配置文档

前言

- 本项目为论文Mining Fix Patterns for FindBugs Violations的部分实现源代码，我们使用的是其中自动收集警告数据集的部分。警告收集以及警告跟踪的方法原理在论文2.1、2.2、2.3节中，可自行查看。若收集到的数据不符合需求，可自行对源代码进行diy，以获取所需的警告数据。本文档仅提供一种项目配置方案，如有不同需求，可自行配置。

项目运行环境

- Linux (Ubuntu 18.04)
- Jdk 1.8
- Maven 3.6.3
    - 需要更换阿里云镜像
- Neo4J Community 3.5.X
    - 注意Jdk和Neo4J版本要相互对应，如：最新版本的Neo4J需要Jdk11，Neo4J 3.5.X需要Jdk 1.8

配置流程

- 项目结构如图所示，其中violation-collection为本项目进行配置的警告数据收集部分。

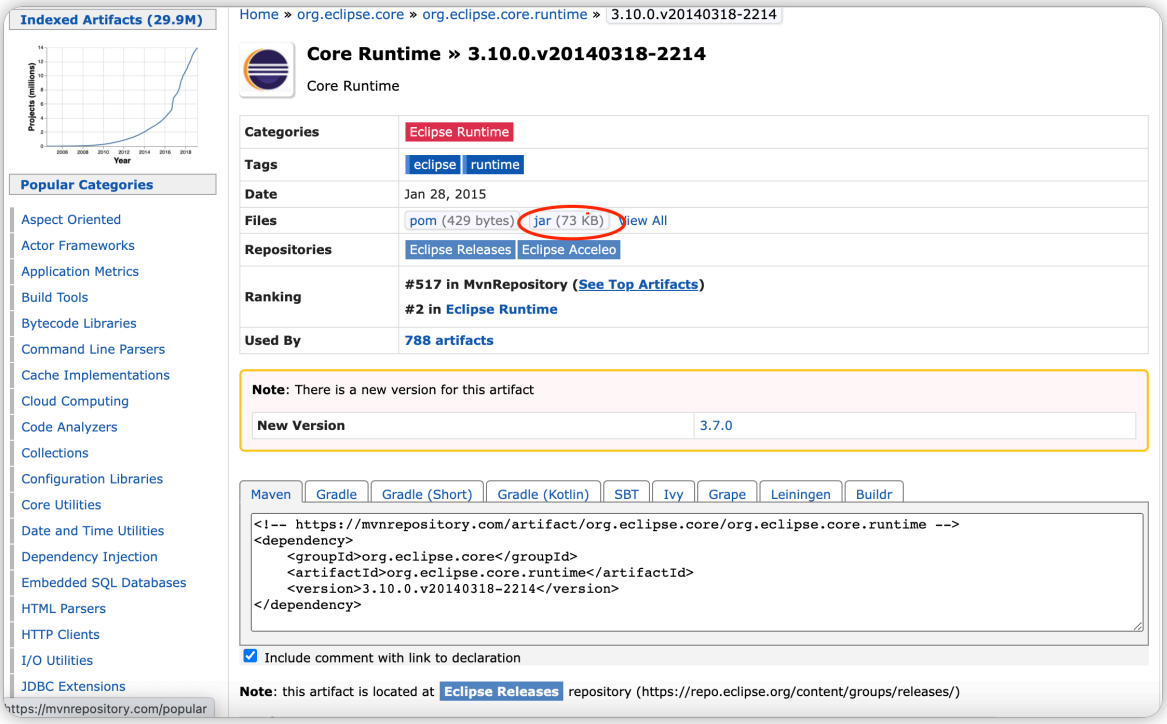
```
(base) ~/findbugs-violations  master ±  tree -L 1
.
├── GitTraverse
├── Mined Fix Patterns
├── Parser
├── PatternMining
├── README.md
├── data
├── gumtree
├── parsing-utils
├── repo-iterator
├── simple-utils
└── violation-collection
```

- 由于violation-collection依赖于simple-utils、repo-iterator、parsing-utils，需要先对这三部分进行构建，最后对violation-collection进行构建。
    - 进入simple-utils目录
    - 执行"mvn clean install -DskipTests"
    - 进入repo-iterator目录
    - 执行"mvn clean install -DskipTests"

