Table Version Control System

一个基于 Git 思想的表格版本控制系统，支持单元格级别的粒度控制和表结构的版本管理。

🚀 特性

- **单元格级版本控制** - 每个单元格都有独立的版本历史
- **表结构版本管理** - 支持列的增删改、移动、排序等操作的版本控制
- **行操作支持** - 支持行的添加、删除、排序等操作
- **Git 风格操作** - 支持分支、提交、合并、差异比较等 Git 核心功能
- **历史提交切换** - 支持切换到任意历史提交状态（detached HEAD）
- **智能冲突解决** - 提供多种冲突解决策略
- **TypeScript 支持** - 完整的类型安全支持
- **格式化支持** - 支持单元格格式、公式等复杂数据

📦 安装

```
npm install table-git
```

🎯 快速开始

快速演示

运行内置演示查看系统功能：

```
git clone <repository-url>
cd table-git
npm install
npm run build
node demo.js
```

这将展示：

- 创建示例表格
- 添加产品数据
- 创建分支进行价格调整
- 删除行操作演示
- 行排序功能演示
- 切换到历史提交（detached HEAD）
- 查看提交历史
- 演示分支切换

基础使用

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

// 创建新的表格仓库
const repo = createTableGit('main');

// 添加列定义
const column = createColumn('产品名称', {
  dataType: 'string',
  width: 150,
  constraints: { required: true }
});

repo.addColumn('Sheet1', column);

// 添加数据
repo.addCellChange('Sheet1', 1, 1, 'iPhone 15');
repo.addCellChange('Sheet1', 1, 2, 5999);

// 提交变更
const commitHash = repo.commit('初始化产品表', 'Alice', 'alice@example.com');
console.log(`提交成功: ${commitHash.substring(0, 7)}`);
```

分支操作

```
// 创建分支
repo.createBranch('feature-branch');
repo.checkout('feature-branch');

// 在分支中进行修改
repo.addCellChange('Sheet1', 1, 2, 6299); // 调整价格
repo.commit('价格调整', 'Bob', 'bob@example.com');

// 切换回主分支
repo.checkout('main');

// 切换到历史提交 (detached HEAD)
const history = repo.getCommitHistory();
repo.checkoutCommit(history[1].hash); // 切换到第二个提交
```

行操作

```
// 添加行
const row = createRow({ height: 30 });
repo.addRow('Sheet1', row);

// 删除行
repo.deleteRow('Sheet1', 'row_id_123');
```

```
// 排序行
repo.sortRows('Sheet1', [
  { columnId: 'price_column', ascending: false }
]);

repo.commit('行操作示例', 'User', 'user@example.com');
```

合并和冲突解决

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

const diffEngine = new DiffMergeEngine(repo);
const mergeResult = diffEngine.merge('feature-branch');

if (!mergeResult.success && mergeResult.conflicts) {
  const resolver = new ConflictResolver();

  // 自动解决冲突
  const resolved = resolver.batchResolve(mergeResult.conflicts, 'current');

  // 或者手动解决每个冲突
  mergeResult.conflicts.forEach(conflict => {
    if ('position' in conflict) {
      const resolution = resolver.resolveCellConflict(conflict, 'merge');
      console.log(`冲突已解决: ${resolution?.value}`);
    }
  });
}
```

核心概念

对象模型

- **CellObject**: 单元格对象，包含值、公式、格式等信息
- **TableStructure**: 表结构对象，管理列和行的元数据
- **SheetTree**: 工作表树，管理单个工作表的数据和结构
- **CommitObject**: 提交对象，记录变更历史

版本控制

系统采用类似 Git 的版本控制模型：

1. **工作区** - 当前编辑状态
2. **暂存区** - 准备提交的变更
3. **提交历史** - 已提交的版本历史
4. **分支** - 独立的开发线

