RegFeatMiner：为回归测试搭建大规模机器学习降噪特征集的数据挖掘软件
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“RegFeatMiner：为回归测试搭建大规模机器学习降噪特征集的数据挖掘软件”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。该软件的目标是自动化挖掘构建大规模低噪声的回归测试数据集，通过从 Git 代码仓库中提取信息，生成包含用于机器学习的特征和标签的大规模“生产 - 测试变更对”JSON 数据集，为未来在回归测试领域引入机器学习、用机器学习实现过时用例自动检测提供数据集层面的有力支持。

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，训练能够自动检测过时测试用例的工件需要大量的真实的数据集以支持训练和测试验证。现有的数据集的规模、数量、质量皆不足以满足回归测试巨量的数据需求。因此，“RegFeatMiner：为回归测试搭建大规模机器学习降噪特征集的数据挖掘软件”应运而生。本软件适用于任何基于 Git 的代码仓库，可以生成规模庞大的特征数据集并内置数据集降噪模块，为回归测试领域软件测试工程师、科研工作者、机器学习工程师提供数据集支持。

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、读取相互匹配的“生产 - 测试”变更对信息、为生产 - 测试变更对标注回归标签或非回归标签、标签二次降噪等。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，配合变更提取算法和区间刺穿功能精准地分析出生产代码变更的细粒度 18 维特征，最终输出大规模的“特征 – 标签”JSON 数据集。

2.2 软件主要模块概述

Git 操作模块：负责与 Git 仓库交互，包括克隆仓库、遍历、分析提交历史等。

输入读取模块：读取 Git 代码仓库变更信息，“生产 – 测试”变更对信息 。

词法解析模块：解析 Java 文件的 AST 语法树，为提取变更的细粒度特征做准备。

特征提取模块：从变更中提取细粒度的、可用于机器学习的特征纳入数据集。

标签生成模块：基于变更对是否回归生成标签数据。

标签降噪微调模块：基于五种策略为标签进行微调， 使回归属性更为真实。

数据集导出模块：生成并导出大规模的、包含特征和标签的 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将机器学习引入回归测试领域而缺乏数据集的机器学习工程师。