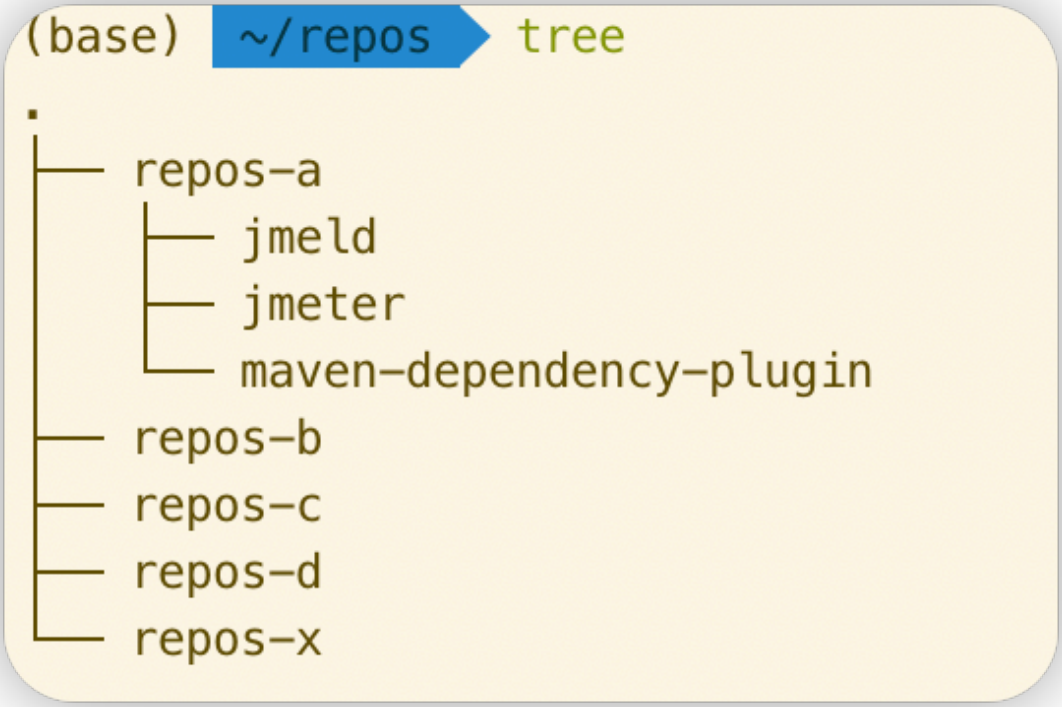
    - 进入parsing-utils目录
    - 执行"mvn clean install -DskipTests"
    - 进入violation-collection目录
    - 执行"mvn clean install -DskipTests"
    - 在构建过程中，若出现依赖无法解析，可根据groupId到https://mvnrepository.com/进行搜索，例如：
        - 无法解析该依赖

```
<dependency>
    <groupId>org.eclipse.core</groupId>
    <artifactId>org.eclipse.core.runtime</artifactId>
    <version>3.10.0.v20140318-2214</version>
</dependency>
```

        - 下载jar包



        - 执行命令"mvn install:install-file -Dfile=xxxxx.jar -DgroupId=xxxxx -DartifactId=xxxxx -Dversion=xxxxx -Dpackaging=jar",其中:
            - Dfile: 要安装的jar的本地路径
            - DgroupId：要安装的jar的groupId （org.eclipse.core）
            - DartifactId: 要安装的jar的 artificialId （org.eclipse.core.runtime）
            - Dversion: jar版本 (3.10.0-v20140318-2214)
            - Dpackaging: 打包类型 (jar)
- 启动neo4j数据库
- 创建repoDir和workingDir两个个文件夹，其中repoDir存放apache项目，workingDir存放生成的报告等数据，文件结构如下图所示：
    - repoDir

