Lab2：基于字符命令界面的多文件编辑器

实验目标

本实验在Lab1的基础上要求实现一个**基于命令行的多文件编辑器**，包含**纯文本编辑器**与**XML编辑器**两类，增加**编辑时长统计**、**拼写检查**两大模块。

实验重点考察:

- 在已有架构基础上的扩展能力
- 多态和接口设计
- 横切关注点的模块化处理
- 第三方库依赖隔离与适配器模式应用

文件编码格式：统一使用UTF-8编码

Lab2 与 Lab1 的关系

Lab2 是在 Lab1 基础上的增量开发：

- Lab1 的 18 个命令必须完整实现并保持功能正常
- Lab2 修改 2 个已有命令（`init`、`editor-list`）
- Lab2 新增 7 个命令

评分：Lab2 仅评分新增和修改的功能，但如果Lab1功能有明显缺陷会影响总评。

一、功能需求

1. 编辑器模块扩展

XML编辑器 (XmlEditor)

XML说明:
XML (eXtensible Markup Language) 是一种用于存储和传输数据的标记语言，具有良好的可读性和可扩展性，广泛应用于配置文件、数据交换等场景。

重要约束:

- **本实验要求所有XML元素都必须有唯一的 `id` 属性**，用于命令操作中的元素定位
- 这是为了简化实现。在实际应用中XML元素不一定都有id，但本实验统一要求

功能职责:

- 支持元素级编辑操作：插入元素(insert-before)、追加子元素(append-child)、修改元素ID(edit-id)、修改元素文本(edit-text)、删除元素(delete-element)

- 支持树形结构的可视化输出(xml-tree)
- 支持拼写检查功能：扫描文档文本节点并输出拼写错误报告
- 支持 undo/redo（与文本编辑器相同）

数据结构要求:

- 解析XML文件为**树形结构**(DOM树)
- 内部维护元素节点，并建立 `id -> element` 的映射以支持快速查找
- 保存时序列化回XML格式

建议使用的设计模式:

- 命令模式 (Command)：实现undo/redo功能
- 组合模式 (Composite)：表示XML树形结构
- 适配器模式 (Adapter)：适配XML解析库

XML语法规则:

1. XML声明 `<?xml version="1.0" encoding="UTF-8"?>` 必须写在首行（如需启用日志，见下文说明）
2. 根标签只能有一个，子标签可以有多个，且必须成对存在
3. 标签属性值必须用双引号包起来
4. **每个元素必须有唯一的 `id` 属性**，用于命令操作中的元素定位
5. 本次实验**不支持混合内容**：元素要么只有文本内容，要么只有子元素，不能两者混合
6. 本次实验不考虑注释、自闭合标签等高级特性

混合内容说明:

```
<!-- 正确：只有文本内容 -->
<title id="t1">Hello World</title>

<!-- 正确：只有子元素 -->
<book id="b1">
    <title id="t1">Hello</title>
    <author id="a1">John</author>
</book>

<!-- 错误：混合内容（既有文本又有子元素） -->
<book id="b1">
    Some text here
    <title id="t1">Hello</title>
</book>
```

XML文件示例:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore id="root">
    <book id="book1" category="COOKING">
        <title id="title1" lang="en">Everyday Italian</title>
        <author id="author1">Giada De Laurentiis</author>
        <year id="year1">2005</year>
        <price id="price1">30.00</price>
    </book>
    <book id="book2" category="CHILDREN">
        <title id="title2" lang="en">Harry Potter</title>
        <author id="author2">J K. Rowling</author>
    </book>
</bookstore>
```

日志与XML的兼容:

- XML文件不支持 `# log` 这种注释语法
- 如果需要为XML文件启用日志，在根元素添加特殊属性 `log="true"`

示例:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore id="root" log="true">
    <!-- 内容 -->
</bookstore>
```

程序加载时检测到根元素的 `log="true"` 属性，自动启用日志记录。

2. 统计模块 (Statistics)

核心功能：记录每个文件在当前会话(Session)中的编辑时长，并以可读格式显示。

会话(Session)定义:

- **会话**：从程序启动到退出的一次完整运行周期
- 每次启动程序开始新的会话，所有文件的编辑时长重置为0
- 即使通过工作区恢复打开的文件，编辑时长也从0开始计算

时长计算规则:

