1. 概览

本项目是在 Lab1 文本编辑器的基础上扩展出的 **多文件、多类型编辑器**（文本 + XML），核心目标包括：

- 在**不破坏原有架构**的前提下扩展 XML 编辑能力；
- 将 **撤销/重做、日志记录、工作区持久化、编辑时长统计、拼写检查** 等横切关注点模块化；
- 通过 **接口 + 多态** 与多种编辑器类型解耦；
- 通过 **适配器模式** 隔离第三方或可替换组件（如拼写检查器、XML 解析逻辑）。

围绕这些目标，项目中主要使用了以下设计模式：

- 命令模式（Command）
- 组合模式（Composite，用于 XML DOM）
- 观察者模式（Observer）
- 备忘录模式（Memento）
- 接口与多态（面向抽象编程）
- 适配器模式（Adapter，用于拼写检查和可替换的 XML 解析）

下面分别对这些模式做详细分析。

2. 命令模式（Command）

2.1 使用动机

命令行编辑器的核心是“用户输入一行命令，系统执行相应操作并支持撤销/重做”。如果直接在解析器中写大量 `if/else` 或 `switch`，既难以扩展新命令，也难以统一管理 `undo/redo`。

因此，将“编辑操作”抽象为命令对象，有几个好处：

- 把命令解析与执行解耦；
- 所有可撤销操作统一放入撤销栈；
- 新增命令只需新增命令类，不改老代码。

2.2 结构与类关系

- 抽象命令接口： `com.editor.command.Command`

    - 核心方法：`execute()`、`undo()`、`canUndo()`
- 文本编辑命令（Lab1）：
    - `AppendCommand`、`InsertCommand`、`DeleteCommand`、`ReplaceCommand`
- XML 编辑命令（Lab2 新增）：
    - `XmlInsertBeforeCommand`
    - `XmlAppendChildCommand`
    - `XmlEditIdCommand`
    - `XmlEditTextCommand`
    - `XmlDeleteElementCommand`
- 命令调用者（Invoker）：
    - `TextEditor` / `XmlEditor`
        - 内部维护 `undoStack`、`redoStack`
        - 提供 `executeCommand(Command)`，`undo()`，`redo()`
- 命令创建者（解析器）：
    - `CommandParser`
        - 解析字符串命令，实例化对应的 `Command` 子类
        - 交给当前活动 `Editor` 执行

2.3 模式带来的收益

- **扩展性**：Lab2 新增的 6 个 XML 命令，都以新的 `Xml*Command` 类的形式加入，不用修改编辑器的核心撤销/重做逻辑。
- **统一的撤销/重做机制**：文本命令和 XML 命令共用一套 `undo/redo` 管理，命令只需关心自身逻辑。
- **利于日志与统计**：命令执行与日志记录/统计分离；命令执行完成后只需触发事件即可，由观察者统一处理日志。

相对于 Lab1，Lab2 的变化是：**命令的覆盖范围从“文本操作”扩展到“XML 树操作”**，但原有命令框架无须重构，体现了命令模式在扩展上的优势。

3. 组合模式（Composite）——XML DOM 树

3.1 使用动机

XML 本质上是一个树形结构，节点之间是**整体–部分**的层次关系。我们希望：

- 用统一结构表示“单个节点”和“包含子节点的子树”；
- 方便实现 `insert-before`、`append-child`、`delete-element` 等树操作；

- 方便实现 `xml-tree` 这样的树形可视化输出。

3.2 结构与类关系

- 组件类（Component / Composite）：
  - `com.editor.editor.XmlElement`
    - 字段：`tagName`、`id`、`attributes`、`textContent`、`children`、`parent`
    - 操作：
      - 结构修改：`addChild(XmlElement)`、`removeChild(XmlElement)`、`insertBefore(...)`
      - 状态判断：`hasChildren()`、`hasTextContent()`、`hasMixedContent()`
- 根节点持有者：
  - `XmlEditor`：内部维护 `XmlElement root` 作为整棵 DOM 树的入口
  - 同时维护 `id -> XmlElement` 索引表，支持 `getElementById(id)`

XML 编辑命令直接对 `XmlElement` 树操作，例如：

- `insert-before`：在 `parent.insertBefore(newChild, refChild)` 上实现；
- `append-child`：在 `parent.addChild(newChild)` 上实现；
- `delete-element`：在 `parent.removeChild(element)` 上实现。

`xml-tree` 命令则通过递归遍历 `XmlElement` 树，输出树形结构。

3.3 模式带来的收益

- **统一处理单个节点与子树**：命令类不需要区分“叶子节点”和“中间节点”，都作为 `XmlElement` 操作。
- **树操作直观**：新增/移动/删除节点都围绕 `parent/children` 展开，逻辑清晰。
- **显示逻辑简单**：`xml-tree` 输出只需一个递归函数，利用 `children` 关系就能完成整棵树的展示。

