RegMiner X：高效挖掘 Git 仓库回归测试元数据集的自动化软件
设计说明书

1、引言

1.1 编写目的

本文档的目的是详细描述“RegMiner X：高效挖掘 Git 仓库回归测试元数据集的自动化软件”的设计与实现，并为使用本软件的软件测试工程师或科研人员提供详细的目录架构介绍以及使用指导。该软件本质为数据挖掘与大型数据集构建软件，目标是自动化从代码仓库中挖掘和匹配生产 – 测试变更对元数据集，弥补了传统 Git 软件无法直接挖取和匹配生产 – 测试回归对的不足，为软件测试领域内“回归测试”子领域的研究者提供充足的高质量生产 – 测试回归变更对，从而在元数据集层面为回归测试的研究和应用提供有力的数据集保障。

1.2 背景

回归测试是软件测试领域重要的研究课题之一。随着软件开发的不断迭代和复杂性增加，人工维护回归测试用例的开销较大，回归测试的自动化需求逐步提高。然而，无论是进行理论层面的回归测试研究还是进行实证研究，研究者和软件测试工程师都需要大规模、高质量的“生产代码变更 – 对应的测试代码变更”回归对数据集作为研究基础。现有的版本控制软件（如 Git）并不具备直接从代码仓库挖掘和匹配“生产 – 测试”回归对的功能，因而此前的数据挖掘工作大多由人工完成，耗费巨大的时间成本且难以保障匹配的精准性。现有的公开回归测试数据集无论在数量、质量、规模上都无法满足进一步的研究需求。因此，“RegMiner X：高效挖掘 Git 仓库回归测试元数据集的自动化软件”应运而生。本软件适用于任何基于 Git 的代码仓库，可以挖掘任何由 Git 版本控制工具管理的代码仓库，生成大规模、高质量的“生产 – 测试”回归变更对的标准 JSON 数据集。为回归测试领域软件测试工程师、科研工作者等提供有力的数据集支持。此外，元数据集规模的突破也为后续进一步从元数据中提取更细粒度的变更特征、把机器学习引入回归测试领域打下坚实基础。

2、软件概述

2.1 软件功能概述

软件的主要功能如下：从远程代码仓库自动克隆 Git 代码仓库到本地设备、自动遍历和智能分析 commit 历史记录、完成“生产 – 测试”对的命名规则匹配、挖掘两种不同逻辑形式的“生产 – 测试”回归变更对（分别对应下游不同的分析研究任务）。同时此软件内部可以通过 java 的词法分析器解析 java 文件的结构并生成 AST 语法生成树，可以进一步分析 java 文件的内部结构特征辅助分析。最终，RegMiner X 会输出大规模的“生产 – 测试”回归变更对数据集，以 JSON 的形式存储“生产 – 测试”变更各自的 commitID、相对路径及

变更的时间信息。

2.2 软件主要模块概述

- **Git 基础操作模块**：负责与 Git 仓库交互，包括克隆仓库、分析结构、分析提交历史等。
- **遍历 commit 历史模块**：遍历所有 commit 历史，发现全部可能构成回归对的基本元素。
- **命名规则匹配模块**：为生产代码在代码仓库中精准地定位到与之相关的测试代码文件。
- **回归匹配模块 I**：匹配出第一种形式的“生产 – 测试”回归变更对。
- **回归匹配模块 II**：匹配出第二种形式的“生产 – 测试”回归变更对。
- **词法解析模块**：使用词法解析器为 java 文件生成 AST 抽象语法树，细粒度分析结构。
- **数据集导出模块**：生成并导出大规模的、包含“生产 – 测试”回归变更对的 JSON 数据集。