- **开始计时**：当文件成为活动文件时（通过 `load` 或 `edit` 命令）
- **停止计时**：当切换到其他文件、关闭文件或退出程序时
- **累计时长**：一个会话中，文件每次成为活动文件都会累计时长
- **重置时长**：文件关闭后，如果再次打开(`load`)，时长重置为0

显示要求:

- 在 `editor-list` 命令中，每个文件名后显示编辑时长

- 使用可读格式：根据时长大小自动选择合适单位

时长格式规范:

时长范围	显示格式	示例
< 1分钟	X秒	`45秒`
1-59分钟	X分钟	`25分钟`
1-23小时	X小时Y分钟	`2小时15分钟`
≥ 24小时	X天Y小时	`1天3小时`

建议使用的设计模式:

- 装饰器模式 (Decorator)：在显示文件列表时，为每个文件名添加时长信息
- 观察者模式 (Observer)：监听文件切换事件，自动更新时长统计

说明:

- 统计模块是相对独立的横切功能，不应与核心编辑功能强耦合
- 如果统计功能失败，应仅提示警告，不影响其他功能继续执行

3. 拼写检查模块 (Spell Checking)

> **核心考察点**：主要考察**架构设计能力**和**第三方库管理能力**，而非算法实现。

选择合适的拼写检查服务，实现对编辑器中的文本内容进行拼写检查，并报告错误。

推荐的拼写检查服务:

- API：https://dev.languagetool.org/public-http-api
- Java：https://dev.languagetool.org/java-api
- Python：`pyspellchecker`库或`language-tool-python`库

说明:

- 可以使用上述推荐的服务，也可以选择其他合适的拼写检查库
- 如使用在线API服务，需考虑网络异常情况的处理
- 拼写检查功能失败时应提示警告，不影响其他功能

考察重点:

1. **依赖隔离**：第三方库依赖被限制在适配器内
2. **接口抽象**：定义清晰接口，编辑器依赖接口而非实现
3. **依赖注入**：依赖从外部传入，而非内部创建
4. **可测试性**：使用Mock对象测试，无需真实库

拼写检查范围:

- **文本文件(.txt)**：检查所有文本内容
- **XML文件(.xml)**：仅检查元素的文本内容，不检查标签名、属性名、属性值

建议使用的设计模式: 适配器模式 (Adapter)

二、 命令设计

命令约定

- 命令默认对**当前活动文件**生效
- 带空格的文本参数使用双引号包裹，文本内容本身不包含双引号
- 所有命令参数区分大小写

命令速查表

在Lab2中，相比Lab1的命令集有以下**新增和变动**:

工作区命令

命令	功能	变动
`init <text|xml> <file> [with-log]`	创建新缓冲区，支持XML	修改
`editor-list`	显示文件列表，支持时长显示	修改

XML编辑命令

命令	功能	适用文件
`insert-before <tag> <newId> <targetId> ["text"]`	插入元素	.xml
`append-child <tag> <newId> <parentId> ["text"]`	追加子元素	.xml
`edit-id <oldId> <newId>`	修改元素ID	.xml
`edit-text <elementId> "text"`	修改元素文本	.xml
`delete-element <elementId>`	删除元素	.xml
`xml-tree [file]`	显示XML树	.xml

拼写检查命令

命令	功能	适用文件
`spell-check [file]`	拼写检查	.txt .xml

注意：Lab1 的 `delete <line:col> <len>` 命令保持不变，XML 的删除命令命名为 `delete-element` 以避免冲突。

2.1 工作区命令（修改）

1. `init` - 创建新缓冲区（修改）

```
init <text|xml> <file> [with-log]
```

功能：创建一个未保存的新缓冲文件，并初始化基础结构。

参数说明:

- `text`：创建纯文本文件
- `xml`：创建XML文件，写入合法的空结构
- `<file>`：文件路径
- `with-log`（可选）：是否启用日志

初始化内容:

创建文本文件（`init text test.txt with-log`）：

```
# log
```

创建XML文件（`init xml config.xml`）：

```
<?xml version="1.0" encoding="UTF-8"?>
<root id="root">
</root>
```

说明：初始化XML文件时，根元素标签名固定为 `root`，id属性固定为 `root`。

创建带日志的XML文件（`init xml config.xml with-log`）：

```
<?xml version="1.0" encoding="UTF-8"?>
<root id="root" log="true">
</root>
```

说明:

