开发指南

开发环境设置

必需工具

- Node.js >= 18.12.0
- pnpm >= 8.0.0
- Git
- Three.js开发工具（浏览器插件）

项目设置

1. 克隆项目

```
git clone https://github.com/your-org/space-management.git
cd space-management
```

2. 安装依赖

```
pnpm install
```

3. 配置环境变量

```
cp .env.example .env
```

4. 启动开发服务器

```
pnpm dev
```

5. 构建生产版本

```
pnpm build
```

开发规范

代码风格

- 使用TypeScript进行开发

- 使用ESLint和Prettier进行代码格式化
- 遵循React Hooks的最佳实践
- 使用函数式组件和Hooks

文件组织

- 组件文件放在`src/components`目录下
- 每个组件创建独立目录，包含index.tsx和样式文件
- 工具函数放在`src/utils`目录下
- 类型定义放在`src/types`目录下
- Redux相关代码放在`src/store`目录下

命名规范

- 组件使用PascalCase命名
- 函数和变量使用camelCase命名
- 常量使用UPPER_SNAKE_CASE命名
- 类型和接口使用PascalCase命名

功能开发流程

1. 图元开发

1. 在`types`目录下定义图元类型
2. 在`components/Elements`下创建图元组件
3. 在`utils/elementUtils.ts`中添加图元相关工具函数
4. 在`store/spaceSlice.ts`中添加图元相关操作
5. 在`Canvas2D.tsx`和`Canvas3D.tsx`中实现图元渲染

2. 命令开发

1. 在`types/command.ts`中定义命令类型
2. 在`store/commandSlice.ts`中实现命令处理逻辑
3. 在相关组件中调用命令

3. 工具函数开发

1. 在`utils`目录下创建或更新工具函数
2. 编写单元测试
3. 在组件中使用工具函数

测试规范

单元测试

- 使用Jest和React Testing Library
- 测试文件以`.test.ts(x)`结尾
- 测试覆盖关键业务逻辑和工具函数

组件测试

- 测试组件的关键交互
- 测试组件的渲染逻辑
- 测试组件的状态管理

调试技巧

Redux DevTools

- 使用Redux DevTools查看状态变化
- 使用Redux DevTools调试命令执行

Canvas调试

- 使用debug模式显示辅助信息
- 使用Chrome DevTools的Canvas面板
- 使用自定义的调试工具函数

Three.js调试

- 使用Three.js Inspector
- 使用Scene Explorer
- 使用性能监控工具

性能优化

React优化

- 使用React.memo避免不必要的重渲染
- 使用useMemo和useCallback缓存值和函数
- 使用虚拟列表处理大量数据

Canvas优化

- 使用离屏Canvas进行缓存
- 实现图层系统
- 优化重绘策略

Three.js优化

- 使用实例化渲染
- 实现LOD（细节层次）
- 优化光照和阴影计算

发布流程

1. 版本发布

1. 更新版本号

2. 更新更新日志
3. 构建生产版本
4. 运行测试
5. 提交代码并创建标签

2. 部署

1. 构建Docker镜像
2. 运行集成测试
3. 部署到测试环境
4. 验证功能
5. 部署到生产环境