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，模块功能流程如图 2 所示。

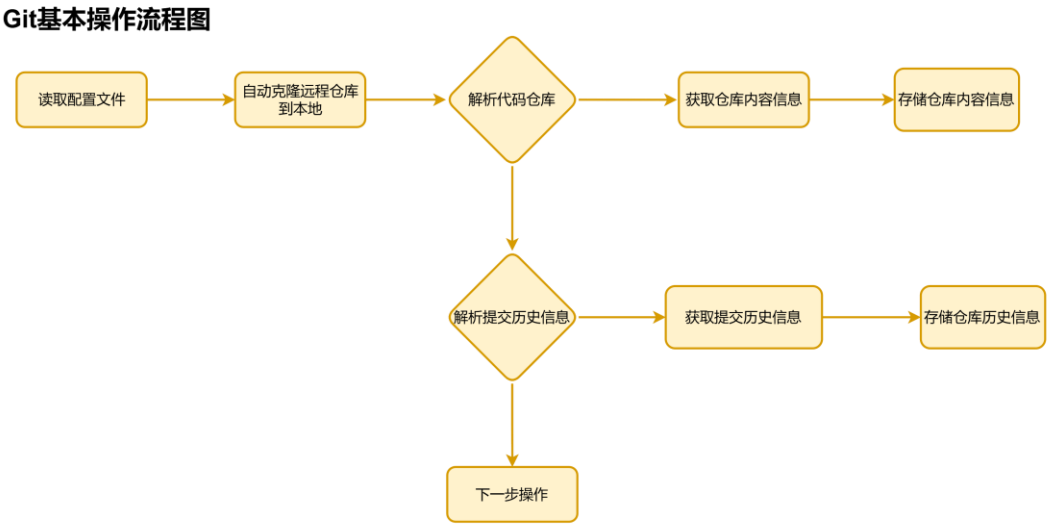

图 2：Git 基本操作模块的功能示意图

```java
package Resource;

public class Resource {  14 个用法  helloechen *
    public static String projectName ="rtree";  //待克隆的仓库名称  1 个用法
    public static String inputDir ="./Method/";  1 个用法
    public static String outputDir ="./Feature";  1 个用法
    public static String gitrepository ="java_data"; //克隆的目标路径  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/";  1 个用法
    public static String githubToken="";//填入个人的github令牌，免密链接github  1 个用法
    public static String remotePath="https://github.com/davidmoten/rtree.git";  //仓库的远程地址
}
```

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，为后续读入生产变更对后进行特征挖掘、数据标注提供了最底层的代码仓库支持。

3.2 读取生产 - 测试变更对信息（由用户提供）

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。因此，我们的软件会读取这些重要的信息用于生产 – 测试对的定位，为后续分析生产代码的分析特征，标注回归标签提供信息支持。以下为一个生产测试对的信息示例，示例如图 4 所示。

```json
{
    "prod_path": "src/main/java/org/cactoos/io/Input.java",
    "test_path": "src/test/java/org/cactoos/io/FileAsInputTest.java",
    "prod_time": "2017-05-24 03:39:45",
    "test_time": "2017-05-24 04:53:22",
    "test_commitID": "a8100f2f12d95aa82dee5fdd7eb6d8fb9042ef81",
    "project": "cactoos",
    "isfound": "found test change",
    "pro_commitID": "073ba56317f314f16d1fc70f0b5048b6b40f5093"
}
```

