计算机辅助翻译软件测试文档

**1. 功能测试用例**

**1.1 项目管理功能测试**

测试用例 FT-PM-001: 创建新项目

- 前置条件: 无

- 测试步骤:

1. 点击"新建项目"按钮

2. 输入项目名称"Test Project"和可选描述

3. 点击"创建"按钮

- 预期结果:

- 项目成功创建并显示在项目列表中

- 项目属性与输入一致

测试用例 FT-PM-002: 导入文档到项目

- 前置条件: 已存在一个测试项目

- 测试步骤:

1. 选择测试项目

2. 点击"导入文档"按钮

3. 选择测试用TXT/PDF/DOCX文件

4. 确认导入选项

- 预期结果:

- 文档成功导入并显示在项目文档列表中

- 文档内容被正确解析为翻译段

测试用例 FT-PM-003: 导出翻译文档

- 前置条件: 项目中存在已翻译文档

- 测试步骤:

1. 选择已翻译文档

2. 点击"导出"按钮

3. 选择导出格式(如DOCX)

4. 指定保存位置

- 预期结果:

- 文档按指定格式成功导出

- 导出文件内容与翻译内容一致

- 格式标记被正确保留

**1.2 翻译记忆库功能测试**

测试用例 FT-TM-001: 添加翻译单元到记忆库

- 前置条件: 已创建一个记忆库

- 测试步骤:

1. 打开记忆库管理界面

2. 点击"添加翻译单元"

3. 输入源文本和目标文本

4. 保存

- 预期结果:

- 新翻译单元显示在记忆库列表中

- 单元内容与输入一致

测试用例 FT-TM-002: 记忆库模糊匹配

- 前置条件: 记忆库中包含类似但不完全相同的翻译单元

- 测试步骤:

1. 在翻译编辑器中输入与记忆库中相似的源文本

2. 观察记忆匹配面板

- 预期结果:

- 显示模糊匹配结果

- 匹配率正确计算

- 差异部分高亮显示

测试用例 FT-TM-003: 导入/导出TMX文件

- 前置条件: 已创建一个记忆库

- 测试步骤:

1. 点击"导入TMX"按钮

2. 选择标准TMX测试文件

3. 确认导入

4. 点击"导出TMX"按钮保存文件

- 预期结果:

- TMX文件内容正确导入到记忆库

- 导出的TMX文件符合标准格式

- 导入前后数据一致

**1.3 术语库功能测试**

测试用例 FT-TB-001: 术语识别与提示

- 前置条件: 术语库中包含术语"computer"的翻译

- 测试步骤:

1. 在翻译编辑器中输入包含"computer"的句子

2. 观察术语提示

- 预期结果:

- "computer"被识别为术语

- 显示正确的术语翻译

- 术语在文本中高亮显示

测试用例 FT-TB-002: 术语一致性检查

- 前置条件: 术语库中定义特定术语的翻译

- 测试步骤:

1. 在翻译文档中使用不一致的术语翻译

2. 运行质量检查

- 预期结果:

- 检测到术语不一致

- 报告具体位置和建议的正确翻译

测试用例 FT-TB-003: 批量导入术语

- 前置条件: 准备包含多条术语的TBX/Excel测试文件

- 测试步骤:

1. 点击"批量导入术语"

2. 选择测试文件

3. 确认导入选项

- 预期结果:

- 所有术语正确导入

- 术语属性(词性、领域等)正确解析

- 重复术语按规则处理

**1.4 翻译引擎功能测试**

测试用例 FT-TE-001: 机器翻译预翻译

- 前置条件: 已配置有效的翻译引擎API

- 测试步骤:

1. 选择未翻译文档

2. 点击"预翻译"按钮

3. 选择配置的翻译引擎

4. 执行预翻译

- 预期结果:

- 文档段被自动翻译

- 翻译质量符合引擎能力

- 翻译来源被标记为机器翻译

测试用例 FT-TE-002: 翻译引擎配置测试

- 前置条件: 无

- 测试步骤:

1. 进入设置界面

2. 添加新的翻译引擎配置

3. 输入无效API密钥

4. 测试连接

- 预期结果:

- 连接测试失败并显示适当错误信息

- 无效配置不被保存

**2. 单元测试用例**

**2.1 项目管理模块单元测试**

测试用例 UT-PM-001: 项目创建验证

test('创建项目应生成唯一ID和时间戳', () => {

const project = ProjectManager.createProject('Test Project');

expect(project.projectId).toBeDefined();

expect(project.createdDate).toBeInstanceOf(Date);

expect(project.name).toBe('Test Project');

});

测试用例 UT-PM-002: 文档解析测试

test('应正确解析TXT文档为段', () => {

const mockTxt = "第一段。\n\n第二段。";

const segments = DocumentParser.parseTxt(mockTxt);

expect(segments).toHaveLength(2);

expect(segments[0].sourceText).toBe("第一段。");

