API 参考

本文档汇总 `table-git` 的主要对象与函数，按照功能分组列出常用方法。更多细节可阅读源码或查看类型声明。

> 所有类型均已导出，导入路径默认为 `table-git`。

TableGit

`TableGit` 是整个工具包的核心类，负责表格数据、结构、暂存区与历史的协同管理。

```
import { TableGit, createTableGit } from 'table-git';

const repo: TableGit = createTableGit('main');
```

仓库生命周期

方法	说明
`init(branchName?: string)`	初始化仓库，可指定初始分支名。调用工厂函数创建实例时通常会自动执行。
`reset()`	清空暂存区与未保存状态，回到当前 HEAD。

单元格操作

方法	说明
`addCellChange(sheet, row, col, value, formula?, format?)`	在暂存区记录单元格新增或更新。
`deleteCellChange(sheet, row, col)`	记录单元格删除。
`getCell(row, col)`	获取当前快照下的单元格对象。
`getCellValue(row, col)`	读取当前快照中的单元格值。

列操作

方法	说明
`addColumn(sheet, column)`	添加列元数据到工作表。
`updateColumn(sheet, columnId, updates)`	更新列属性（宽度、约束、描述等）。
`moveColumn(sheet, columnId, newIndex)`	调整列顺序。
`deleteColumn(sheet, columnId)`	根据 ID 删除列。

方法	说明
`deleteColumnByIndex(sheet, index, options?)`	根据索引删除列，可包含暂存区。
`getNextColumnOrder(sheet, options?)`	依据现有结构返回下一个列顺序值。

行操作

方法	说明
`addRow(sheet, row)`	添加行元数据。
`deleteRow(sheet, rowId)`	删除指定 ID 的行。
`deleteRowByIndex(sheet, index, options?)`	根据索引删除行。
`sortRows(sheet, criteria[])`	执行排序并记录操作。
`getNextRowOrder(sheet, options?)`	获取下一行顺序值。

版本控制

方法	说明
`commit(message, author, email)`	将暂存区写入历史，返回提交哈希。
`createBranch(name)`	基于当前 HEAD 创建新分支。
`checkout(ref)`	切换到目标分支或提交（支持 detached HEAD）。
`checkoutCommit(hash)`	直接切换到指定提交哈希。
`getCurrentBranch()`	返回当前分支名（若处于 detached HEAD 则返回 `null`）。
`getBranches()`	列出分支。
`createTag(name, options?)`	创建标签，可指定目标提交或附加注释（`author`、`email`、`message`）；支持 `force` 覆盖。
`deleteTag(name)`	删除现有标签。
`listTags(options?)`	列出标签名称，传入 `{ withDetails: true }` 可获取 `TagInfo` 详情。
`getTag(name)`	返回指定标签的 `TagInfo`（包含指向的提交、类型、注释等）。
`status()`	返回暂存区与工作区的差异概览。
`getStagedChanges()`	返回已暂存的变更列表。
`getCommitHistory(limit?)`	获取提交历史，可限制数量。

数据持久化

方法	说明
`exportState(options?)`	返回一个纯数据对象（`SerializedTableGitState`），用于自定义持久化（数据库、远端服务等）。默认采用 `'minimal'` 预设，仅导出当前引用可达的对象集合，不附带工作区 / 暂存区 / 快照。
`importState(state, options?)`	从 `exportState` 返回的数据恢复仓库状态。默认还原暂存区、快照与工作区，可通过选项关闭。
`exportStateAsJSON(options?)`	基于 `exportState` 直接返回 JSON 字符串。默认无需额外格式化（无空格换行），若传入 `pretty: true` 则输出带缩进的可读版。

`exportState` 支持的关键选项：

- `includeWorkingState`：是否附带当前工作表与工作区缓存（`'minimal'` 预设默认为 `false`，`'full'` 预设为 `true`）。
- `includeSnapshots`（默认 `false`）：导出 `tableSnapshots`，适合频繁切换提交而不想重新构建快照的场景。
- `includeStagedChanges`（`'minimal'` 预设默认 `false`，`'full'` 预设 `true`）：是否返回暂存区记录。
- `preset`：`'minimal' | 'full'`，默认为 `'minimal'`。精简模式仅导出当前引用可达的对象，且默认关闭工作区、暂存区与标签冗余信息；`'full'` 模式与旧行为相同。
- `roots`：指定根引用集合并裁剪导出的对象集合（例如只导出指定分支/标签/提交）。
- `stripDefaults`、`stripTagDetails`：控制是否移除默认值或标签的注释信息。精简模式默认开启。

`importState` 支持的选项：

- `restoreWorkingState`（默认 `true`）：如导入数据中包含工作区则直接使用，否则自动根据 HEAD 重建。
- `restoreSnapshots`（默认 `true`）：导入快照提升后续切换性能。
- `restoreStagedChanges`（默认 `true`）：保留暂存区，方便在持久化后继续提交。

DiffMergeEngine

负责基于 `TableGit` 仓库计算差异并执行合并。

```
import { DiffMergeEngine } from 'table-git';

const engine = new DiffMergeEngine(repo);
const result = engine.merge('feature-branch');
```

- `diff(targetBranch)`：返回当前分支与目标分支的差异。
- `merge(targetBranch)`：尝试将目标分支合并进当前分支，返回是否成功及冲突信息。

ConflictResolver

辅助处理冲突。

- `resolveCellConflict(conflict, strategy)`：对单条单元格冲突执行策略（`current`、`incoming`、`merge` 等）。
- `resolveStructureConflict(conflict, strategy)`：处理结构冲突。
- `batchResolve(conflicts, strategy)`：对一组冲突批量执行同一策略。

TableStructure & SheetTree

- `TableStructure`：维护列、行的元数据快照。
- `SheetTree`：管理单个工作表的树形数据，包括单元格与结构。

工厂函数

位于 `utils/factory`，但已通过包入口导出。

函数	说明
`createTableGit(branchName?)`	创建并初始化 `TableGit` 实例。
`createSampleTable()`	创建内置示例，适合测试或 Demo。
`createColumn(id, options)`	快速构造列元数据，自动补齐必需字段。
`createRow(options?)`	生成行元数据。
`createCell(row, col, value, formula?, format?)`	生成单元格对象。

工具函数 (`utils/hash`)

函数	说明
`calculateHash(value)`	对任意结构化数据生成稳定的 SHA1 哈希。
`generateId(prefix?)`	生成唯一 ID，尽可能利用加密随机源。
`deepClone(value)`	对常见结构（对象、数组、Map、Set、Date 等）进行深克隆。
`deepEqual(a, b)`	判断两份数据结构是否相等。
`parsePosition(position)`	将 "row,col" 字符串解析为数值。
`formatPosition(row, col)`	将位置转换为字符串形式。

类型与常量

- `ChangeType`：枚举所有变更类型。
- `Change`：表示一条变更记录。
- `ColumnMetadata`、`RowMetadata`、`CellFormat`、`CellValue` 等：描述结构化表格元素。
- `SortCriteria`：排序操作的输入结构。

这些类型在 TypeScript 中可直接导入：

```
import {
  Change,
  ChangeType,
  ColumnMetadata,
  RowMetadata,
  SortCriteria
} from 'table-git';
```

错误与异常

大多数方法会在输入非法或状态不匹配时抛出 `Error`，常见情况包括：

- 重复创建已存在名称的分支。
- 删除不存在的列或行。
- 在 detached HEAD 状态下尝试提交（需先创建新分支）。

建议在外层捕获异常并结合 `status()` / `getStagedChanges()` 进行提示。