变更类型

支持以下变更类型：

- `CELL_ADD/UPDATE/DELETE` - 单元格操作
- `COLUMN_ADD/UPDATE/DELETE/MOVE` - 列操作
- `ROW_ADD/UPDATE/DELETE/SORT` - 行操作

分支状态

- **普通分支** - 标准的分支状态，可以进行提交
- **Detached HEAD** - 切换到历史提交的状态，只读模式

🔧 API 参考

TableGit 类

```
class TableGit {
  // 初始化
  init(branchName?: string): void

  // 单元格操作
  addCellChange(sheet: string, row: number, col: number, value: CellValue,
formula?: string, format?: CellFormat): void
  deleteCellChange(sheet: string, row: number, col: number): void
  getCellValue(row: number, col: number): CellValue | undefined
  getCell(row: number, col: number): CellObject | undefined

  // 列操作
  addColumn(sheet: string, column: ColumnMetadata): void
  updateColumn(sheet: string, columnId: string, updates:
Partial<ColumnMetadata>): void
  deleteColumn(sheet: string, columnId: string): void
  moveColumn(sheet: string, columnId: string, newIndex: number): void

  // 行操作
  addRow(sheet: string, row: RowMetadata): void
  deleteRow(sheet: string, rowId: string): void
  sortRows(sheet: string, criteria: SortCriteria[]): void

  // 版本控制
  commit(message: string, author: string, email: string): string
  createBranch(branchName: string): void
  checkout(branchName: string): void
  checkoutCommit(commitHash: string): void
  reset(): void

  // 状态查询
  status(): object
  getStagedChanges(): Change[]
  getCommitHistory(limit?: number): CommitObject[]
```

```
  getCurrentBranch(): string
  getBranches(): string[]
}
```

便利函数

```
// 创建实例
createTableGit(branchName?: string): TableGit
createSampleTable(): TableGit

// 创建对象
createColumn(name: string, options?: object): ColumnMetadata
createRow(options?: object): RowMetadata
createCell(row: number, col: number, value: CellValue, formula?: string,
format?: CellFormat): CellObject
```

🩹 测试

项目包含完整的测试套件，覆盖所有核心功能：

```
# 运行测试
npm test

# 运行测试并监听变化
npm run test:watch

# 运行测试并生成覆盖率报告
npm run test -- --coverage
```

测试覆盖功能：

- ☑ 基础表格操作
- ☑ 分支创建和切换
- ☑ 历史提交切换
- ☑ 行和列操作
- ☑ 单元格格式和公式
- ☑ 版本控制核心功能

🏗 构建

```
# 构建项目
npm run build

# 开发模式（监听变化）
npm run dev
```

```
# 清理构建文件
npm run clean
```

项目结构

```
src/
├── core/                  # 核心功能
│   ├── cell.ts           # 单元格对象
│   ├── structure.ts      # 表结构管理
│   ├── sheet.ts          # 工作表树
│   ├── commit.ts         # 提交对象
│   ├── table-git.ts      # 主版本控制引擎
│   ├── diff-merge.ts     # 差异比较和合并
│   └── conflict-resolver.ts # 冲突解决
├── types/                # 类型定义
│   └── index.ts
├── utils/                # 工具函数
│   ├── hash.ts           # 哈希和工具函数
│   └── factory.ts        # 便利创建函数
└── index.ts              # 主入口文件

tests/                    # 测试文件
examples/                 # 使用示例
```

使用场景

1. **协作表格编辑** - 多人协作编辑同一表格，跟踪每个人的修改
2. **数据变更追踪** - 追踪财务报表、数据分析表的历史变更
3. **表结构演进** - 管理数据库表结构的版本变化
4. **实验性分析** - 在不同分支中进行假设分析和对比
5. **审计合规** - 提供完整的数据变更审计轨迹
6. **数据回滚** - 快速回滚到任意历史版本
7. **A/B 测试** - 在不同分支中测试不同的数据方案
8. **团队协作** - 避免数据冲突，支持并行编辑

示例

查看 `examples/usage-examples.ts` 文件获取完整的使用示例，包括：

- 基础表格操作
- 分支操作和合并
- 历史提交切换
- 行操作演示
- 差异比较
- 冲突解决
- 表格结构版本控制

运行示例：

```
npm run build
node demo.js
```

或者运行完整示例：

```
npm run dev
# 然后在另一个终端运行
node dist/examples/usage-examples.js
```

贡献

欢迎提交 Issue 和 Pull Request！

许可证

MIT License

路线图

- [x] 单元格级版本控制
- [x] 表结构版本管理
- [x] 分支和合并操作
- [x] 历史提交切换
- [x] 行操作支持
- [x] 冲突解决机制
- [ ] 支持多工作表
- [ ] 实现远程仓库同步
- [ ] 添加图形化差异显示
- [ ] 支持单元格公式依赖分析
- [ ] 添加数据导入导出功能
- [ ] 实现权限控制系统
- [ ] 添加撤销/重做功能
- [ ] 支持表格模板系统