快速入门教程

本文将带你从零开始构建一个基于 Table Memory Engine 的表格记忆工作流，并说明如何在对话中触发标签、同步表格以及输出 Prompt。

环境准备

- Node.js 18+（与 `table-git` 保持一致）
- 已克隆本仓库并在根目录执行 `npm install`
- 根目录执行 `npm run build`，确保生成 `dist/` 供子包引用
- 进入 `packages/table-memory-engine` 执行 `npm install`

第一步：初始化 TableGit 与适配器

```
import { TableGit } from 'table-git';
import { TableGitAdapter } from '@table-git/memory-engine';

const tableGit = new TableGit();
tableGit.init('main', { defaultSheetName: 'memory' });

const adapter = new TableGitAdapter({
  tableGit,
  defaultSheetId: 'memory',
  autoInit: false,
  defaultAuthor: 'Memory Bot',
  defaultCommitMessage: changes => `自动应用 ${changes.length} 条记忆指令`
});
```

- `init` 将创建主分支与默认工作表
- 适配器负责把节点产生的 `TableChangeCommand` 翻译为 TableGit 操作

第二步：构建节点图

```
import { FlowBuilder } from '@table-git/memory-engine';

const flow = FlowBuilder.create('memory-demo', '记忆同步流程')
  .useNode('LoadTable', { sheetId: 'memory' })
  .useNode('ParseTags')
  .useNode('ApplyChanges')
  .useNode('FormatPrompt', { formatter: 'prompt' })
  .build();
```

- `LoadTable`：载入工作表快照并写入上下文变量
- `ParseTags`：扫描对话消息，输出结构化指令

- `ApplyChanges`：将指令转换为变更命令并执行
- `FormatPrompt`：根据最新快照生成 Prompt 文本

第三步：准备运行时

```
import { NodeRuntime, registerBuiltinNodes } from '@table-git/memory-engine';

const runtime = new NodeRuntime({ adapter });
registerBuiltinNodes(runtime);
```

- `registerBuiltinNodes` 会注入内置节点、解析器、格式化器
- 若需自定义节点，可调用 `runtime.register(customNode)`

第四步：准备对话输入

```
const conversation = [
  {
    id: 'msg-1',
    role: 'assistant',
    content: '[[table:setCell sheet=memory row=0 column=0 value="Alice"]]'
  },
  {
    id: 'msg-2',
    role: 'assistant',
    content: '[[table:setCell sheet=memory row=0 column=1 value="已完成
onboarding"]]'
  }
];
```

- 默认解析器识别 `[[table:action key=value ...]]` 语法
- 可通过注册新的 `TagParserPlugin` 扩展语法

第五步：执行流程

```
const result = await runtime.run(flow, {
  conversation,
  sheetId: 'memory',
  services: {}
});

console.log(result.context.variables?.formatted);
```

运行结果示例：

```
Sheet: memory
Revision: <hash>
Metadata: {}
Rows:
Row 0: C0: Alice, C1: 已完成 onboarding
```

进阶：监听事件

```
runtime.getEventBus().on('afterNode', ({ nodeId, nodeType }) => {
  console.log(`[event] ${nodeType} (${nodeId}) 执行完成`);
});
```

可用事件包括：`beforeLoad`、`afterLoad`、`beforeNode`、`afterNode`、`beforeApply`、`afterApply`、`conflict`、`error`。

进阶：自定义标签解析器

```
import { ComposedTagParser } from '@table-git/memory-engine';

const parser = runtime.getParser();
parser.register({
  name: 'json-directive',
  match: turn => turn.content.trim().startsWith('{'),
  parse: turn => {
    const data = JSON.parse(turn.content);
    return [{
      tag: 'json',
      action: data.action,
      target: { sheetId: data.sheet },
      payload: data.payload,
      raw: turn.content
    }];
  }
});
```

进阶：自定义格式化器

```
runtime.getFormatters().register({
  name: 'summary',
  title: '概览摘要',
  description: '输出首行作为概要',
  factory: async ({ snapshot }) => {
    const firstRow = snapshot.rows[0] ?? [];
    return `记忆摘要: ${firstRow.join(' / ')}`;
```

```
  }
});
```

随后在节点图中将 `FormatPrompt` 的 `formatter` 配置为 `summary` 即可。

下一步

- 阅读 `api-reference.md` 了解完整 API
- 查阅 `node-library.md` 探索内置节点与自定义扩展模式
- 在 `examples/` 中尝试更多场景脚本

附录：事件驱动入口

如果希望根据不同事件触发相应的工作流，可使用 `MemoryWorkflowEngine`：

```
import {
  MemoryWorkflowEngine,
  registerDefaultEventFlows
} from '@table-git/memory-engine';

const engine = new MemoryWorkflowEngine(runtime);
registerDefaultEventFlows(engine);

await engine.dispatch({
  id: 'evt-1',
  type: 'ai:reply',
  sheetId: 'memory',
  conversation
});
```

默认事件映射：

- `table:init` → 加载表格并格式化输出
- `ai:reply` → 加载 + 标签解析 + 应用 + 格式化
- `user:message` → 加载 + 标签解析 + dry-run 预览

你可以通过 `engine.register` 自定义事件与节点流的映射，从而扩展出更多业务场景。