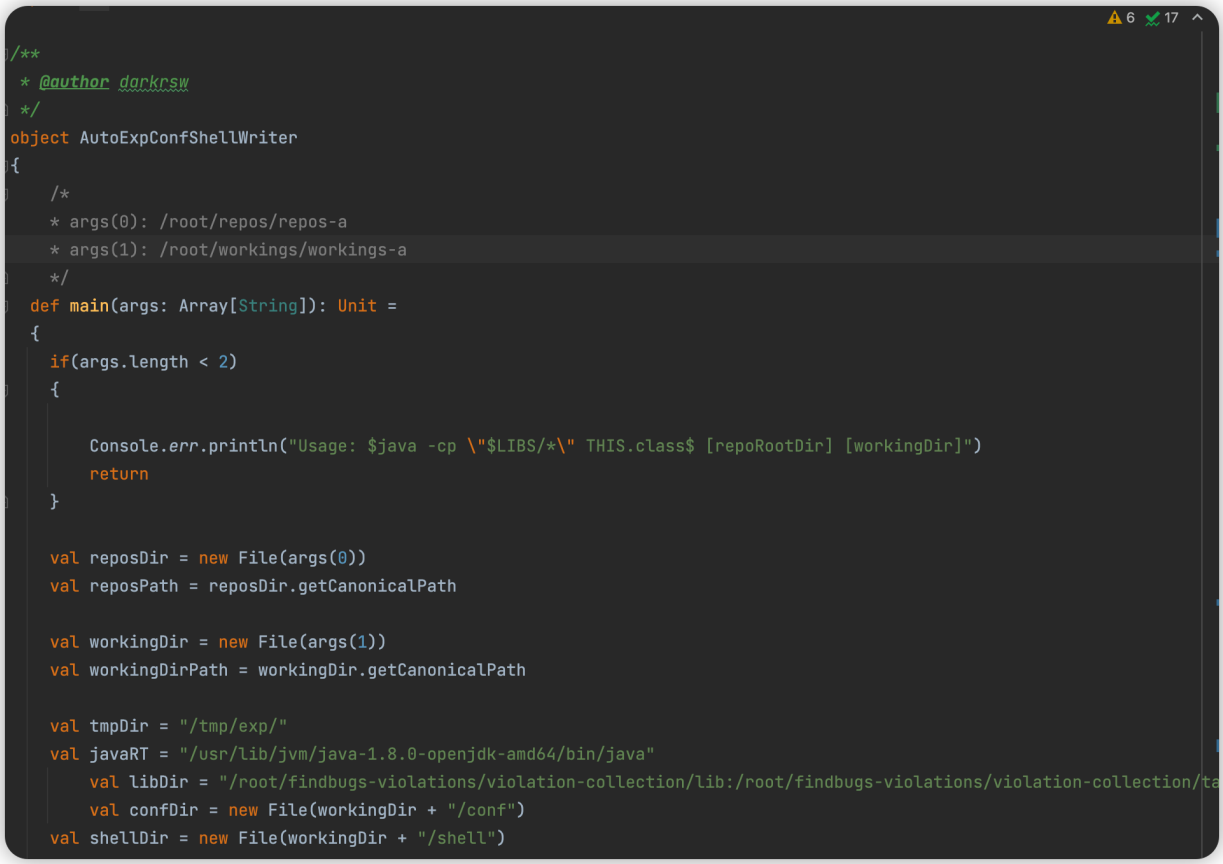
    - 

        - repos目录下，按字母顺序有若干repos-x，存放apache项目。
    - workingDir

```
(base) ~/workings tree
.
├── workings-a
│   ├── conf
│   │   ├── jmeld
│   │   ├── jmeter
│   │   └── maven-dependency-plugin
│   ├── log
│   │   ├── jmeld
│   │   ├── jmeter
│   │   └── maven-dependency-plugin
│   ├── reports
│   │   ├── jmeld
│   │   ├── jmeter
│   │   └── maven-dependency-plugin
│   └── shell
│       ├── jmeld
│       ├── jmeter
│       └── maven-dependency-plugin
├── workings-b
├── workings-c
├── workings-d
└── workings-x
```

