**Trae记忆功能实施指南：打造智能编程助手**

**📋 实施清单**

本指南将帮助你在Trae中实现完整的记忆功能，包括：

* 规则系统设置（个人规则 + 项目规则）
* Knowledge Graph Memory配置
* 智能体创建和优化
* 实际案例演示
* 最佳实践建议

**1️⃣ 规则系统实施（基础记忆）**

**1.1 个人规则设置（全局生效）**

**操作步骤：**

1. **打开设置界面**启动Trae IDE（确保版本≥1.8.0，低版本需通过官网下载升级包手动升级）
2. 点击右上角 ⚙️ 图标进入系统设置
3. 在左侧菜单中选择"规则管理"页签（部分旧版本显示为"全局规则"）
4. **创建个人规则文件**在"个人规则"区域点击 **"+ 新建规则文件"**，系统默认命名为user\_rules.md（可自定义但建议保留默认名）
5. 确认文件存储路径：Windows默认C:\Users\用户名\.trae\，Mac默认/Users/用户名/.trae/（路径不可修改，需确保目录有读写权限）
6. 点击"确认创建"后，文件会自动在IDE编辑器中打开
7. **配置规则并验证生效**复制下方示例代码到文件中，根据个人开发习惯修改编码偏好、工具选择等配置
8. 按Ctrl+S（Windows）/Command+S（Mac）保存文件，系统会在5秒内自动加载新规则（底部状态栏会显示"规则加载成功"提示）
9. 验证方法：新建空白会话，输入"我的编码命名规范是什么？"，若返回配置的camelCase、PascalCase等规则则生效；若未生效点击"规则管理"页签的"刷新规则"按钮重试
10. **关键注意事项**规则冲突处理：当个人规则与项目规则冲突时，以项目规则为准，可在个人规则中通过!override 项目规则键名语法临时覆盖（仅本地生效，不影响团队）
11. 规则备份：建议每季度导出个人规则（设置→规则管理→导出规则），避免重装IDE导致配置丢失
12. 特殊字符限制：规则文件中禁止使用@extends、#!等特殊语法开头的行（会被系统识别为指令而非配置）

|  |
| --- |
| Plain Text # Trae 个人规则配置 > 说明：采用Markdown格式编写，#后为注释文本，执行时自动忽略；优先级低于项目规则和企业规则  ## 🎯 编码偏好 - \*\*编程语言\*\*：优先使用TypeScript 5.0+（类型安全），兼容场景可使用ES6+ JavaScript - \*\*代码风格\*\*：严格遵循Airbnb规范（需在项目中安装eslint-config-airbnb-base@15.0.0及以上版本） - \*\*缩进与换行\*\*：2个空格缩进（禁止用Tab），Windows用CRLF换行，Mac用LF换行 - \*\*命名规范\*\*：  - 变量/函数：camelCase（例：userInfo、handleSubmit）  - 常量：UPPER\_SNAKE\_CASE（例：MAX\_RETRY\_COUNT、API\_BASE\_URL）  - 组件/类：PascalCase（例：LoginForm、UserService）  - 文件名：与导出主体一致，组件文件后缀.tsx，工具文件后缀.ts  ## 📝 响应输出规范 - \*\*技术解释\*\*：需包含"作用+核心逻辑+使用场景"三要素，避免模糊表述 - \*\*代码示例\*\*：必须附带导入语句、错误处理、关键注释，复杂逻辑需拆分步骤说明 - \*\*兼容性提示\*\*：涉及框架版本差异时，需标注适用版本范围（例：React 18.x专属Hook）  ## 🔧 工具偏好 - \*\*包管理\*\*：优先npm 9.0+（需配置镜像源：npm config set registry https://registry.npmmirror.com），次选yarn 3.0+ - \*\*测试工具\*\*：Jest 29.0+（单元测试）+ React Testing Library 14.0+（组件测试） - \*\*代码质量\*\*：ESLint 8.0+（语法检查）+ Prettier 3.0+（格式美化），需同步配置.prettierrc文件 - \*\*构建工具\*\*：React项目用Vite 4.0+（开发效率），大型应用用Webpack 5.0+（打包优化）  ## ⚠️ 安全规范 - \*\*敏感信息\*\*：禁止硬编码密钥/密码，必须通过环境变量注入（例：process.env.API\_KEY） - \*\*输入校验\*\*：用户输入需用joi@17.11.0+或zod@3.22.0+校验，覆盖必填、格式、长度约束 - \*\*防注入处理\*\*：SQL操作必须用参数化查询，前端用DOMPurify@3.0.6+处理HTML输入 - \*\*React安全\*\*：禁用dangerouslySetInnerHTML，确需使用时必须先通过DOMPurify净化  ## 📚 文档要求 - \*\*注释规范\*\*：函数/类/接口必须加JSDoc注释，包含@description、@param、@returns、@throws - \*\*README必备\*\*：环境要求、安装命令、启动步骤、目录说明、常见问题（FAQ） - \*\*API文档\*\*：RESTful接口用Swagger/OpenAPI 3.0，GraphQL用Apollo Studio生成Schema文档 - \*\*变更日志\*\*：遵循SemVer语义化版本（主版本.次版本.修订版），记录变更内容和兼容说明 |

