← 返回

Select an Agent

to view its workspace

管理知识库

Knowledge Base Off

Seed 选择模式现在设置为 "Off"，你的知识库将不会被使用。

```cpp
};

// ==========================================
// 4. 任务封装 (修复 C26495 未初始化警告)
// ==========================================
struct ScheduledTask {
    shared_ptr<ITask> task;
    chrono::system_clock::time_point runTime;

    // <--- 修复：显式初始化成员变量
    bool isPeriodic = false;
    chrono::milliseconds interval = chrono::milliseconds(0);

    bool operator>(const ScheduledTask& other) const {
        return runTime > other.runTime;
    }
};

// ==========================================
// 5. 任务调度器 (修复锁警告)
// ==========================================
class TaskScheduler {
private:
    priority_queue<ScheduledTask, vector<ScheduledTask>, greater<ScheduledTask>> taskQueue;
    mutex qMutex;
    condition_variable cv;
    bool running = true;
    thread workerThread;
    vector<IObserver*> observers;
    mutex obsMutex;

    TaskScheduler() {
        workerThread = thread(&TaskScheduler::WorkerLoop, this);
    }

    void NotifyObservers(const string& msg) {
        lock_guard<mutex> lock(obsMutex);
        for (auto obs : observers) obs->OnTaskEvent(msg);
    }

    void WorkerLoop() {
        while (true) { // 使用 true，在内部判断退出
            ScheduledTask currentTask;
            bool haveTask = false;

            {
                unique_lock<mutex> lock(qMutex);
                // 等待条件：有任务且时间到了，或者停止运行
                cv.wait(lock, [this] {
                    if (!running) return true;
                    if (taskQueue.empty()) return false;
                    return true;
                });

                if (!running && taskQueue.empty()) break; // 退出循环

                if (!taskQueue.empty()) {
                    auto topTask = taskQueue.top();
                    auto now = chrono::system_clock::now();
                    if (topTask.runTime > now) {
                        // 时间未到，继续等待，设置超时
                        cv.wait_until(lock, topTask.runTime);
                        // 醒来后重新循环检查
                        continue;
                    }

                    // 取出任务
                    currentTask = topTask;
                    taskQueue.pop();
                    haveTask = true;
                }
            } // <--- 锁在这里自动释放，解决 C26115

            if (haveTask) {
                NotifyObservers("Executing: " + currentTask.task->GetName());
                try {
                    currentTask.task->Execute();
                } catch (...) {}

                if (currentTask.isPeriodic && running) {
                    currentTask.runTime = chrono::system_clock::now() + currentTask.interval;
                    {
                        lock_guard<mutex> lock(qMutex);
                        taskQueue.push(currentTask);
                    }
                    NotifyObservers("Rescheduled: " + currentTask.task->GetName());
                    cv.notify_one();
                }
            }
        }
    }

public:
    static TaskScheduler& Instance() {
        static TaskScheduler instance;
        return instance;
    }

    ~TaskScheduler() { Stop(); }

    void AddObserver(IObserver* obs) {
        lock_guard<mutex> lock(obsMutex);
        observers.push_back(obs);
    }

    void AddTask(shared_ptr<ITask> task, int delayMs, int intervalMs = 0) {
        ScheduledTask st;
        st.task = task;
        st.runTime = chrono::system_clock::now() + chrono::milliseconds(delayMs);
        st.interval = chrono::milliseconds(intervalMs);
        st.isPeriodic = (intervalMs > 0);

        {
            lock_guard<mutex> lock(qMutex);
            taskQueue.push(st);
        }
        cv.notify_one();

        stringstream ss;
        ss << "Added: " << task->GetName();
        LogWriter::Instance().Write(ss.str());
        NotifyObservers(ss.str());
    }

    void Stop() {
        {
            lock_guard<mutex> lock(qMutex);
            running = false;
        }
        cv.notify_all();
        if (workerThread.joinable()) workerThread.join();
    }
};

// ==========================================
// 6. 工厂与界面
// ==========================================
class TaskFactory {
public:
    static shared_ptr<ITask> CreateTask(char type) {
        switch (toupper(type)) {
            case 'A': return make_shared<TaskBackup>();
            case 'B': return make_shared<TaskMatrix>();
            case 'C': return make_shared<TaskHttp>();
            case 'D': return make_shared<TaskReminder>();
            case 'E': return make_shared<TaskStats>();
            default: return nullptr;
        }
    }
};

class ConsoleUI : public IObserver {
private:
    mutex uiMutex;
public:
    ConsoleUI() { TaskScheduler::Instance().AddObserver(this); }

    void OnTaskEvent(const string& message) override {
        lock_guard<mutex> lock(uiMutex);
        cout << "\n[Notify] " << message << "\n> " << flush;
    }

    void ShowMenu() {
        while (true) {
            {
                lock_guard<mutex> lock(uiMutex);
                cout << "\n=== Task Scheduler ===\n";
                cout << "[A] Backup (Delay 2s)\n";
                cout << "[B] Matrix (Every 5s)\n";
                cout << "[C] HTTP (Now)\n";
                cout << "[D] Reminder (Every 1min)\n";
                cout << "[E] Stats (Delay 10s)\n";
                cout << "[Q] Quit\n";
                cout << "> " << flush;
            }

            char choice;
            cin >> choice;
            if (toupper(choice) == 'Q') break;
                                            tory::CreateTask(choice);

                int d = 0, i = 0;
```

追问当前话题

常规模式　GPT 4.1　x 1

Nanobanana Pro is free

Flower-soil　TRIAL

订阅与设置