- 新缓冲区标记为已修改，需要使用 `save` 命令保存到磁盘
- 创建后自动成为当前活动文件

2. `editor-list` - 显示文件列表（修改）

```
editor-list
```

功能：显示工作区中所有打开的文件及其状态，包括编辑时长。

显示格式（可选以下任一种）：

格式1:

```
* file1.txt [modified] (2小时15分钟)
  file2.xml (45秒)
```

格式2:

```
> file1.txt* (2小时15分钟)
  file2.xml (45秒)
```

说明:

- **当前活动文件标记**：`*` (格式1)或 `>` (格式2)
- **已修改未保存标记**：`[modified]` (格式1)或后缀 `*` (格式2)
- **编辑时长**：括号内显示当前会话中的编辑时长

2.2 XML编辑命令（新增）

1. `insert-before` - 插入元素

```
insert-before <tagName> <newId> <targetId> ["text"]
```

功能：在目标元素前（同级）插入一个新元素。

参数说明:

- `<tagName>`：新插入元素的标签名
- `<newId>`：新元素的唯一ID，不可与已有元素重复
- `<targetId>`：目标元素的ID，新元素将被插入到该元素前
- `"text"`：可选，新元素的文本内容

异常处理:

- `newId` 已存在：提示"元素ID已存在: [newId]"
- `targetId` 不存在：提示"目标元素不存在: [targetId]"
- 尝试在根元素前插入：提示"不能在根元素前插入元素"

示例:

原XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore id="root">
    <book id="book1">
        <title id="title1">First Book</title>
    </book>
</bookstore>
```

执行命令:

```
insert-before book newBook book1 ""
```

结果:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore id="root">
    <book id="newBook"></book>
    <book id="book1">
        <title id="title1">First Book</title>
    </book>
</bookstore>
```

---

2. `append-child` - 追加子元素

```
append-child <tagName> <newId> <parentId> ["text"]
```

功能：在某元素内追加一个子元素（作为最后一个子元素）。

参数说明:

- `<tagName>`：要追加的子元素标签名
- `<newId>`：子元素ID，需唯一
- `<parentId>`：父元素ID
- `"text"`：可选，子元素的文本内容

异常处理:

- `parentId` 无效：提示"父元素不存在: [parentId]"
- `newId` 重复：提示"元素ID已存在: [newId]"
- 父元素已有文本内容：提示"该元素已有文本内容，不支持混合内容"

示例:

```
append-child price price4 book1 "29.99"
```

3. `edit-id` - 修改元素ID

```
edit-id <oldId> <newId>
```

功能：修改某个元素的ID。

参数说明:

- `<oldId>`：原始ID，必须存在
- `<newId>`：目标ID，必须未被占用

异常处理:

- `oldId` 不存在：提示"元素不存在: [oldId]"
- `newId` 已被占用：提示"目标ID已存在: [newId]"
- 尝试修改根元素ID：提示"不允许修改根元素ID"

示例:

```
edit-id book1 book001
```

---

4. `edit-text` - 修改元素文本

```
edit-text <elementId> "text"
```

功能：修改某元素的文本内容。

参数说明:

- `<elementId>`：元素的ID
- `"text"`：新文本内容（可以为空字符串）

异常处理:

- `elementId` 不存在：提示"元素不存在: [elementId]"
- 元素有子元素：提示"该元素有子元素，不支持混合内容"

示例:

```
edit-text title1 "New Book Title"
```

---

5. `delete-element` - 删除元素

```
delete-element <elementId>
```

功能：删除指定ID的元素（包括其所有子元素）。

异常处理:

- `elementId` 不存在：提示"元素不存在: [elementId]"
- 尝试删除根元素：提示"不能删除根元素"

示例:

```
delete-element book1
```

---

6. `xml-tree` - 显示XML树形结构

```
xml-tree [file]
```

功能：以树形结构打印XML文件内容，展示元素的层级关系、属性和文本内容。

参数说明:

- 不指定参数：显示当前活动文件
- `file`：显示指定XML文件

适用对象：仅适用于XML编辑器（`.xml` 文件）

输出格式要求:

- 使用树形字符（`├──`、`└──`、`│`）或缩进表示层级关系
- 显示元素的所有属性（包括id）
- 显示元素的文本内容（如果有）

示例1 - 使用树形字符:

原XML文件:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore id="root">
    <book id="book1" category="COOKING">
        <title id="title1" lang="en">Everyday Italian</title>
    </book>
</bookstore>
```

输出:

```
bookstore [id="root"]
└── book [id="book1", category="COOKING"]
    └── title [id="title1", lang="en"]
        └── "Everyday Italian"
```

示例2 - 使用缩进:

输出:

```
bookstore [id="root"]
  book [id="book1", category="COOKING"]
    title [id="title1", lang="en"]
      "Everyday Italian"
```

说明：两种格式均可，推荐使用树形字符以获得更好的可视化效果。显示类命令不改变文件状态，不进入撤销栈。

2.3 拼写检查命令（新增）

1. `spell-check` - 拼写检查

```
spell-check [file]
```

功能：检查文本文件、xml文件中的拼写错误。

参数说明:

- 不指定参数：检查当前活动文件
- `file`：检查指定文本文件

输出格式参考:

对于文本文件(.txt):

```
拼写检查结果：
第1行，第5列："recieve" -> 建议：receive
第3行，第12列："occured" -> 建议：occurred
```

对于XML文件(.xml)（仅检查元素文本内容）：

```
拼写检查结果：
元素 title1："Itallian" -> 建议：Italian
元素 author2："Rowlling" -> 建议：Rowling
```

说明：XML文件的拼写检查不包括标签名、属性名和属性值，只检查元素的文本内容。

三、评分指南

评分标准

总分：100分

评分项	分值	说明
架构设计	25分	模块划分、接口设计、依赖管理
自动化测试	15分	要求代码分层，每层都有单独的测试，对测试的覆盖率不做要求
命令实现	40分	Lab2新增和修改的命令功能实现的正确性
代码质量	20分	代码结构、可读性、规范性

1. 架构设计 (25分)

需要提供一个简单的文档，给出各个模块的描述以及依赖关系，重点说明相比Lab1的改进。

评分细则：

评分项	分值	要求
模块划分	7分	新增模块职责清晰，与已有模块集成良好
接口设计	6分	接口抽象合理，支持多态（文本/XML编辑器）
设计模式应用	6分	正确使用建议的设计模式，特别是适配器模式
依赖管理	6分	第三方库依赖隔离良好，拼写检查器可替换

2. 自动化测试 (15分)

针对新增模块完成自动测试代码。

3. 命令实现 (40分)

命令功能评分 (共9个命令，40分)：

评分规则：

- **基础分**：40分
- **扣分规则**：每缺少一个命令扣5分
- **命令清单**：共9个命令需要实现（2个修改 + 7个新增）

工作区命令（修改，2个）

- init、editor-list

XML编辑命令（新增，6个）

- insert-before、append-child、edit-id、edit-text、delete-element、xml-tree

拼写检查命令（新增，1个）

- spell-check

评分要点：

- 命令功能实现正确
- 异常处理完善（特别是混合内容检查）
- 边界条件处理正确
- 输出格式符合要求

示例：

- 实现全部9个命令：40分
- 缺少2个命令：40 - 2×5= 30分
- 缺少5个命令及以上将记为0分

4. 代码质量 (20分)

评分细则：

评分项	分值	要求
代码结构	8分	新增代码组织良好，模块化清晰
命名规范	4分	变量、函数、类命名符合规范
代码可读性	4分	代码逻辑清晰，有必要的注释
编码风格	4分	遵循语言编码规范，与Lab1风格一致

四、提交要求

提交时间

ddl：2025年12月15日23:59分

提交方式

每人单独完成，提交内容为一份压缩包，以 `学号_姓名.zip` 格式命名。

提交内容

压缩包应包含以下内容：

1. 源代码

- 所有自己编写的代码文件（包括Lab1和Lab2的完整代码）
- 代码应有清晰的目录结构
- **不要提交**第三方库的源代码

2. 架构设计文档

文档应包含以下章节：

2.1 系统架构

- 模块划分图
- 模块职责说明
- 模块依赖关系

2.2 核心设计

- 设计模式应用说明
- 其他设计相关说明

2.3 运行说明

- 使用的编程语言及版本
- 安装依赖的步骤
- 运行程序的命令
- 运行测试的命令

2.4 测试文档

- 测试用例列表
- 测试执行结果

注意事项

1. **代码必须能正常运行**，确保在提交前测试过
2. **文档必须完整**，缺少必要文档将影响评分
3. **遵守学术诚信**，独立完成，不得抄袭，一旦发现抄袭，两份作业都计0分。
4. **按时提交**，逾期提交将扣分，每逾期一天扣除10%。