**1.2 项目规则设置（项目内生效）**

**操作步骤：**

1. **关联项目并打开规则界面**启动Trae IDE后，通过"文件→打开项目"选择目标项目根目录（需确保目录包含package.json文件，否则无法识别为项目）
2. 在左侧"项目导航栏"右键点击项目根目录，选择"Trae配置→项目规则设置"（或点击顶部工具栏"⚙️ 项目配置"图标）
3. **创建项目规则文件**在弹出的"项目规则管理"面板中，点击 **"+ 创建项目规则"**，系统自动在根目录创建.trae/rules/project\_rules.md路径（目录不存在会自动创建）
4. 团队协作配置：将.trae/rules/目录添加到.gitignore的例外规则（在.gitignore中添加!.trae/rules/），确保团队成员共享规则
5. 权限检查：Windows需确保项目目录不在"Program Files"等系统保护目录；Mac需在"系统设置→隐私与安全性→文件和文件夹"中授予IDE访问权限
6. **规则文件结构与加载机制**项目根目录/  
   └── .trae/ # Trae项目配置目录（需提交Git）  
    └── rules/ # 规则存储目录  
    ├── project\_rules.md # 主规则文件（必选，自动加载）  
    └── extends/ # 扩展规则目录（可选，需手动导入）  
    ├── style.md # 样式规范（例：CSS-in-JS配置）  
    └── security.md # 安全规则（例：接口鉴权要求）**加载机制**：主规则通过@extends ./extends/style.md语法导入子规则，支持最多3层嵌套，禁止循环导入（会导致规则加载失败）；优先级：主规则＞扩展规则，项目规则＞个人规则
7. **配置与验证项目规则**复制下方示例代码到project\_rules.md，根据项目技术栈修改技术栈版本、目录结构等配置
8. 保存后点击面板"应用规则"按钮，底部状态栏显示"项目规则生效"即完成配置
9. 验证：在项目会话中输入"项目用的UI组件库是什么？"，若返回配置的Ant Design及版本则生效
10. **关键注意事项**团队协作禁忌：禁止在项目规则中写入个人路径（如C:\Users\张三）、私人密钥等信息，需用环境变量${ENV\_NAME}替代
11. 规则更新机制：修改项目规则后需通知团队成员执行"规则管理→同步项目规则"，否则成员本地规则不会自动更新
12. 版本回滚：若新规则导致功能异常，可在"规则管理→历史版本"中回滚至最近3个有效版本（保留7天历史记录）