        - workings-x和repos-x一一对应，每个working-x中都包含了conf、log、reports、shell四个文件夹，分别存放各个项目的配置信息、日志、警告报告、shell脚本，conf、log、reports、shell都可以通过运行scala脚本自动生成，在下文中会提及。
- 配置findbugs-violations/violation-collection/src/main/scala/edu/lu/uni/serval/alarm/util/AutoExpConfShellWriter.scala
    - 该scala脚本用于根据repoDir中已有的项目，在workingDir中生成对应的conf、log、reports、shell。

- 如图所示：



```
/**
 * @author darkrsw
 */
object AutoExpConfShellWriter
{
    /*
    * args(0): /root/repos/repos-a
    * args(1): /root/workings/workings-a
    */
  def main(args: Array[String]): Unit =
  {
    if(args.length < 2)
    {

        Console.err.println("Usage: $java -cp \"$LIBS/*\" THIS.class$ [repoRootDir] [workingDir]")
        return
    }

    val reposDir = new File(args(0))
    val reposPath = reposDir.getCanonicalPath

    val workingDir = new File(args(1))
    val workingDirPath = workingDir.getCanonicalPath

    val tmpDir = "/tmp/exp/"
    val javaRT = "/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java"
        val libDir = "/root/findbugs-violations/violation-collection/lib:/root/findbugs-violations/violation-collection/ta
        val confDir = new File(workingDir + "/conf")
    val shellDir = new File(workingDir + "/shell")
```

- 输入参数：repoRootDir，workingDirtmpDir
- tmpDir为处理过程中数据临时存放文件夹，可任意指定
- javaRT为jdk所在路径
- libDir存放该项目运行所需的所有库
    - 可执行"mvn -DoutputDirectory=./lib -DgroupId=edu.lu.uni.serval -DartifactId=violation-collection -Dversion=0.1-SNAPSHOT dependency:copy-dependencies"将所需第三方依赖打包进lib文件夹中，其中DoutputDirectory为输出的目标文件夹，DgroupId、DartifactId、Dversion为violation-collection对应的pom.xml的项目信息，如图所示：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
    <modelVersion>4.0.0</modelVersion>
    <groupId>edu.lu.uni.serval</groupId>
    <artifactId>violation-collection</artifactId>
    <packaging>jar</packaging>
    <version>0.1-SNAPSHOT</version>
    <name>violation-collection</name>
```

    - 还需将编译violation-collection项目生成的violation-collection-0.1-SNAPSHOT.jar放入libDir中。
- 在生成shell脚本后，可通过执行shell脚本，调用AlarmExpExecutor对仓库中的apache项目进行自动编译，并使用spotbugs进行扫描，扫描结果存放在reports文件夹中,也可自行修改接口对AlarmExpExecutor进行调用。

- 配置violation-collection/src/main/scala/edu/lu/uni/serval/alarm/util/db/graph/neo4j/VioDBFacade.scala
  - 如图所示：

  - 配置viodbURI，user，passwd
- 配置violation-collection/src/main/scala/edu/lu/uni/serval/alarm/tracking/TrackingExecutorByProject.scala
  - 该scala脚本对同一项目在的不同commit间进行警告跟踪，构建警告生命周期，若想了解警告跟踪，可详细阅读该部分源码
  - 如图所示：

```scala
package edu.lu.uni.serval.alarm.tracking

import ...

object TrackingExecutorByProject extends LazyLogging
{
    /**
     * example
     * args(0):maven-dependency-plugin
     * args(1):/root/workings/workings-a/reports
     * args(2):/root/repos/repos-a/
     */
    def main(args: Array[String]): Unit =
    {
        try{
            internal(args(0), args(1), args(2))
        } catch
        {
            case e: Throwable => logger.error( message = "Unknown Error", e)
            Runtime.getRuntime.exit(127)
        }
        finally
        {
            Runtime.getRuntime.exit(0)
        }
    }

    def internal(projectName: String, alarmRootPath: String, repoRootPath: String): Unit =
    {
```

  - 输入参数：projectName, alarmRootPath, repoRootPath，详见截图中注释的example
- violation-collection/src/main/scala/edu/lu/uni/serval/alarm/util/TrackingHelperUtils.scala
  - 若按照上文描述创建项目仓库，则可使用该scala脚本创建一个.map文件，该文件保存了项目名，以及该项目对应仓库信息。
  - 脚本代码如图所示：