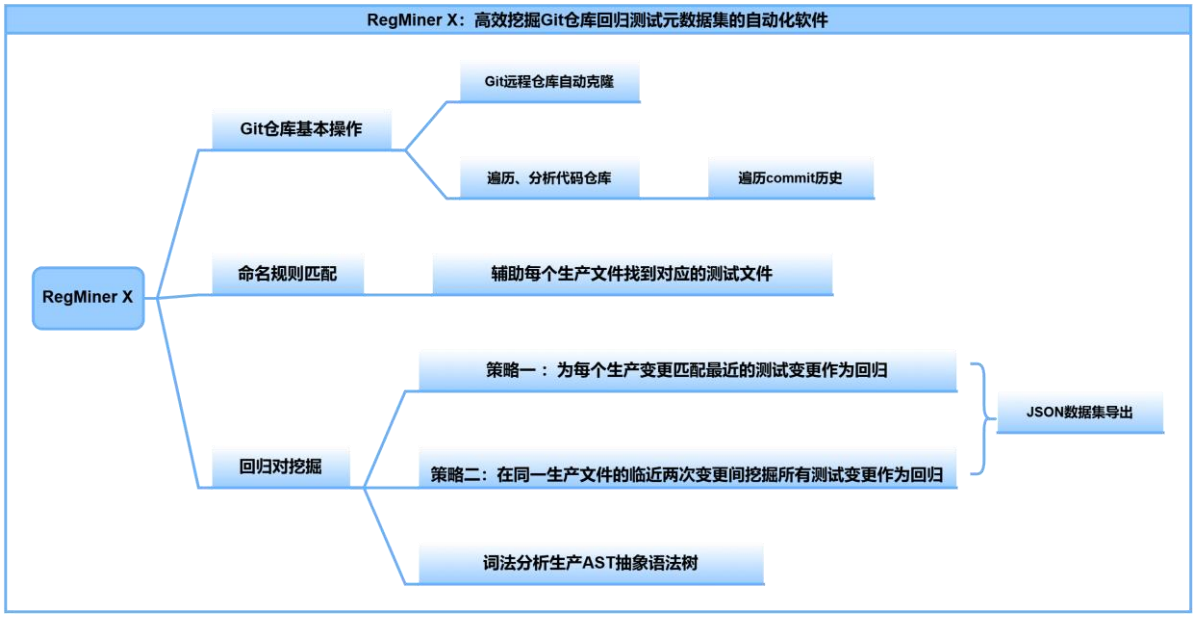

图 1： 软件功能总框图

2.3 技术栈

编程语言： Java

框架： Maven

数据输入/输出格式：JSON

Git 辅助操作库： JGit（Java）

2.4 目标用户

本软件兼备科研属性与应用属性。目标用户为有一定 Java 语言了解的软件测试工程师、软件测试科研工作者以及热衷于利用大规模数据进行测试实证研究、引入机器学习的机器学习工程师。

3.软件详述

本软件设计遵循高内聚、低耦合的原则，采用多软件包架构。高内聚确保每个模块聚焦于单一任务，内部逻辑紧密，易于理解和维护；低耦合则通过定义清晰的接口，使模块之间的依赖最小化，避免了复杂的相互影响，从而提高了软件的可扩展性和可维护性。通过将软件划分为多个独立的软件包，每个包负责特定的功能，模块之间通过简洁的调用相互协同。这种设计不仅便于独立开发与部署，还使得未来功能扩展变得更加灵活，使得本软件可以在未来进行进一步的开发拓展。

3.1 Git 基本功能模块

Java 的 JGit 库是一个开源的轻量级 Git 实现，允许开发者通过 Java 语言与 Git 仓库进行交互。JGit 支持大多数常见的 Git 操作，如克隆仓库、获取提交历史、检查文件状态等，适用于需要在 Java 环境中嵌入 Git 功能的应用场景。在本软件中，调用了 JGit 的 jar 包辅助软件进行 Git 仓库的挖掘与探索，实现根据配置文件信息自动克隆目标仓库到本地指定路径的功能。配置文件中存放着目标仓库的信息，