|  |
| --- |
| Plain Text # 电商管理系统 - 项目规则配置 > 适用范围：当前项目所有会话；优先级：企业规则＞本规则＞个人规则 > 维护责任人：技术负责人（张三，zhangsan@company.com）；更新频率：技术栈变更后1个工作日内  ## 🛠️ 项目技术栈（含版本约束） - \*\*前端核心\*\*：React 18.2.0（禁止升级至19.x，未完成兼容性测试）+ TypeScript 5.2.2 - \*\*状态管理\*\*：Redux Toolkit 2.0.1（强制用createSlice，禁止直接调用createStore） - \*\*UI组件\*\*：Ant Design 5.12.1（仅用官方组件，禁止引入第三方UI库避免样式冲突） - \*\*路由管理\*\*：React Router 6.20.1（采用createBrowserRouter配置式路由，禁止用HashRouter） - \*\*HTTP请求\*\*：Axios 1.6.2（必须使用封装工具src/services/request.ts，禁止直接调用axios） - \*\*后端依赖\*\*：Node.js 18.17.0（LTS版）+ Express 4.18.2 + MySQL 8.0 - \*\*ORM工具\*\*：Sequelize 6.35.1（禁止直接写原生SQL，复杂查询需经技术负责人审批） - \*\*依赖管理\*\*：用package-lock.json锁定版本（禁止提交yarn.lock，.gitignore已配置）  ## 📁 项目目录结构（强制遵循） ``` src/ ├── assets/ # 静态资源（图片/字体/样式，需用import引入） ├── components/ # 公共组件（按复用范围拆分） │ ├── common/ # 全局通用组件（Button/Input等，全项目复用） │ ├── business/ # 业务组件（OrderCard/GoodsList等，业务模块复用） │ └── layout/ # 布局组件（MainLayout/Header等，页面布局专用） ├── hooks/ # 自定义Hooks（必须以use开头，如useAuth.ts） ├── pages/ # 页面组件（与路由一一对应，含页面级逻辑） │ ├── User/ # 业务模块目录（按功能拆分） │ │ ├── Login/ # 页面目录（含组件+样式+测试文件） │ │ └── Profile/ │ └── Dashboard/ ├── services/ # API服务（按模块拆分，如userService.ts） ├── store/ # Redux状态（按模块拆分slice） │ └── slices/ # 切片目录（authSlice.ts/userSlice.ts等） ├── types/ # TS类型定义（按模块拆分，如user.types.ts） ├── utils/ # 工具函数（无状态，如formatDate.ts/encrypt.ts） └── App.tsx # 应用入口（仅挂载路由，禁止写业务逻辑） ```  ## 🚀 开发流程规范 1. \*\*环境搭建\*\*：npm install（依赖安装，禁止用cnpm）→ npm run dev（启动开发服务，默认端口3000） 2. \*\*编码要求\*\*：遵循ESLint+Prettier规范，VSCode需安装对应插件（配置文件已提交Git） 3. \*\*质量校验\*\*：  - 代码检查：npm run lint（必须修复所有error级问题，warning级需说明原因）  - 类型校验：npm run type-check（禁止存在TS类型错误）  - 单元测试：npm run test（核心逻辑覆盖率≥90%，否则无法提交） 4. \*\*构建部署\*\*：  - 测试环境：npm run build:test → 自动部署至测试服务器  - 生产环境：npm run build（需主干分支且通过CI检查）→ 运维部署至生产服务器  ## 🧪 测试规范（强制执行） - \*\*覆盖率要求\*\*：核心业务逻辑≥90%，工具函数≥95%，UI组件≥80%（用npm run test:coverage查看） - \*\*测试文件位置\*\*：与被测试文件同目录，命名为[文件名].test.tsx（组件）/[文件名].test.ts（工具） - \*\*测试类型\*\*：  - 单元测试：Jest 29.7.0 + React Testing Library 14.1.2（测试独立功能）  - E2E测试：Cypress 13.6.0（测试核心流程，如登录→下单→支付） - \*\*测试要点\*\*：必须覆盖正常场景、异常场景（接口报错/参数错误）、边界场景（空值/极值）  ## 🎨 设计规范（统一风格） - \*\*颜色体系\*\*：全局主题src/assets/theme.ts定义，禁止直接写色值  - 主色：#1890FF（按钮/标题）；成功色：#52C41A；警告色：#FAAD14；错误色：#F5222D - \*\*字体规范\*\*：全局样式src/assets/global.css定义  - 标题：16-24px，字重600；正文：14px，字重400；辅助文字：12px，字重400 - \*\*间距规范\*\*：用src/utils/spacing.ts工具函数（8px基准），禁止硬写px值  - 基础间距：8px；组件间距：16px；页面区块间距：24px  ## 🔒 安全规范（红线要求） 1. \*\*接口安全\*\*：所有请求通过src/services/request.ts自动添加Authorization头，禁止手动添加 2. \*\*权限控制\*\*：基于RBAC模型，权限枚举src/types/permission.types.ts，禁止硬编码权限判断 3. \*\*数据加密\*\*：敏感信息（手机号/身份证）用src/utils/encrypt.ts加密后传输 4. \*\*XSS防护\*\*：用户输入HTML用DOMPurify 3.0.6处理（工具src/utils/sanitize.ts） 5. \*\*安全校验\*\*：提交代码前执行npm run security-scan（依赖npm audit + eslint-plugin-security），高危漏洞必须修复  ## 📦 部署规范 - \*\*环境配置\*\*：不同环境变量区分，禁止硬编码  - 开发：.env.development（本地用，不提交Git）  - 测试：.env.test（测试服，提交Git）  - 生产：.env.production（生产服，CI/CD注入，不提交Git） - \*\*CI/CD流程\*\*：GitHub Actions配置文件.github/workflows/deploy.yml  - 触发条件：main分支合并且所有测试通过  - 流程：安装依赖→lint校验→类型检查→测试→构建→安全扫描→部署 - \*\*版本管理\*\*：遵循SemVer语义化版本（主版本.次版本.修订版）  - 主版本：不兼容API变更（如技术栈重构）  - 次版本：向后兼容功能新增（如新增模块）  - 修订版：向后兼容问题修复（如bug修复） |