图 4：软件所需的生产-变更对的基本定位信息示例

- 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

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，为定位后的后续挖掘特征和标签标注工作提供了必要准备。

3.3 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，为后续提取代码的具体更改特征做好充分准备。

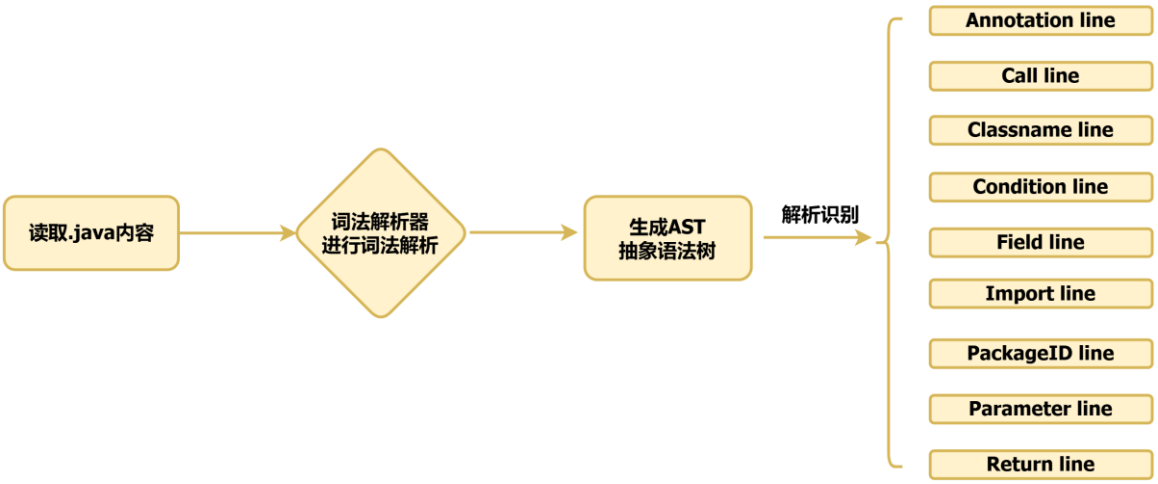

图 5：细粒度解析 java 文件内容的功能流程

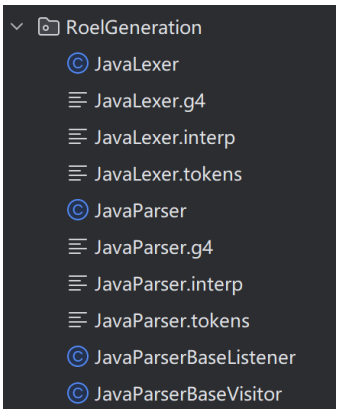

图 6：软件中词法分析器的相关部分

3.4 生产代码细粒度特征提取模块

软件中 Code Diff 模块主要用于对生产环境中代码变更进行详细分析。在进行生产变更内容的详细分析时，Code Diff 模块通过比对新旧版本代码的差异，准确地列出新增、修改和删除的代码行。得到了更改的行号，可以对旧文件和新文件的 AST 语法树分别进行区间穿刺寻找行号交集，得到此更改行更改的是 java 文件中结构的类型，最后通过统计一个 java 文件各个结构发生的代码变更行数，可以得到细粒度的代码变更特征。

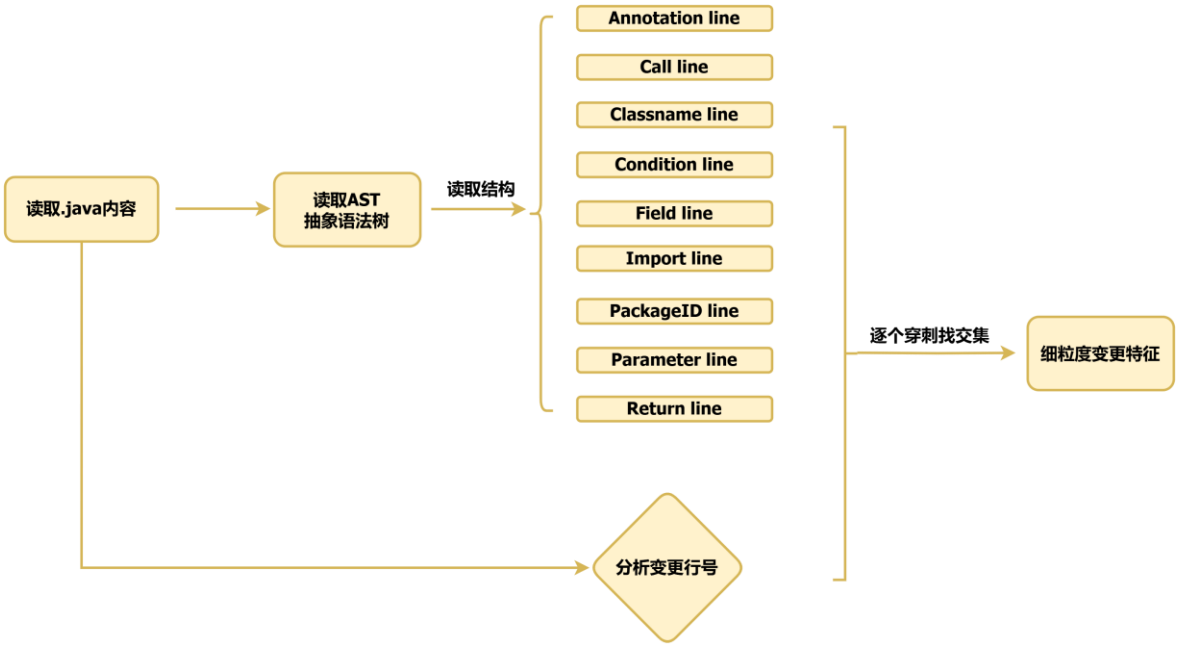

图 7：细粒度提取变更特征的功能流程


```
"add_annotation_line": 0,
"add_call_line": 3,
"add_classname_line": 1,
"add_condition_line": 0,
"add_field_line": 1,
"add_import_line": 0,
"add_packageid_line": 1,
"add_parameter_line": 1,
"add_return_line": 0,
"del_annotation_line": 0,
"del_call_line": 0,
"del_classname_line": 0,
"del_condition_line": 0,
"del_field_line": 0,
"del_import_line": 0,
"del_packageid_line": 0,
"del_parameter_line": 0,
"del_return_line": 0,
```