});

**2.2 翻译记忆库模块单元测试**

测试用例 UT-TM-001: 记忆库模糊匹配算法

test('应计算正确的模糊匹配率', () => {

const source = "This is a test sentence.";

const target = "This is a test sentence.";

const similar = "This is a test phrase.";

const exactMatch = TranslationMemory.fuzzyMatch(source, target);

const similarMatch = TranslationMemory.fuzzyMatch(source, similar);

expect(exactMatch.score).toBe(100);

expect(similarMatch.score).toBeGreaterThan(70);

expect(similarMatch.score).toBeLessThan(100);

});

```

测试用例 UT-TM-002: TMX导入验证

test('应正确解析标准TMX文件', async () => {

const tmxContent = `<?xml version="1.0"?>

<tmx version="1.4">

<body>

<tu>

<tuv xml:lang="EN"><seg>Hello</seg></tuv>

<tuv xml:lang="ZH"><seg>你好</seg></tuv>

</tu>

</body>

</tmx>`;

const units = await TMXParser.parse(tmxContent);

expect(units).toHaveLength(1);

expect(units[0].sourceText).toBe('Hello');

expect(units[0].targetText).toBe('你好');

});

**2.3 术语库模块单元测试**

测试用例 UT-TB-001: 术语识别测试

test('应识别文本中的术语', () => {

const text = "This computer has a powerful processor.";

const termEntries = [

{ term: "computer", translation: "计算机" },

{ term: "processor", translation: "处理器" }

];

const matches = TerminologyRecognizer.findTerms(text, termEntries);

expect(matches).toHaveLength(2);

expect(matches[0].term).toBe('computer');

expect(matches[0].startPos).toBe(5);

});

测试用例 UT-TB-002: TBX导出验证

test('生成的TBX应符合标准格式', () => {

const termEntries = [

{

term: "software",

translation: "软件",

partOfSpeech: "noun",

definition: "计算机程序和数据"

}

];

const tbx = TBXGenerator.generate(termEntries);

expect(tbx).toContain('<termEntry id="1">');

expect(tbx).toContain('<langSet xml:lang="en">');

expect(tbx).toContain('<term>software</term>');

});

**2.4 翻译引擎模块单元测试**

测试用例 UT-TE-001: 翻译引擎调用验证

test('应正确处理翻译引擎响应', async () => {

const mockEngine = {

translate: jest.fn().mockResolvedValue({

translatedText: "Hola",

detectedSourceLang: "en"

})

};

const result = await TranslationEngineWrapper.translate(

mockEngine, "Hello", "en", "es"

);

expect(mockEngine.translate).toHaveBeenCalledWith("Hello", "en", "es");

expect(result.translatedText).toBe("Hola");

expect(result.sourceLang).toBe("en");

});

**3. 测试策略与执行计划**

**3.1 测试策略**

1. 单元测试:

- 使用Jest测试框架

- 覆盖所有核心业务逻辑

- 目标覆盖率: 80%以上

2. 功能测试:

- 使用Cypress进行端到端测试

- 覆盖所有用户交互流程

- 验证各模块集成功能

3. 性能测试:

- 针对大型文档导入/导出

- 记忆库搜索响应时间

- 术语识别效率

**3.2 测试执行计划**

1. 开发阶段:

- 开发者编写单元测试并与代码同步提交

- 每日构建运行单元测试套件

2. 集成阶段:

- QA团队执行功能测试用例

- 报告缺陷并验证修复

3. 发布前:

- 执行完整回归测试套件

- 性能基准测试

- 用户验收测试(UAT)

**4. 测试数据准备**

**4.1 标准测试文档集**

- 包含各种格式(TXT, DOCX, PDF, HTML)的测试文档

- 不同语言对的翻译文档

- 大型文档(测试性能)和小型文档(测试功能)

**4.2 测试记忆库和术语库**

- 预装标准TMX记忆库(1000+翻译单元)

- 专业领域术语库(如IT、医学、法律)

- 空库(测试初始状态)

**4.3 自动化测试脚本**

// 示例Cypress端到端测试脚本

describe('Translation Workflow', () => {

beforeEach(() => {

cy.login('testuser', 'password');

cy.createProject('Test Project');

});

it('should complete basic translation workflow', () => {

cy.importDocument('test.docx');

cy.openSegment(1);

cy.get('.source-text').should('contain', 'Hello world');

cy.clickMachineTranslate();

cy.get('.translation-suggestion').should('contain', '你好世界');

cy.acceptSuggestion();

cy.get('.target-text').should('have.value', '你好世界');

cy.saveSegment();

cy.exportDocument('test-out.docx');

});

});