4. 观察者模式（Observer）

4.1 使用动机

系统中有多个“横切关注点”：

- **日志记录**：命令执行的记录；
- （潜在）监控/统计：后续也可能基于事件做统计。

如果把这些逻辑直接嵌入命令或编辑器，会高度耦合且难以扩展。观察者模式的目标是：

- 将“事件源”（命令执行、文件操作）与“响应者”（日志模块）解耦；
- 支持按文件粒度开启/关闭日志。

4.2 结构与类关系

- 抽象接口：
  - `com.editor.observer.Subject`： `attach(Observer)`、`detach(Observer)`、`notifyObservers(Event)`
  - `com.editor.observer.Observer`： `update(Event event)`
  - `Event`：封装事件类型（`LOAD`、`INIT`、`EDIT`等）、命令字符串、关联文件路径。
- 被观察者（Subject）：
  - `Workspace`：在`load/init/save/close/edit`等操作后触发事件；
  - `TextEditor` / `XmlEditor`：在`append/insert/delete/replace/undo/redo`或 XML 命令执行后触发`EDIT`事件。
- 观察者（Observer）：
  - `Logger`：实现`Observer`接口，订阅`Workspace`和各个`Editor`，基于事件写日志。

4.3 模式带来的收益

- **解耦**：命令和编辑器无需关心日志逻辑，只需触发事件；日志模块也不需要了解命令内部细节。
- **灵活控制订阅关系**：可以按需为某个文件的编辑器`attach/detach``Logger`，配合`log-on/log-off`与 XML 根元素`log="true"`实现自动/手动日志控制。
- **可扩展性**：未来若需要添加统计或监控，只需添加新的`Observer`，不用修改现有`Subject`代码。

相比较 Lab1，Lab2 在此基础上扩展为同时监听 **文本编辑器 + XML 编辑器 + 统计模块事件**，但核心 Observer 结构保持不变。

5. 备忘录模式（Memento）

5.1 使用动机

实验要求支持 **工作区恢复**：

- 程序退出前保存当前打开的文件、活动文件、日志状态等；
- 下次启动时自动恢复上一次会话的工作区。

直接序列化整个`Workspace`或所有`Editor`会破坏封装，且不够灵活。备忘录模式的目的在于：

- 在不暴露内部细节的前提下保存必要状态；
- 控制存储内容的粒度（只存“工作区元信息”，不存具体文本/XML 内容）。

5.2 结构与类关系

- 备忘录类（Memento）：
  - `com.editor.memento.Memento`
  - 保存的信息：
    - 打开的文件列表： `openFiles`
    - 当前活动文件： `activeFile`
    - 修改标记： `modifiedStatus`
    - 日志开关状态： `logStatus`
- 发起者（Originator）：
  - `Workspace`
    - `createMemento()`：根据当前状态创建 `Memento`
    - `restoreMemento(Memento)`：根据 `Memento` 恢复状态
- 管理者（Caretaker）：
  - `Workspace.saveWorkspace()`：将 `Memento` 序列化到 `.editor_workspace`
  - `Workspace.loadWorkspace()`：程序启动时读取 `.editor_workspace` 并恢复

5.3 模式带来的收益

- **封装性好**：外部代码不需要也无法直接操作 `Workspace` 内部状态，只能通过 `Memento` 间接保存/恢复。
- **与 Lab2 扩展兼容**：即使编辑器从单一 `TextEditor` 扩展为多态 `Editor`（含 `XmlEditor`），Memento 只关心“文件级别状态”，不需要变化。

6. 接口与多态设计（Editor 抽象）

6.1 使用动机

Lab2 相比 Lab1 的关键变化是：**从“单一文本编辑器”扩展为“多种类型编辑器”**。为了避免在工作区、命令解析器中处处写 `if (endsWith .txt/.xml)`，需要抽象一个统一的编辑器接口。

6.2 结构与类关系

- 编辑器接口：

    - com.editor.editor.Editor （实现 Subject ）
    - 统一能力：
        - getFilePath()
        - isModified()/setModified()
        - executeCommand(Command)
        - undo()/redo()/canUndo()/canRedo()
        - save()
        - 类型判断： isTextEditor() 、 isXmlEditor() （默认基于 instanceof ）
- 具体实现：
    - TextEditor ： 专注于行文本编辑，提供 getLines() 、 show() 、 append/insert/delete/replace 等。
    - XmlEditor ： 专注于 XML DOM 编辑，提供 getRoot() 、 getElementById() 、 toXmlString() 、 loadFromFile() 等。
- 使用方：
    - Workspace ：
        - 内部使用 Map<String, Editor> 管理所有文件的编辑器
        - loadFile/initFile/saveFile/closeFile/setActiveFile 均针对 Editor 编程
    - CommandParser ：
        - 在 undo/redo/save/log-on/log-off 等通用命令中仅依赖 Editor 接口；
        - 在文本特有命令（ append/insert/delete/replace/show ）中，通过 editor.isTextEditor() 再强转为 TextEditor ；
        - 在 XML 特有命令（ insert-before/append-child/... ）中，通过 editor.isXmlEditor() 再强转为 XmlEditor 。

6.3 模式带来的收益

- **统一管理多种编辑器**： Workspace 不再区分“文本文件/ XML 文件”，只管理 Editor ，大大减少条件分支。
- **可扩展性**：未来如果要增加 Markdown 编辑器、JSON 编辑器，只需实现新的 Editor 子类，并在 load/init 时选择合适的实现即可。
- **与命令模式自然结合**： executeCommand/undo/redo 由接口统一定义，命令层可以透明地作用于不同类型编辑器。