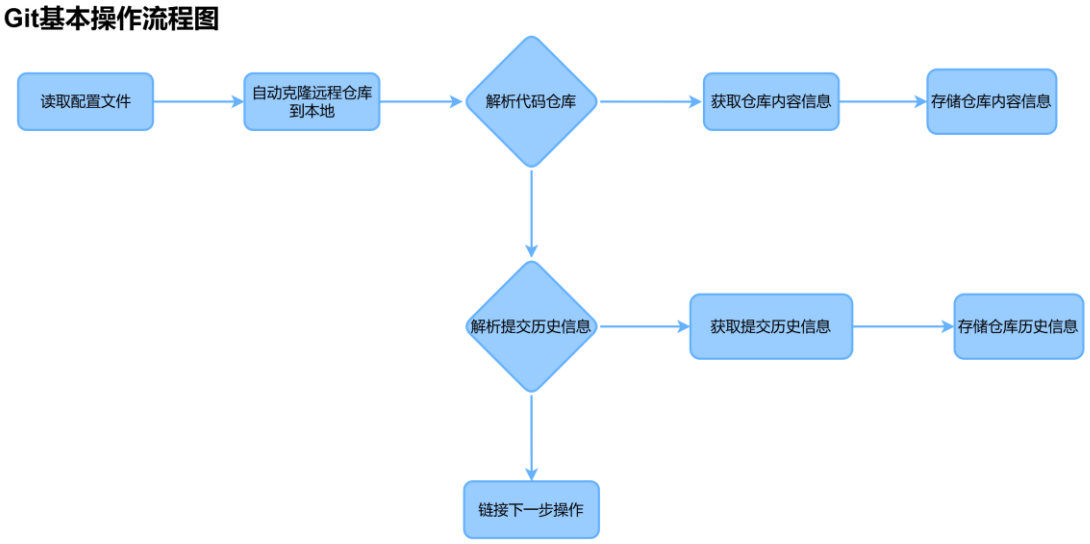

图 2：Git 基本操作模块功能流程示意

```java
package Resource;

public class Resource {
    public static String projectName ="commons-compress";//此处替换成目标仓库的名称;
    public static String outputDir ="./Feature";  1 个用法
    public static String gitrepository ="java_data";  1 个用法
    public static String branchName ="master";  1 个用法
    public static String commitInfo="./Commits/";  1 个用法
    public static String allFilesDirctory="./AllFiles/";  0 个用法
    public static String githubToken="";//此处填入自己的github令牌，免密链接  1 个用法
    public static String remotePath="";//此处放入远程仓库的URL  1 个用法
}
```

图 3：配置文件结构示例，内含必要的目标仓库定位信息

JGit 库的引入，使得本软件具备自动克隆目标仓库、分析代码仓库、遍历和分析 commit 历史的功能，为后续挖掘“生产 – 测试”回归变更对提供最基本的 Git 操作支持。

3.2 遍历 commit 历史

在版本控制系统的软件仓库中，每一次代码提交都对应着唯一的哈希值作为 commitID，而更改文件的相对路径、更改文件的系统时间也是我们在软件仓库中定位到文件某次具体更改的关键信息。软件在运行的入口处会首先完整地遍历一遍代码仓库，将历史中所有出现过变动的文件（任何形式的更改）存入 HashSet 作为一级缓存；将每次 commit 的 commitID 和对应修改的文件路径列表分别作为键和值存入 HashMap 作为二级缓存，大幅缩短后续遍历匹配所需时间。于此同时，相关信息会同步输入日志中，供用户检查验证。

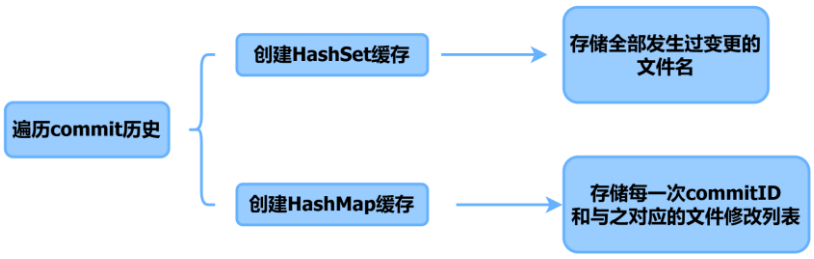

图 4：为后续遍历做的缓存策略，降低软件遍历总耗时

注意：在庞大的代码仓库中精准定位某个文件的某次更改需要以下信息：

- prod_path：生产代码在仓库中的相对路径
- test_path：测试代码在仓库中的相对路径
- prod_time：生产代码的变更时间
- test_time：与生产代码最近的一次测试代码的变更时间
- pro_commitID：生产代码变更的 commitID
- test_commitID：测试代码变更的 commitID
- project：项目仓库名称