```scala
package edu.lu.uni.serval.alarm.util

import ...

object TrackingHelperUtils
{
    def main(args: Array[String]): Unit =
    {
        mapProject2Pack( mapFilePath = "prj-pack.map",  pathTemplate = "/root/exp/violation/repos/repos-%s")
    }

    def mapProject2Pack(mapFilePath: String, pathTemplate: String) : Unit  =
    {
        val mapFile = new File(mapFilePath)

        val index = 'a'

        0 to 20 foreach { x =>
            val indexChar = (index + x).toChar
            println(indexChar)

            val reportRootDir = new File( pathTemplate.format(indexChar) )

            val prjList = reportRootDir.listFiles().filter(_.isDirectory())
            prjList.foreach( x => FileUtils.write(mapFile, FilenameUtils.getBaseName(x.getCanonicalPath)
                ":repos-"+indexChar + "\n",  encoding = "UTF-8",  append = true) )
        }
    }
```

- .map脚本数据信息如图所示：

```
maven-dependency-plugin:repos-a
jmeter:repos-a
jmeld:repos-a
```

- violation-collection/src/main/scala/edu/lu/uni/serval/alarm/tracking/FixedAlarmCollector.scala
  - 该scala脚本从neo4j数据库中导出已修复警告信息

- 脚本代码如图所示：

```
import ...

object FixedAlarmCollector extends LazyLogging
{
    def main(args: Array[String]): Unit =
    {
        try{
            collectFixedAlarms(args(0), args(1), args(2), args(3), args(4))
        } catch
        {
            case e: Throwable => logger.error( message = "Unknown Error", e)
                Runtime.getRuntime.exit(127)
        }
        finally
        {
            Runtime.getRuntime.exit(0)
        }
//      collectFixedAlarms("fixed-alarms.list",
//              "/root/repos/repos-%s/%s/.git",
//              "/root/findbugs-violations/violation-collection/prj-pack.map",
//              "vtype-stat.csv",
//              "summary-project-vtype.csv"
//              )

    }
```

  - 输入参数：outputPath，repoPathTemplate, mapFilePath, vtypePath, summaryPath，其中：
    - outputPath:导出正报警告数据文件名
    - repoPathTemplate:仓库路径模版，可参考截图的注释样例
    - mapFilePath:上文提及由脚本生成的.map文件路径
    - vtypePath:导出正报警告类型统计信息
    - summaryPath:导出各项目正报警告类型统计信息
  - 若只需要导出单个项目正报警告数据信息，请自行修改接口
- violation-collection/src/main/scala/edu/lu/uni/serval/alarm/tracking/UnfixedAlarmCollector.scala
  - 该scala脚本从neo4j数据库中导出为修复警告信息

- 脚本代码如图：

```
package edu.lu.uni.serval.alarm.tracking

import ...



object UnfixedAlarmCollector
{
  def main(args: Array[String]): Unit =
  {
    doTask( fixSummaryPath = "fixed-summary-project-vtype.csv", args(0))
  }

  def doTask(fixSummaryPath: String, project: String): Unit =
  {
    val summaryFile = new File(fixSummaryPath)
    val summaryList = CSVReader.open(summaryFile).all()


    // prepare project to pack map
        val prj2PackMap = TrackingHelperUtils.readProject2PackMap( mapFilePath = "/home/darkrsw/repo/fa
        println("prj2pack map size: " + prj2PackMap.size)
    val repoPathTemplate = "/mnt/archive1/data/violations/repos/repos-%s/%s/.git"

    //val project = "Activiti-Activiti"
    val outFile = new File(s"unfixed-$project.csv")

    val pack = prj2PackMap(project)
    val repoPath = repoPathTemplate.format(pack,project)
    val tracker = new AlarmTracker(project)
    tracker.initGitRepo(repoPath)
```

  - 输入参数：fixSummaryPath, project，其中：
    - fixSummaryPath:FixedAlarmCollector中summaryPath对应文件
    - project:项目名
  - 其他路径请自行配置
- 主要使用的脚本已经介绍完毕，其他如有需要，请自行配置或对接口、源码进行调整