7. 适配器模式（Adapter）

7.1 拼写检查适配器

使用动机

拼写检查功能理论上应依赖一个外部库（例如 Hunspell），但实验要求中强调“第三方库依赖隔离与适配器模式应用”，即：

- 上层只依赖一个稳定的、与业务贴近的接口；
- 底层可以替换成不同的实现（简单自实现 / 专业库）。

结构与类关系

- 目标接口（Target）：
  - `com.editor.spellcheck.SpellChecker`
    - `List<SpellError> checkSpelling(String text)`
    - `String getSuggestion(String word)`
- 适配器实现（Adapter）：
  - `SimpleSpellChecker`
    - 内部使用一个简易英文词典集合 `Set<String> dictionary`
    - 使用编辑距离（Levenshtein）算法生成建议
- 错误实体：
  - `SpellError`： 包含错误单词、行号、列号、建议拼写
  - `XmlSpellError`： 用于 XML，附加元素 ID 信息
- 使用者：
  - `CommandParser` 中的 `spell-check` 命令：
    - 对 `.txt` 文件：将全文拼接为字符串交给 `spellChecker.checkSpelling`
    - 对 `.xml` 文件：遍历 `XmlElement` 的文本内容，逐段检查，将结果映射为“元素 ID + 错误词 + 建议”

模式收益

- **隔离实现细节**：命令解析器完全不知道拼写检查实现细节，只依赖 `SpellChecker` 接口。
- **便于替换实现**：如果将来要接入真正的拼写库，只需再写一个实现 `SpellChecker` 的类即可，无需修改命令或编辑器代码。

7.2 XML 解析逻辑的适配思路

当前 `XmlEditor` 里实现了一个轻量级的 XML 解析与序列化逻辑，直接基于字符串和正则表达式。虽未直接接入外部库，但接口设计已做好“适配层”的准备：

- 对上暴露的仍是 `XmlElement`（DOM 视图）和 `loadFromFile()/toXmlString()` 等方法；
- 若将来改用标准 DOM/SAX 库，只需在 `XmlEditor` 内部写一层“解析结果 -> XmlElement 树”的适配代码，上层命令与工作区无须感知。

这体现了“预留适配器位置”的架构意识。

8. 横切关注点：编辑时长统计模块

8.1 模块职责

- `com.editor.statistics.Statistics` 负责：
  - 记录每个文件在当前会话中被编辑的总时长（基于毫秒）；
  - 提供人类可读的格式（秒/分钟/小时）；
  - 在文件成为活动文件、关闭文件时更新计时。

8.2 与 Workspace 的协作

- `Workspace` 内持有一个 `Statistics` 实例：
  - 在 `load/init/setActiveFile/closeFile` 中调用：
    - `statistics.onFileActivated(file)`
    - `statistics.onFileClosed(file)`
  - 在 `editor-list` 命令中调用：
    - `workspace.getStatistics().getFormattedEditTime(file)` 拼接到输出。
- `App` 在收到 `exit` 时调用：
  - `workspace.getStatistics().stopAll()`，确保会话结束时停止所有计时。

8.3 模式角度的评价

虽然编辑时长统计没有直接使用 GOF 中的命名模式，但其实现体现了如下设计思想：

- **横切关注点模块化**：统计逻辑被封装在独立模块 `Statistics` 中，而不是分散在各个命令或编辑器内部。
- **单一职责**： `Workspace` 负责“什么时候开始/停止计时”， `Statistics` 负责“如何计时和格式化显示”。

这与观察者模式一起，共同体现了对横切关注点的良好分层与解耦。

9. 与 Lab1 的对比与演进

从设计模式角度看，Lab2 在 Lab1 基础上的主要演进点包括：

- **命令模式保持不变但应用范围扩大**：
  - Lab1：只覆盖文本编辑命令；
  - Lab2：扩展到 XML 编辑命令， `Editor` 接口保证命令栈机制可复用。
- **观察者模式复用并增强**：
  - Lab1：主要用于命令执行日志；
  - Lab2：同时监听文本和 XML 编辑事件，并与日志自动开关（ `# log` 、 `log="true"`) 结合。
- **备忘录模式继续承担工作区持久化**：
  - 保存的仍是“工作区元信息”，对新增的 XML 编辑器透明。
- **新增多态抽象与组合模式**：
  - 通过 `Editor` 接口将文本/XML 编辑统一在一套工作流中；
  - 通过 `XmlElement` + 组合模式，引入完整的 XML DOM 支持。
- **新增适配器思路**：
  - 通过 `SpellChecker` 接口隔离拼写检查实现；
  - `XmlEditor` 内部结构预留第三方 XML 库接入点。

整体上，Lab2 在延续 Lab1 架构风格的基础上，通过合理使用设计模式，实现了 **“在已有架构基础上的自然扩展”**，满足了多类型编辑器、横切关注点模块化以及第三方库可替换性的实验要求。