上述六条信息是在庞大的代码仓库中定位生产-测试对的关键信息，也是本软件后续挖

掘到“生产 – 测试”回归变更对后要提取的信息，作为数据集进行输出。

3.3 命名规则匹配：

根据标准 java 项目的开发原则和目录规则，对应生产类 UserService 的测试类通常命名为 UserServiceTest。对应生产类 OrderProcessor 的测试类通常命名为 OrderProcessorTest。对于路径匹配而言，如果已知生产代码文件在代码仓库中的具体相对路径 src/main/java/com/company/project/service/UserService.java，则其对应的测试代码文件通常存放在 src/test/java/com/company/project/service/UserServiceTest.java 路径下。根据此命名规则，RegMiner X 可以轻松根据“生产代码”匹配到对应的“测试代码”，为观察两者的回归和共同进化打下了坚实的基础。此外，为了使 RegMiner X 具有更强的鲁棒性，软件同时支持多种常见的命名匹配规则，使 RegMiner X 在不同的代码仓库下皆能正常工作。

3.4 第一种挖掘回归对的策略

软件会重新从头遍历代码仓库的每一次 commit，寻找到其中全部的生产代码更改。一旦软件检测到了某次 commit 中存在某个生产文件的更改，RegMiner X 会根据命名匹配规则自动推理出其测试文件的命名名称，并在此次 commit 及接下来的所有 commit 中寻找此测试文件的修改操作。第一次找到的测试更改即和此次生产变更构成一个回归的测试对，提取此回归对的如下信息作为数据集。

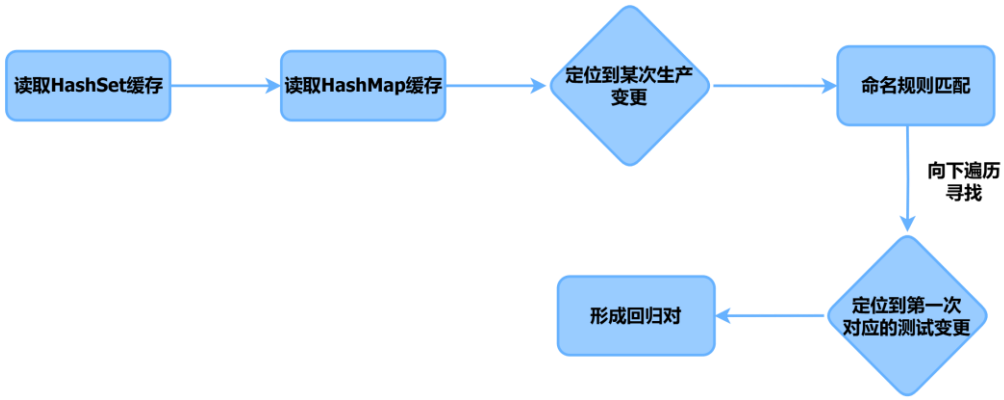

图 5：第一种策略下的回归对挖掘流程


```
{
    "prod_path": "src/main/java/org/cactoos/io/Output.java",
    "test_path": "src/test/java/org/cactoos/text/NotNullOutputTest.java",
    "prod_time": "2017-05-24 04:53:22",
    "test_time": "2017-06-03 20:14:09",
    "test_commitID": "d0fe41e9b9cee89883aa2ad3817a8b11653e83b9",
    "project": "cactoos",
    "isfound": "found test change",
    "pro_commitID": "a8100f2f12d95aa82dee5fdd7eb6d8fb9042ef81"
}
```


图 6：第一种策略挖掘到的回归变更对的形式示例

- prod_path；生产代码在仓库中的相对路径

- test_path：测试代码在仓库中的相对路径
- prod_time：生产代码的变更时间
- test_time：与生产代码最近的一次测试代码的变更时间
- pro_commitID：生产代码变更的 commitID
- test_commitID：测试代码变更的 commitID
- project：项目仓库名称

以上信息已足够精准在代码仓库中定位此次生产 – 测试变更对的全部信息

3.5 第二种挖掘回归对的策略

软件会重新从头遍历代码仓库的每一次 commit，寻找到其中全部的生产代码更改。一旦软件检测到了某次 commit 中存在某个生产文件的更改，RegMiner X 会首先向下继续遍历 commit，寻找同一个生产文件的下一次更改。两次 commit 会形成一个夹在中间的区间，此时根据命名匹配规则自动推理出其测试文件的命名名称，并在这个区间内重新遍历每一个 commit，将其中含有的所有对应测试文件变更悉数提取出来，每个都与该生产文件的首次更改构成一个回归的“生产 – 测试”变更对。 同样提取此回归对的如下信息作为数据集。