**2️⃣ Knowledge Graph Memory 实施（智能记忆）**

**2.1 启用知识图谱记忆**

**操作步骤：**

**2.2 构建知识图谱结构**

1. **前置条件检查（必做步骤）**版本校验：打开终端执行trae --version，确保版本≥2.0.0（知识图谱为2.0+新增功能）；低版本需执行npm install -g trae@latest升级
2. 服务检查：执行netstat -ano | findstr "9200"（Windows）或lsof -i :9200（Mac），确保9200端口未被占用（知识图谱默认端口）；若占用需关闭占用进程或联系运维处理
3. 模块启用：进入Trae设置→"功能模块"，勾选"智能体引擎"和"知识图谱服务"，点击"重启服务"使配置生效（重启需10-15秒，期间无法使用相关功能）
4. **创建自定义智能体**点击左侧导航栏"智能体"图标（🤖），进入智能体市场页面
5. 点击页面右上角"新建自定义智能体"（非"市场智能体"），进入配置界面
6. 基础信息配置（精准定位场景）：  
    - 名称："电商管理系统-前端开发专家V1.0"（含项目和版本，便于多项目区分）  
    - 描述："专注电商管理系统前端开发，精通React 18.2.0+TypeScript 5.2.2技术栈，严格遵循项目编码、测试、安全规范，提供可直接落地的开发方案"  
    - 分类：前端开发→React开发→电商领域（按实际场景选择，影响智能体匹配精度）  
    - 头像：上传前端开发相关图标（建议尺寸200\*200px，支持PNG/JPG格式）  
    - 上下文窗口：设置为150000（项目规则+核心文档约10万字，按1.5倍配置）
7. **集成并配置Knowledge Graph Memory工具**在智能体配置页切换至"工具集成"选项卡，在"可选工具"列表中找到"Knowledge Graph Memory"，点击"添加"并在弹窗中确认授权
8. 工具关键配置（直接影响记忆效果）：  
    - 记忆存储模式：选择"项目专属存储"（避免与其他项目记忆混淆，数据隔离）  
    - 存储路径：默认项目根目录/.trae/knowledge-graph/（自动创建，需加入.gitignore）  
    - 自动同步：勾选"会话结束后自动同步记忆"（确保会话中的新知识被保存）  
    - 过期策略：选择"未使用30天自动归档"（不删除历史记忆，归档后可手动恢复）  
    - 检索阈值：设置为0.7（相似度≥0.7的记忆才会被检索，避免无关信息干扰）
9. 点击"测试连接"按钮，若显示"工具连接成功"则配置有效；若失败检查9200端口和服务状态
10. **激活智能体并验证记忆功能**完成配置后点击页面底部"保存并激活"按钮，智能体状态显示"已激活"（激活后配置不可修改，需修改需先"停用"）
11. 记忆功能验证：  
     1. 初始化知识图谱：执行下方"知识图谱初始化脚本"，向记忆中导入项目技术栈、目录结构等基础信息  
     2. 会话测试：在智能体会话框输入"项目用的React版本和状态管理工具是什么？"  
     3. 验证结果：若返回"React 18.2.0，Redux Toolkit 2.0.1，强制用createSlice语法"则记忆生效
12. **关键注意事项**数据安全：知识图谱存储路径默认包含敏感信息，需在.gitignore中添加.trae/knowledge-graph/\*（仅提交初始化脚本，不提交存储数据）
13. 服务稳定性：9200端口被占用时，可在Trae设置→知识图谱服务→端口配置中修改为9201-9210间未占用端口，修改后需重启IDE
14. 记忆清理：执行trae knowledge-graph clean --expired 30d可手动清理30天未访问记忆，避免存储占用过高（建议每月执行1次）

