Trae 长效记忆系统自动记录功能全流程升级方案（终版）

一、项目背景与核心目标

1.1 现状痛点分析

现有长效记忆系统存在“手动触发依赖症”，需用户明确通知Agent才能记录信息，导致记忆覆盖率低、时效性差，无法发挥长效记忆的核心价值。经技术排查，核心问题如下：

- **配置与功能脱节**：`memory-integration-config.yaml` 中已配置 `auto_record_operations: true` 等自动记录开关，但未实际实现对应功能
- **架构支持但未落地**：具备事件驱动架构及监听器组件，但仅用于系统内部通信，未对接外部对话与操作场景
- **集成层缺失**：缺少与Trae IDE的钩子集成代码，无法捕获对话内容、代码执行等关键交互数据

1.2 核心升级目标

构建“无感式全自动记忆系统”，无需用户手动干预，实现全场景数据自动捕获、智能筛选、安全存储，具体目标包括：

- 全场景覆盖：自动记录用户输入、Agent响应、代码执行、文件变更、错误日志等核心交互数据
- 智能高效：通过多重筛选机制排除冗余信息，仅保留高价值记忆，兼顾性能与实用性
- 安全可控：内置隐私保护机制，自动过滤敏感数据，支持快速回滚与配置调整
- 兼容稳定：基于现有技术栈开发，与历史配置无缝兼容，不影响系统原有功能

二、整体技术架构设计

2.1 架构分层设计

采用四层架构设计，实现“数据捕获-智能处理-存储优化-配置管控”全链路闭环，各层职责如下：

- **集成层**：通过Trae IDE钩子与中间件，实现交互数据的实时捕获与拦截
- 再全**智能处理层**：包含智能筛选、实体分析、相似度计算等模块，筛选高价值记忆
- **存储优化层**：采用队列异步处理、批量压缩存储，提升性能并降低资源占用
- **配置管控层**：统一管理自动记录开关、阈值参数、隐私规则，支持灵活调整

2.2 核心技术栈确认

基于现有系统技术栈选型，确保兼容性与可维护性，核心依赖如下：

- 开发语言：TypeScript 5.2.2（与项目前端技术栈一致）
- IDE集成：`@traejs/ide-sdk ≥1.2.0`（官方推荐集成工具）
- 核心框架：`@traejs/core ≥2.0.0`（匹配知识图谱功能版本要求）
- 辅助工具：Jest 29.7.0（单元测试）、ESLint 8.0+（代码规范校验）

三、核心功能实现方案

3.1 集成层：Trae IDE交互捕获实现

3.1.1 对话与操作钩子开发

基于IDE SDK开发钩子组件，监听全量交互事件，代码实现如下：

Code block

```
// src/integrations/trae-ide/hooks/InteractionHook.ts
import { IDEPlugin, EventType, InteractionEvent } from '@traejs/ide-sdk';
import { MemoryService } from '../../../services/MemoryService';
import { ContextExtractor } from '../../../utils/ContextExtractor';

export class InteractionHook {
  private plugin: IDEPlugin;
  private memoryService: MemoryService;
  private contextExtractor: ContextExtractor;

  constructor(plugin: IDEPlugin) {
    this.plugin = plugin;
    this.memoryService = new MemoryService();
    this.contextExtractor = new ContextExtractor();
    this.initEventListeners(); // 初始化监听器
  }

  // 初始化各类交互事件监听器
  private initEventListeners() {
    // 1. 对话交互事件（用户输入+Agent响应）
    this.plugin.on(EventType.USER_INPUT, (event: InteractionEvent) =>
      this.handleUserInput(event.data)
    );
    this.plugin.on(EventType.AGENT_RESPONSE, (event: InteractionEvent) =>
      this.handleAgentResponse(event.data)
    );
```

```
27
28      // 2. 开发操作事件
29      this.plugin.on(EventType.CODE_EXECUTION, (event: InteractionEvent) =>
30        this.handleCodeExecution(event.data)
31      );
32      this.plugin.on(EventType.FILE_CHANGE, (event: InteractionEvent) =>
33        this.handleFileChange(event.data)
34      );
35
36      // 3. 系统事件
37      this.plugin.on(EventType.ERROR_LOG, (event: InteractionEvent) =>
38        this.handleErrorLog(event.data)
39      );
40      this.plugin.on(EventType.CONVERSATION_END, (event: InteractionEvent) =>
41        this.handleConversationEnd(event.data.conversationId)
42      );
43    }
44
45    // 处理用户输入事件
46    private async handleUserInput(data: any) {
47      const context = this.contextExtractor.extract({
48        activeFile: data.activeFile,
49        projectState: data.projectState,
50        cursorPos: data.cursorPosition
51      });
52      // 提交至记忆服务（异步非阻塞）
53      await this.memoryService.submitMemory({
54        type: 'user_input',
55        content: data.content,
56        conversationId: data.conversationId,
57        timestamp: new Date().toISOString(),
58        context
59      });
60    }
61
62    // 处理Agent响应事件
63    private async handleAgentResponse(data: any) {
64      await this.memoryService.submitMemory({
65        type: 'agent_response',
66        content: data.content,
67        conversationId: data.conversationId,
68        timestamp: new Date().toISOString(),
69        metadata: {
70          confidenceScore: data.confidenceScore,
71          responseTime: data.responseTime
72        }
73      });
```

```
74      }
75
76      // 其他事件处理方法（代码执行、文件变更等）省略...
77  }
```

3.1.2 自动记录中间件集成

开发全局中间件，拦截所有交互请求并同步至记忆系统，确保无数据遗漏：

（注：文档部分内容可能由 AI 生成）