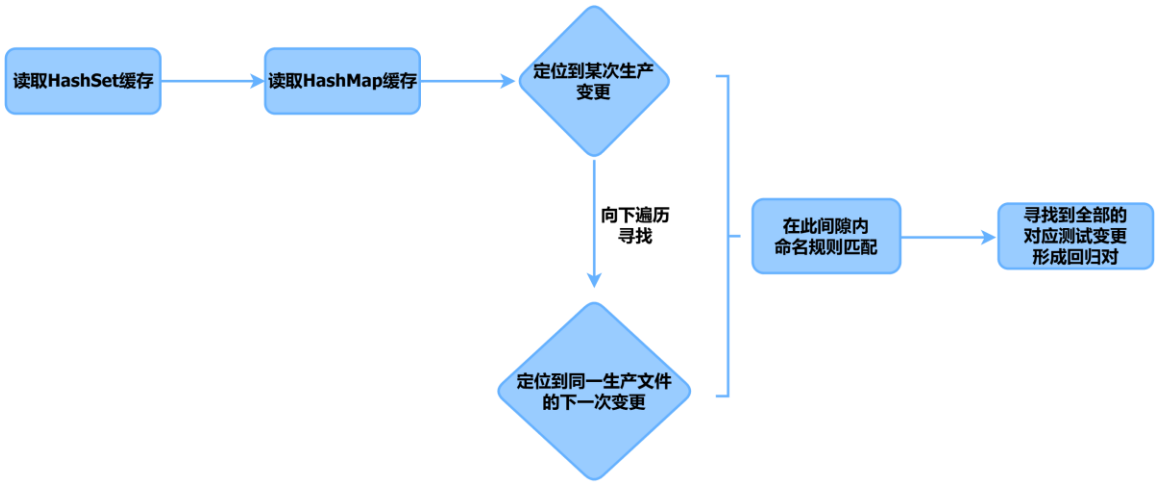

图 7：第二种策略下的回归对挖掘流程


```
{
    "prod_path": "src/main/java/org/cactoos/io/InputAsBytes.java",
    "test_path": "src/test/java/org/cactoos/io/InputAsBytesTest.java",
    "prod_time": "2017-05-24 09:05:17",
    "test_time": "2017-05-24 12:30:00",
    "test_commitID": "6168af91ef7fed4d3a28dd271cc66ebbe0364b03",
    "project": "cactoos",
    "isfound": "found test change",
    "pro_commitID": "8874f5c0cfd5dacbba9922fbb38ded1101923b29"
}
```


图 8：第二种策略挖掘到的回归变更对形式示例

3.6 词法解析模块：

在根据生产代码的 commitID，path 和提交时间等信息具体定位到生产代码后，软件提取生产代码变更前、变更后的具体内容，通过词法解析器生成 AST 抽象语法树。

抽象语法树 (AST) 是一种以树形结构表示程序源代码语法结构的形式，是源代码的抽象语法层次的表示。它是编译器、解释器、静态代码分析工具以及许多程序分析工具中重要的核心数据结构。为更改前、更改后的文件形成抽象语法树，可以在更细粒度的层面了解文件变更的内在特征，为进一步分析回归对的特征提供更高级的特征数据。