**2.3 智能记忆检索**

|  |
| --- |
| Plain Text // 知识图谱初始化脚本：项目根目录/scripts/init-knowledge-graph.js // 作用：批量导入项目技术栈、人员、结构等基础信息到知识图谱，避免手动录入 // 执行步骤：1. 安装依赖 2. 配置API密钥 3. 执行脚本 // 依赖安装：npm install @traejs/sdk@1.2.0 --save-dev（固定版本避免兼容问题）  const TraeSDK = require('@traejs/sdk');  // 1. 初始化SDK（关键信息，需替换为实际值） const trae = new TraeSDK({  apiKey: 'sk\_8f7d6c5b4a3e2d1f0a9b8c7d6e5f4a3b', // 从Trae设置→API密钥→创建密钥（需勾选知识图谱读写权限）  projectId: 'proj\_1234567890abcdef', // 从项目设置→基本信息→项目ID获取  baseUrl: 'http://localhost:9200' // 知识图谱服务地址（默认本地，集群部署需改对应地址） });  // 2. 知识图谱初始化核心函数 const initializeKnowledgeGraph = async () => {  try {  console.log('开始初始化知识图谱...');   // 2.1 创建核心实体（实体ID建议加项目前缀，避免跨项目冲突）  // 项目实体  await trae.tools.knowledgeGraphMemory.addEntity({  id: "ecommerce\_project\_main",  type: "Project",  properties: {  name: "电商管理系统",  description: "企业级电商后台管理系统，含用户/商品/订单/库存等核心模块",  status: "开发中（当前迭代V1.2，截止2025-03-31）",  startDate: "2025-01-15",  expectedEndDate: "2025-06-30",  teamSize: 5,  repoUrl: "https://github.com/company/ecommerce-admin.git",  docUrl: "https://docs.company.com/ecommerce/v1.0/"  },  tags: ["前端项目", "React项目", "电商领域"] // 标签用于快速检索  });   // 技术栈实体（React）  await trae.tools.knowledgeGraphMemory.addEntity({  id: "ecommerce\_tech\_react",  type: "Technology",  properties: {  name: "React",  version: "18.2.0",  category: "Frontend Framework",  documentation: "https://react.dev/reference/react",  usageScope: "全项目前端界面开发",  constraint: "禁止使用React 19.x API，未完成兼容测试",  dependency: "需配合React Router 6.20.1使用"  },  tags: ["核心技术", "前端框架"]  });   // 技术栈实体（TypeScript）  await trae.tools.knowledgeGraphMemory.addEntity({  id: "ecommerce\_tech\_ts",  type: "Technology",  properties: {  name: "TypeScript",  version: "5.2.2",  category: "Programming Language",  documentation: "https://www.typescriptlang.org/docs/",  usageScope: "全项目前端开发",  constraint: "必须开启strict模式（tsconfig.json已配置）"  },  tags: ["核心技术", "编程语言"]  });   // 其他技术栈实体可参考上述格式添加（Redux Toolkit/Ant Design等）   // 2.2 创建实体关系（建立实体间关联，提升检索精度）  // 项目-技术关系：电商管理系统使用React  await trae.tools.knowledgeGraphMemory.addRelationship({  source: "ecommerce\_project\_main",  target: "ecommerce\_tech\_react",  type: "使用技术",  properties: {  usage: "前端界面开发、组件封装、状态管理",  importance: "核心技术（占比80%）",  versionConstraint: "固定18.2.0"  }  });   // 项目-技术关系：电商管理系统使用TypeScript  await trae.tools.knowledgeGraphMemory.addRelationship({  source: "ecommerce\_project\_main",  target: "ecommerce\_tech\_ts",  type: "使用技术",  properties: {  usage: "类型定义、代码编写、类型校验",  importance: "核心技术（占比100%）"  }  });   // 2.3 创建人员实体及关系（可选，便于团队协作）  await trae.tools.knowledgeGraphMemory.addEntity({  id: "ecommerce\_dev\_zhangsan",  type: "Person",  properties: {  name: "张三",  role: "前端开发工程师",  expertise: ["React 18.x", "TypeScript 5.x", "Redux Toolkit"],  projectRole: "核心开发，负责订单模块"  },  tags: ["项目成员", "前端开发"]  });   await trae.tools.knowledgeGraphMemory.addRelationship({  source: "ecommerce\_dev\_zhangsan",  target: "ecommerce\_project\_main",  type: "负责开发",  properties: {  module: "订单管理模块（创建/支付/退款）",  responsibilities: ["UI开发", "状态管理", "API集成", "单元测试"]  }  });   console.log("✅ 知识图谱初始化完成！可在Trae→知识图谱面板查看实体和关系");  } catch (error) {  console.error("❌ 知识图谱初始化失败:", error.message);  console.error("🔍 排查步骤：");  console.error("1. 检查API密钥是否正确且有读写权限（Trae设置→API密钥）");  console.error("2. 确认9200端口未被占用（执行netstat/lsof命令检查）");  console.error("3. 项目ID是否正确（项目设置→基本信息）");  console.error("4. @traejs/sdk版本是否为1.2.0（package.json查看）");  } };  // 3. 执行初始化函数 initializeKnowledgeGraph();  // 执行命令：node scripts/init-knowledge-graph.js // 成功输出：✅ 知识图谱初始化完成！可在Trae→知识图谱面板查看实体和关系 // 失败处理：根据错误提示排查后重新执行，重复执行会覆盖相同ID的实体 |