图 8：挖掘后的生产变更细粒度特征示例

下面是对于此份挖掘特征的详细解释：

- add_annotation_line: 0
  新增的注解行数（annotation line），即代码中增加的注释或元注解的行数。在此数据中为 0，表示没有新增注解行。

- add_call_line: 3
  新增的函数或方法调用行数，表示代码中新增了 3 行调用其他方法或函数的代码。

- add_classname_line: 1
  新增的类名行数，表示代码中新增了一个类的声明。

- add_condition_line: 0
  新增的条件语句行数，例如 if、switch 等语句。这里的值为 0，表示没有新增条件语句。

- add_field_line: 1
  新增的字段（field）行数，例如类中新增的成员变量或全局变量。在此数据中为 1，表示新增了 1 行字段定义。

- add_import_line: 0
  新增的 import 语句行数，表示代码中没有新增导入外部模块或库的语句。

- add_packageid_line: 1
  新增的包名定义行数（package ID line），表示新增了 1 行关于包（package）的定义。

- add_parameter_line: 1
  新增的方法或函数参数行数，表示新增了 1 个参数的定义。

- add_return_line: 0
  新增的返回语句行数，例如 return 语句的行数。这里为 0，表示没有新增返回语句。

- del_annotation_line: 0
  删除的注解行数。此处为 0，表示没有删除任何注解。

- del_call_line: 0
  删除的函数或方法调用行数，表示没有删除任何函数调用代码。

- del_classname_line: 0
  删除的类名行数，表示没有删除任何类的声明。

- del_condition_line: 0
  删除的条件语句行数，表示没有删除任何条件语句。

- del_field_line: 0
  删除的字段行数，表示没有删除任何类的成员变量或全局变量。

- del_import_line: 0
  删除的 import 语句行数，表示没有删除任何导入外部模块的语句。

- del_packageid_line: 0
  删除的包名定义行数，表示没有删除任何包的定义。

- del_parameter_line: 0
  删除的函数参数行数，表示没有删除任何函数或方法的参数。

- del_return_line: 0
  删除的返回语句行数，表示没有删除任何返回语句。

3.6 回归标签标注模块

此模块会在 Git 仓库中定位生产变更和测试变更的具体位置，提取并确认变更时间的真实性。此外，根据回归测试领域相关的研究证明：根据变更时间间隔进行标签标注是目前回归数据集最为可信的标注方式。因此，若初始时间间隔小于 48h，则标定初始标签为回归的正样本；否则就将测试标签标记为非回归的负样本。

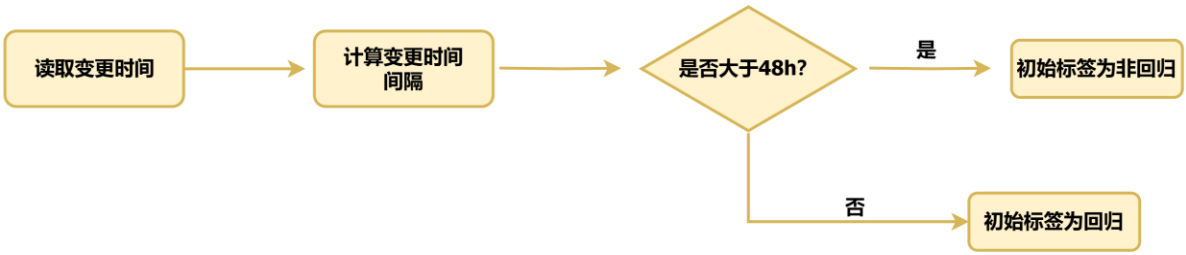

图 9：初始标签标注的逻辑流程

3.7 回归标签的微调降噪模块

根据前人对于回归测试领域的实证研究，仅凭 3.6 的唯一策略会使得数据集存在部分噪声，并使得下游数据驱动任务受到数据质量威胁。因此，软件内部引入了以下五种策略对初始标签进行二次微调和降噪，使其更加符合实证研究中的数据情况。