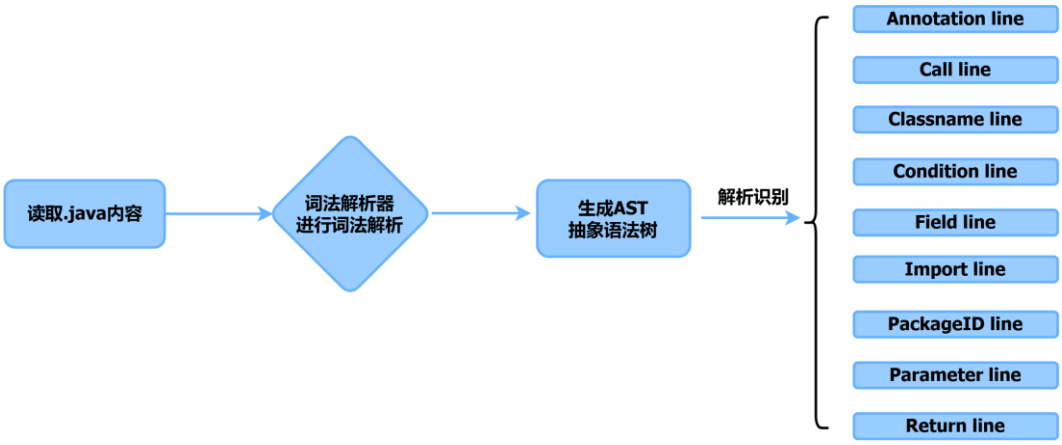

图 9：细粒度解析 java 文件内容的功能流程

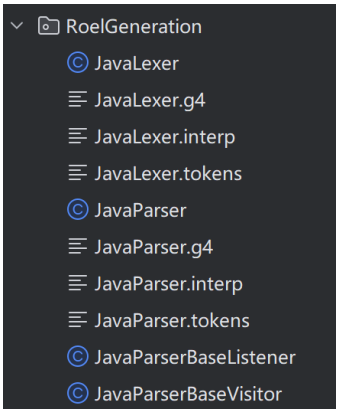

图 10：软件中词法分析器的相关部分

3.7 数据集汇总和导出模块

遍历完整个代码仓库和全部生产 – 测试变更对后，全部挖掘到的特征数据集会以每个变更对一个 JSON 的形式统一输出到输出目录中。形成大规模的数据集，可以供下游的回归测试任务进行使用。单个数据对已在图 4、图 5 做过展示。下面展示的是 RegMiner X 同挖掘多个仓库后的输出目录，每个目录对应着一个完成的 Git 代码仓库，里面存放着对应代码仓库中挖掘到的全部“生产 – 测试”回归变更对，可以作为下游任务的元数据进行使用。