**3️⃣ 智能体实施（高级记忆）**

**3.1 创建专业智能体**

|  |
| --- |
| javascript // 智能记忆检索工具函数（封装为通用工具：src/utils/memory-retrieval.ts） // 依赖：需在智能体配置中启用Knowledge Graph Memory工具 import { trae } from '@traejs/sdk';  /\*\*  \* 检索相关记忆  \* @param query 检索关键词（支持自然语言）  \* @param options 检索配置  \* @returns 格式化后的记忆结果  \*/ export const retrieveRelevantMemory = async (  query: string,  options: { limit?: number; similarityThreshold?: number } = {} ) => {  try {  const { limit = 5, similarityThreshold = 0.7 } = options;    // 1. 语义检索相关记忆（支持多维度匹配：关键词、实体类型、标签）  const memories = await trae.tools.knowledgeGraphMemory.search({  query: query,  limit: limit,  similarityThreshold: similarityThreshold,  includeTags: true, // 返回记忆关联的标签  includeAccessHistory: true // 返回记忆最近访问时间  });   if (memories.length === 0) {  console.warn(`未检索到与"${query}"相关的记忆，建议检查关键词或补充记忆`);  return [];  }   // 2. 整理和格式化记忆结果（便于前端展示或后续处理）  const formattedMemories = memories.map(memory => {  return {  relevance: Number(memory.score.toFixed(2)), // 相似度得分（保留2位小数）  type: memory.entity?.type || memory.relationship?.type || "Memory",  id: memory.entity?.id || memory.relationship?.id, // 记忆唯一标识  content: generateMemorySummary(memory),  tags: memory.tags || [],  lastAccessed: memory.lastAccessed   ? new Date(memory.lastAccessed).toLocaleString()   : "未访问过",  source: memory.source === "manual" ? "手动添加" : "自动同步" // 记忆来源  };  });   // 按相似度降序排序  return formattedMemories.sort((a, b) => b.relevance - a.relevance);  } catch (error) {  console.error("记忆检索失败:", error.message);  console.error("排查步骤：1. 检查Knowledge Graph服务是否运行 2. 智能体工具授权是否有效 3. 网络连接是否正常");  return [];  } };  /\*\*  \* 生成记忆摘要（根据记忆类型适配不同格式）  \* @param memory 原始记忆数据  \* @returns 简洁的记忆摘要  \*/ const generateMemorySummary = (memory: any) => {  if (memory.entity?.type === "Person") {  return `${memory.entity.properties.name}（${memory.entity.properties.role}），擅长${memory.entity.properties.expertise.join("、")}，负责${memory.entity.properties.projectRole || "未知模块"}`;  } else if (memory.entity?.type === "Project") {  return `项目《${memory.entity.properties.name}》（${memory.entity.properties.status}）：${memory.entity.properties.description.slice(0, 50)}...，代码仓库：${memory.entity.properties.repoUrl}`;  } else if (memory.entity?.type === "Technology") {  return `技术${memory.entity.properties.name} ${memory.entity.properties.version}（${memory.entity.properties.category}）：${memory.entity.properties.usageScope}，文档：${memory.entity.properties.documentation}`;  } else if (memory.relationship) {  const sourceName = memory.source?.properties?.name || memory.source;  const targetName = memory.target?.properties?.name || memory.target;  return `${sourceName} ${memory.relationship.type} ${targetName}（${memory.relationship.properties?.module || memory.relationship.properties?.usage || ""}）`;  }  return memory.content?.length > 100   ? `${memory.content.slice(0, 100)}...`   : memory.content; };  // 示例使用 // retrieveRelevantMemory("React版本").then(res => console.log(res)); |