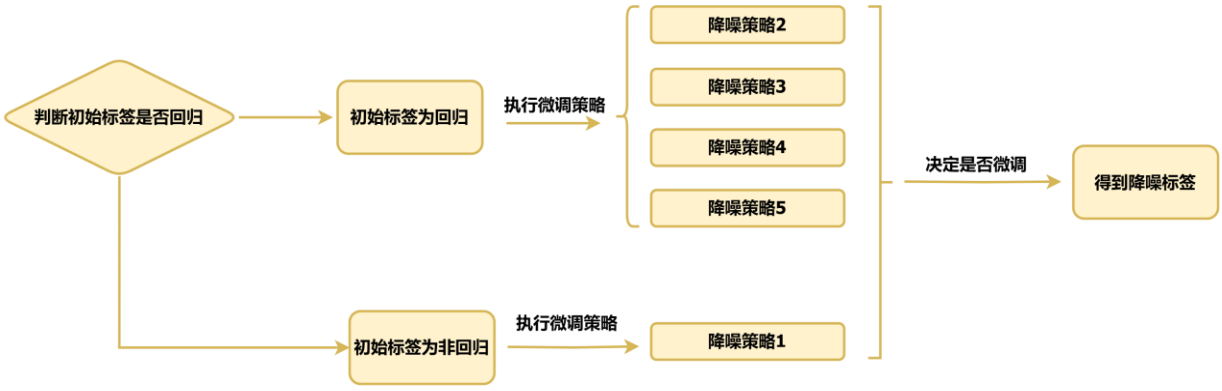

图 10：标签降噪模块的功能逻辑

以下是对五种降噪策略的详细说明：

[1]. 非修改类型变更的生产和测试代码（Non-Modification Type Changes）：

如果生产代码或测试代码的变更类型是新增或删除（非修改类型），且两者的提交之间没有其他生产或测试代码的变更，则视为正样本。这是因为这种类型的变更通常直接相关，

降低了假阳性风险。

[2]. 额外生产代码修改的检测（Additional Production Code Changes）：

如果在生产代码的提交和测试代码的提交之间存在其他生产代码的变更，则可能是独立变更，标记为负样本。这减少了因多个变更交叉引入噪音的可能性。

[3]. 导入语句相关性检测（Import Statements Correlation Check）：

如果生产代码或测试代码的变更仅涉及导入语句，且导入的修改之间没有交集，则标记为负样本。这是因为多个导入语句变更可能是无关的编辑活动。

[4]. 生产和测试代码语义相关性（Semantic Relevance Between Changes）：

测试代码和生产代码通常在语义上应该存在关联。如果变更的语义内容没有交集，则标记为负样本。例如，新增测试方法通常应该覆盖相应的生产方法。

[5]. 注解、修饰符和重构操作（Annotations, Modifiers, and Refactoring Operations）：

如果变更涉及注解、修饰符（例如添加 final）、或仅是重构操作，则标记为负样本。这是因为此类变更通常不会引发真正的生产和测试代码共演。

3.8 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。如图 11 所示，这里展示输出数据集的其中一份数据为例：


```
{
    "repository": "commons-math",  仓库名称
    "prod_path": "src/main/java/org/apache/commons/math/random/GaussianRandomGenerator.java",
    "test_path": "src/test/java/org/apache/commons/math/random/GaussianRandomGeneratorTest.java",
    "prod_time": "2009-09-06 23:32:50",
    "test_time": "2010-01-23 02:18:16",  生产-变更对基本信息
    "type": "EDIT",
    "add_annotation_line": 0,
    "add_call_line": 0,
    "add_classname_line": 0,
    "add_condition_line": 0,
    "add_field_line": 1,
    "add_import_line": 0,
    "add_packageid_line": 0,
    "add_parameter_line": 1,
    "add_return_line": 0,
                                生产变更的细粒度特征
    "del_annotation_line": 0,
    "del_call_line": 0,
    "del_classname_line": 0,
    "del_condition_line": 0,
    "del_field_line": 1,
    "del_import_line": 0,
    "del_packageid_line": 0,
    "del_parameter_line": 1,
    "del_return_line": 0,
    "label": "NEGATIVE",        自动标注并经过降噪的标签
    "prod_commitID": "f8254ebbd3ce543854e93539eb0a3375e8e406b6",
    "test_commitID": "5a91cbf9322b4fc5ed36f5a68b06d33fc22cb3ae",
    "isfound": "found test change",
    "allZero": false
}
```


