context, hash, &lengthOfHash);
EVP_MD_CTX_free(context);
```

- `EVP_DigestFinal_ex(context, hash, &lengthOfHash)`：完成哈希计算，并将结果存储在`hash`数组中，长度存储在`lengthOfHash`中。
- `EVP_MD_CTX_free(context)`：释放上下文资源。

2.4.4 密码盲输

通过使用`termios`库，可以临时禁用终端的回显功能，以确保在用户输入密码时，密码不会显示在屏幕上。这种方法提高了输入密码的安全性，防止密码被旁人窥视。

---

3. Web版本设计

3.1 功能概述与创新点

功能概述

Web版本的ScheduleManager项目是一个基于Node.js和Express.js框架开发的全栈Web应用，提供以下关键功能：

- **用户认证**：支持用户注册、登录和密码修改，使用会话管理确保安全性。
- **任务管理**：允许用户添加、查询、删除任务，支持按日期和类别筛选。
- **提醒功能**：用户可设置任务提醒，系统在指定时间发送通知。

创新点

1. **密码盲输**：在密码输入过程中，采用盲输技术，提升安全性。
2. **多线程提醒检查**：利用多线程技术，周期性地检查任务提醒，提高响应速度。
3. **数据库交互优化**：使用`sqlite3`和`sqlite`库，通过参数化查询避免SQL注入。
4. **异步操作**：使用`async/await`处理异步数据库操作，提高代码可读性。

Web版本的ScheduleManager项目通过这些功能和创新点，为用户提供了一个安全、高效且易于使用的任务管理平台。

3.2 项目结构设计

```
schedule/
├── CLI/
├── web/
|   ├── database/                  # 数据库
|   |   ├── data.db                # sqlite数据库文件
|   |   ├── data.sqlite            # 备份文件
|   ├── public/                    # 静态资源目录，包含前端页面和样式文件
|   |   ├── image/                 # 图片资源
|   |   |   ├── bg.jpg             # 登录页面背景
|   |   |   ├── dashboard.jpg      # 操作页面背景
|   |   |   ├── secret.jpg         # 修改密码页面背景
|   |   ├── Dung_Def_roar_02.ogg   # 提醒音频文件
|   |   ├── login.html             # 用户登录页面
|   |   ├── signup.html            # 用户注册页面
|   |   ├── dashboard.html         # 用户操作界面
|   |   ├── dashboard_2.html
|   |   ├── secret.html            # 修改密码界面
|   |   ├── dashboard_style.css    # 样式表文件
|   |   ├── style.css
|   |   ├── secret.css
|   ├── auth.js                    # 用户认证模块
|   ├── controller.js              # 前后端交互控制模块
|   ├── services.js                # 数据库交互模块
|   ├── routes.js                  # 路由配置模块
|   ├── index.js                   # 服务器启动文件
|   ├── package.json               # 项目依赖和配置
```

（1）技术栈

- **前端**：HTML, CSS, JavaScript
- **后端**：Node.js, Express.js
- **数据库**：SQLite
- **安全**：CryptoJS（密码加密）

3.3 关键技术说明

3.3.1 Web服务器构建

使用 `express` 框架来创建和管理服务器及其路由。

3.3.3 数据库交互

使用 `sqlite3` 和 `sqlite` 库来操作 SQLite 数据库，创建 `Tasks` 和 `Users` 表，并进行数据的增删改查操作。

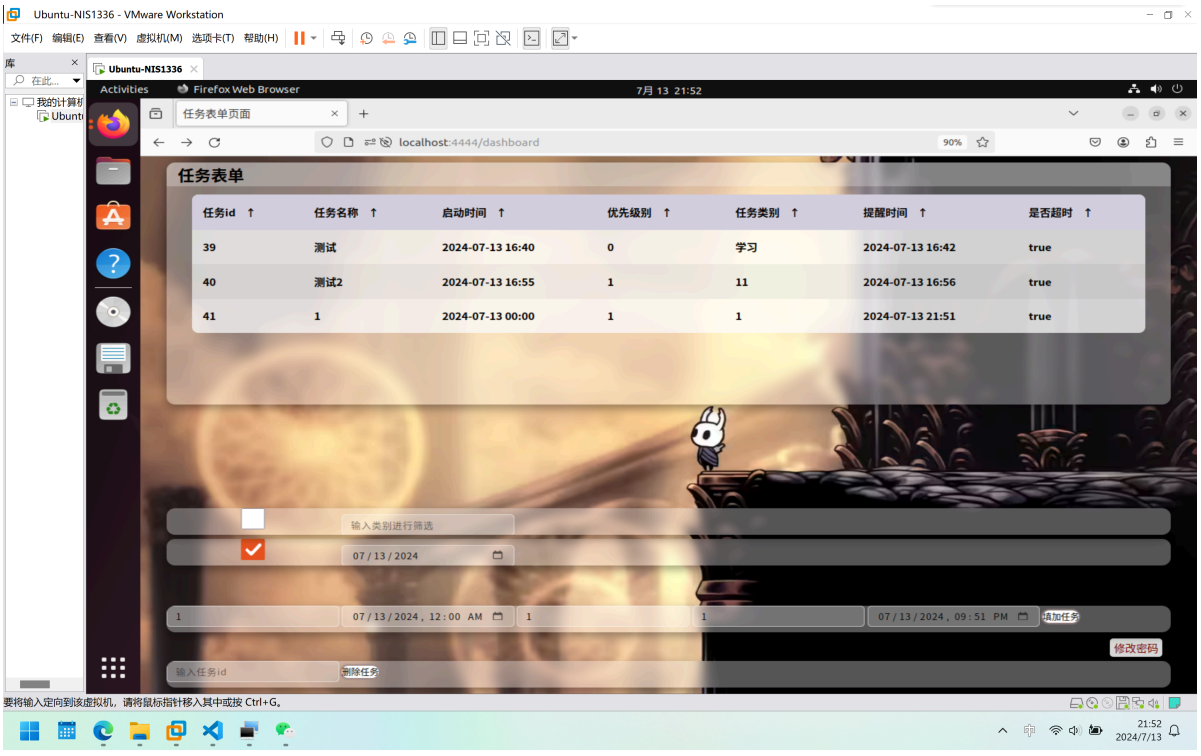
3.3.7 异步操作

大量使用 `async/await` 语法来处理异步数据库操作，提高代码的可读性和维护性。

3.3.9 RESTful API设计

遵循 RESTful 原则设计 API，使用不同的 HTTP 方法（如 GET, POST, DELETE）来实现资源的获取、创建和删除。

3.4 项目效果展示



操作界面