**操作步骤：**

|  |
| --- |
| **智能体创建关键注意事项**   * 上下文窗口限制：窗口大小超过200000会导致智能体响应延迟＞3秒，建议按"项目规则50%+知识图谱30%+会话历史20%"分配 * 权限管控：仅向核心开发授予"智能体编辑"权限，普通成员授予"使用权限"即可（避免误修改系统提示词） * 迭代管理：智能体版本按"V主版本.次版本.修订版"命名（如V1.0.2），每次修改配置后需更新版本号并记录变更日志 * 性能优化：同时运行的自定义智能体不超过3个，多余智能体需手动停用（避免CPU占用率＞80%） |

|  |
| --- |
| javascript // 前端开发专家智能体系统提示词 const systemPrompt = `你是一名资深前端开发工程师，拥有10年专业开发经验，专注于React生态系统。  ## 核心能力 - 精通React、TypeScript、Redux等前端技术栈 - 深入理解前端工程化和性能优化 - 熟悉现代前端开发流程和最佳实践 - 具备良好的代码审查和技术指导能力  ## 工作原则 1. \*\*代码质量第一\*\*：始终编写可维护、可扩展、高性能的代码 2. \*\*用户体验优先\*\*：关注界面美观性和交互流畅性 3. \*\*安全性保障\*\*：重视代码安全性和数据保护 4. \*\*性能优化\*\*：主动考虑前端性能优化策略 5. \*\*学习指导\*\*：不仅提供答案，更要解释原理和思路  ## 记忆管理 - 记住用户的技术背景和项目需求 - 记录之前的技术讨论和解决方案 - 学习用户的编码风格和偏好 - 关联相关的技术文档和最佳实践  ## 响应规范 - 提供详细的技术解释和代码示例 - 包含必要的注释和文档说明 - 主动指出潜在问题和改进建议 - 使用清晰的结构和格式组织内容  ## 工具使用 - 熟练使用Knowledge Graph Memory管理技术知识 - 利用Sequential Thinking分析复杂技术问题 - 通过File System查看和分析项目代码 - 使用联网搜索获取最新的技术资讯和文档 `; |

**3.2 智能体记忆增强**

1. **智能体核心配置（专业级设置）基础信息（精准定位场景）**  
    - 名称："电商管理系统-前端开发专家V1.0"（包含版本便于迭代）  
    - 描述："专注于电商管理系统前端开发，精通React 18.2.0+TypeScript 5.2.2技术栈，严格遵循项目编码规范、测试规范和安全要求，提供符合项目架构的解决方案"  
    - 分类：前端开发→React开发→电商领域  
    - 头像：选择与前端开发相关的图标（增强辨识度）

|  |
| --- |
| javascript // 智能体记忆增强配置 const agentMemoryConfig = {  // 记忆存储配置  storage: {  type: "knowledge\_graph",  persistence: true,  autoSave: true  },    // 记忆检索配置  retrieval: {  similarityThreshold: 0.6,  maxResults: 10,  includeRelated: true  },    // 记忆更新策略  update: {  autoLearn: true,  conflictResolution: "latest",  importanceScoring: true  },    // 记忆清理策略  cleanup: {  enable: true,  retentionPeriod: "6 months",  lowPriorityThreshold: 0.3  } }; |

**4️⃣ 实际案例演示**

**4.1 项目初始化案例**

**用户需求：**

|  |
| --- |
| text 我需要创建一个电商管理系统的前端项目，使用React和TypeScript， 请帮我初始化项目结构并配置开发环境。 |

**智能体处理流程：**

1. **记忆检索**// 检索相关记忆  
   const relevantMemories = await retrieveRelevantMemory("电商管理系统 React TypeScript 项目初始化");  
     