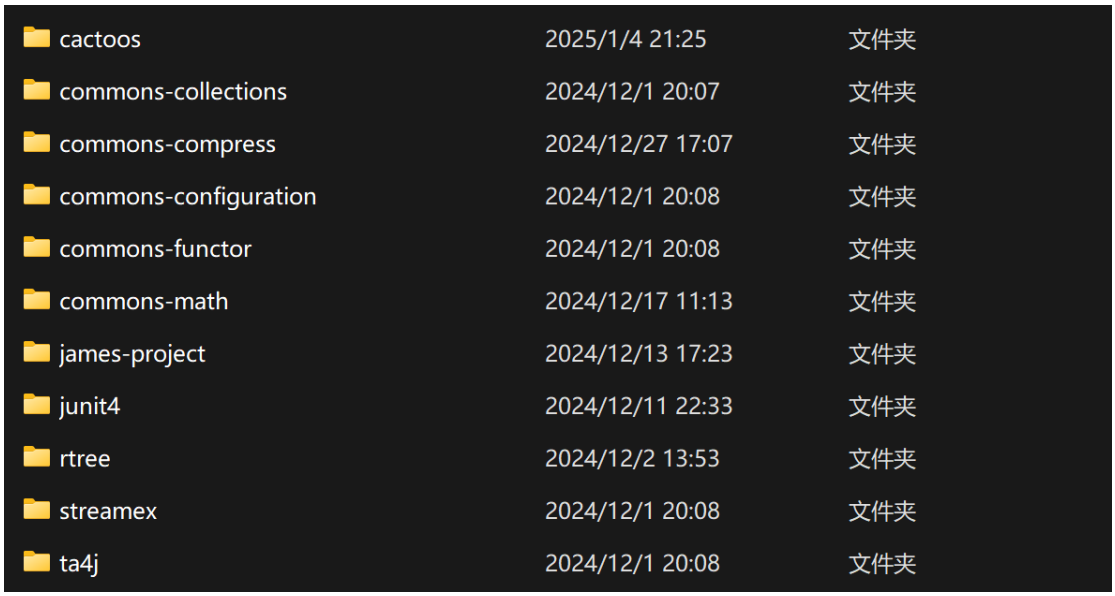

图 11：最终数据集输出目录的示例，每个目录都对应一个完整的仓库数据集

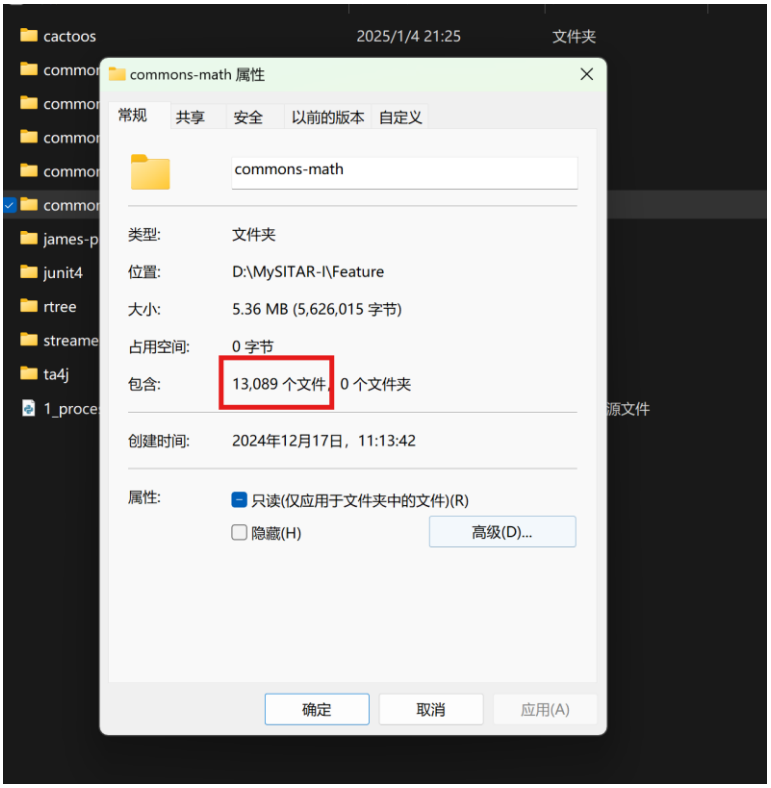

图 12：单个项目的数据集规模从几千到几万对不等

请注意，最终挖掘到的数据集中具体的对数与仓库规模和仓库特征有关。作为一个大规模数据集的挖掘软件，每个仓库的挖掘结果通常从**几千到几万对不等**。然而此软件适用于所有的 Git 软件仓库，所有软件仓库都可以作为“RegMiner X：高效挖掘 Git 仓库回归测试元数据集的自动化软件”的适用对象。因此可以构建数量更为庞大的跨项目数据集。

4、目录架构和软件包功能的介绍：

为了更好地帮助目标用户熟悉 RegMiner X 的内部架构，更熟悉 RegMiner X 的使用方法和使用逻辑，下面介绍 RegMiner X 的具体目录架构以及对应的作用。

4.1、一级目录：

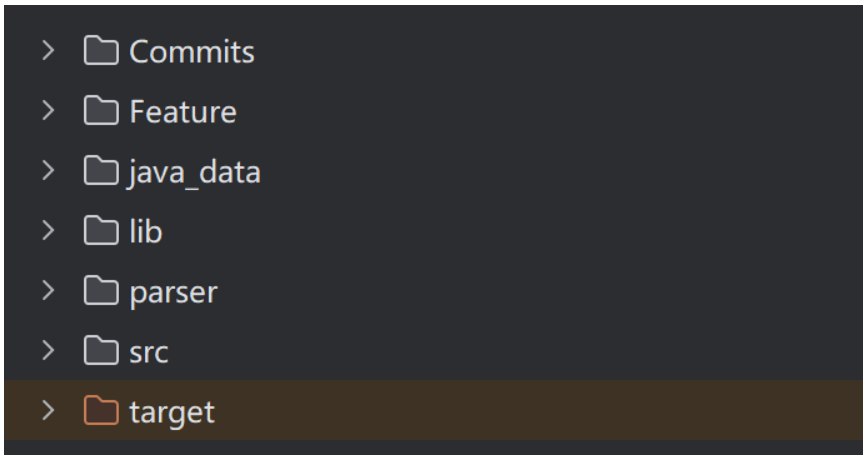

图 13：RegMiner X 的一级目录架构

- **Commits**
  为遍历 Git 仓库做的 commit 缓存，存放 commit 信息，无需人为操作
- **Feature**
  最终输出的数据集会存放在 Feature 目录内
- **java_data**
  从远端克隆仓库时，克隆下来的本地仓库会被存放在 java_data 目录中
- **lib**
  存放部分第三方 jar 包，以供用户直接使用
- **src**
  源码文件夹
- **target**
  Maven 构建工具生成的输出目录，包含编译后的代码、JAR 文件和其他构建产物。