图 11：最终软件输出的 JSON 示例

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“RegFeatMiner：为回归测试搭建大规模机器学习降噪特征集的数据挖掘软件”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 RegFeatMiner 的内部架构，更熟悉 RegFeatMiner 的使用方法和使用逻辑，下面介绍 RegFeatMiner 的具体目录架构以及对应的作用。

4.1、一级目录：

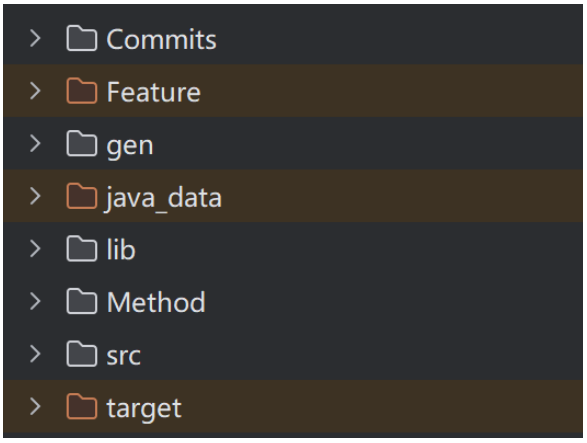

图 12：RegFeatMiner 的一级目录架构

- **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
- **gen**
  存放词法分析器.g4 文件的目录
- **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
- **Method**
  存放生产-测试变更对基本信息的目录，是软件会来读取的目录
- **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）：

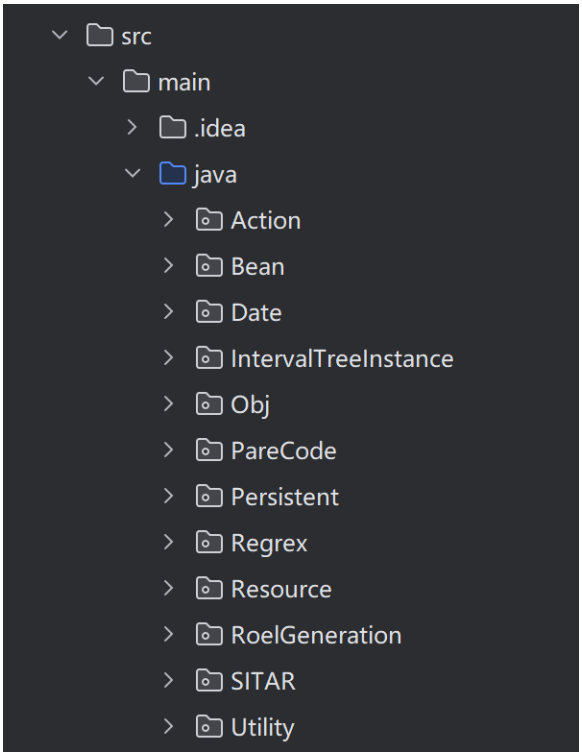

图 13：src 目录下的目录架构

- **Action**
  包含用于生成 AST 抽象语法树的 ASTGenerator.java
  用于分析生产变更种类的 CodeDiff.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 RegFeatMiner 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 RegFeatMiner 的主入口，主逻辑。串联起整个逻辑的同时也是负责标签标记和标

签二次降噪的重要模块。

- **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

RegFeatMiner 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、高质量的细粒度数据集。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。RegFeatMiner 通过自动化的方式，从实际软件项目中挖掘变更特征，快速生成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。RegFeatMiner 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数据集，RegFeatMiner 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。