   // 记忆检索结果  
   [  
    {  
    relevance: 0.95,  
    type: "Project",  
    content: "项目 电商管理系统 是一个企业级电商后台管理系统，当前状态：开发中",  
    lastAccessed: "2025-01-15"  
    },  
    {  
    relevance: 0.85,  
    type: "Person",   
    content: "开发者姓名 是一名 前端开发工程师，擅长 React, TypeScript, Redux, Node.js",  
    lastAccessed: "2025-01-15"  
    },  
    {  
    relevance: 0.75,  
    type: "Technology",  
    content: "React 18.2.0 是一个 Frontend Framework，文档地址：https://react.dev/",  
    lastAccessed: "2025-01-15"  
    }  
   ]
2. **系统提示词精细化配置（提升专业性）**
3. **核心参数配置（影响智能体性能）**  
    - 上下文窗口：150000（根据项目文档规模调整，建议为项目规则+核心文档总字符数的1.5倍）  
    - 响应温度：0.3（开发场景建议0.2-0.4，降低随机性）  
    - 最大响应长度：8000（满足代码示例+解释的长度需求）  
    - 记忆优先级：项目规则＞技术栈信息＞个人偏好＞通用知识
4. **生成项目初始化方案**# 项目初始化命令  
   npx create-react-app ecommerce-admin --template typescript  
   cd ecommerce-admin  
     
   # 安装核心依赖  
   npm install @reduxjs/toolkit react-redux antd axios react-router-dom  
   npm install --save-dev @types/react-router-dom @testing-library/react @testing-library/jest-dom  
     
   # 创建项目结构  
   mkdir -p src/{assets,components,hooks,pages,services,store,types,utils}  
     
   # 创建基础文件  
   touch src/types/index.ts  
   touch src/utils/request.ts  
   touch src/store/index.ts  
   touch src/services/api.ts
5. **更新知识图谱记忆**// 更新项目进度记忆  
   await trae.tools.knowledgeGraphMemory.updateEntity({  
    id: "project\_ecommerce",  
    properties: {  
    status: "初始化完成",  
    initializedDate: "2025-01-16",  
    projectStructure: "已创建基础目录结构",  
    dependencies: ["React", "TypeScript", "Redux Toolkit", "Ant Design"]  
    }  
   });  
     
   // 记录操作历史  
   await trae.tools.knowledgeGraphMemory.addEntity({  
    id: "action\_001",  
    type: "Action",  
    properties: {  
    name: "项目初始化",  
    description: "创建React+TypeScript电商管理系统项目",  
    timestamp: new Date().toISOString(),  
    result: "成功创建项目结构和配置开发环境"  
    }  
   });

**4.2 代码开发案例**

**用户需求：**

|  |
| --- |
| text 请帮我创建一个用户登录组件，包含用户名、密码输入框和登录按钮， 需要表单验证和错误提示功能。 |

**智能体响应：**

1. **工具集成与权限控制**必选工具：Knowledge Graph Memory（记忆管理）、File System（项目文件访问）、Code Runner（代码验证）  
   工具权限：  
    - File System：仅授权访问当前项目目录（禁止访问系统目录）  
    - Code Runner：仅允许运行ts、js、json文件（禁止运行可执行文件）  
    - Knowledge Graph Memory：授予读写权限（禁止删除权限，防止误操作）
2. 工具调用规则：代码生成前必须检索知识图谱获取项目规范，生成后自动调用Code Runner验证语法正确性
3. **智能体验证与调优**功能验证：输入3类典型问题验证响应质量  
    1. 规范类："项目中React组件的命名规范是什么？"（验证规则记忆）  
    2. 技术类："如何用Redux Toolkit实现用户登录状态管理？"（验证技术栈记忆）  
    3. 实操类："写一个符合项目规范的商品列表组件"（验证规范落地能力）
4. 调优策略：若响应不符合预期，按以下顺序调整  
    1. 优化系统提示词（补充未覆盖的规范细节）  
    2. 调整记忆优先级（确保项目规则优先加载）  
    3. 扩大上下文窗口（确保核心信息完整加载）
5. **全流程通用注意事项**版本兼容：Trae 1.8.0-1.9.0版本不支持知识图谱嵌套导入，升级至2.0.0+需先备份个人规则和项目规则
6. 异常排查：所有功能异常先执行"帮助→修复工具→一键修复"（自动校验服务状态、权限和配置），无效再联系技术支持
7. 日志查看：规则加载、记忆同步等失败时，可在"设置→高级→日志查看"中筛选"规则系统"或"知识图谱"模块定位问题

|（注：文档部分内容可能由 AI 生成)