4.2 二级目录（src 目录）：

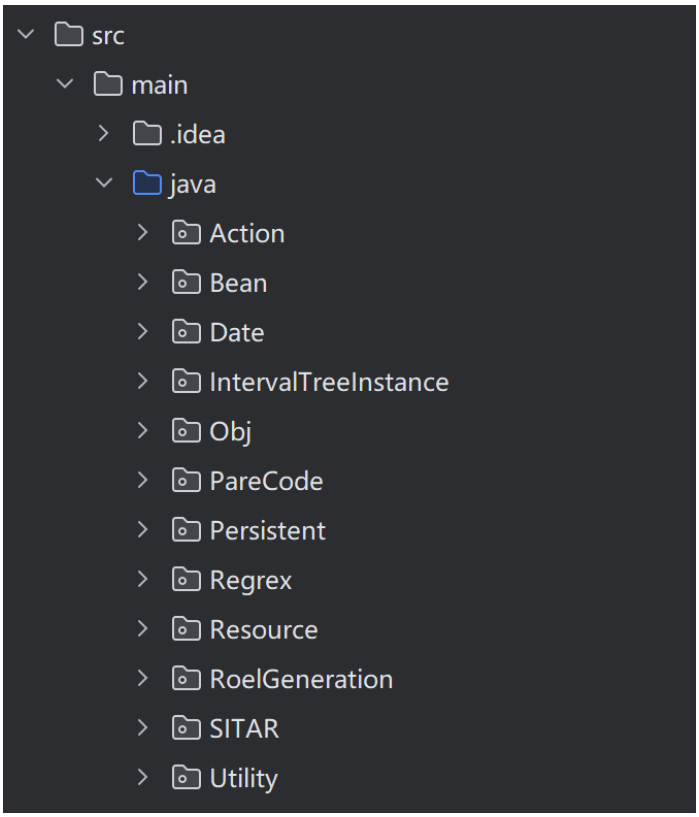

图 14：src 目录下的目录架构

- **Action**
  包含用于生成 AST 抽象语法树的 ASTGenerator.java。
  用于 Git 基本操作的 GitAdapter.java。
- **Bean**
  包含 JSON 与 java 对象序列化和反序列化的必要数据类。
- **Date**
  包含提取 commit 时间，处理变更对间隔时间的类。
- **IntervalTreeInstance**
  含有区间树的定义，是“穿刺”AST 获取细粒度变更特征的重要数据结构。
- **Obj**
  包含了每个 java 文件映射到的工具类。
- **PareCode**
  内含词法分析器会用到的辅助类。
- **Persistent**
  内含序列化和反序列化的代码，关乎信息的读入和数据集最后的导出。
- **Regex**
  存储要用到的正则表达式，主要用于名称匹配和命名规则的检验。
- **Resource**
  内含配置文件。是运行 RegMiner X 所需配置的配置目录。
- **RoelGeneration**
  词法分析器必要的 lexer 分析器。
- **SITAR**

内含 RegMiner X 的主入口，主逻辑。串联起整个挖掘逻辑。

- **Utility**

  通用工具类，内含大量辅助函数，如字符串处理、集合操作等。

5.总结

RegMiner X 是一款专为回归测试场景设计的数据挖掘软件工具，旨在高效搭建大规模、高质量的回归对元数据的数据集。回归测试是软件开发与维护过程中不可或缺的环节，其核心目的是确保新代码的引入或现有代码的修改不会对已有功能造成破坏。然而，构建可靠的数据集常常需要耗费大量的人力与时间。RegMiner X 通过自动化的方式，从实际软件项目中挖掘 commit 历史，快速匹配涵盖生产代码与测试代码的关联变更样本，从而为软件测试的实证研究部分提供数据支持。

该软件适用于软件测试工程师和软件测试工作者，能够显著提升测试数据构建的效率和质量。RegMiner X 不仅能帮助测试团队快速定位可能的问题区域，还为科研工作者提供了一种便捷的手段，支持回归测试、代码共演分析等相关领域的研究。通过生成高质量的细粒度回归元数据集，RegMiner X 为提升测试覆盖率、降低软件维护成本、促进软件工程领域研究提供了强有力的支持。