Analyzing static calls in Java byte-code

Dvinskykh David[1]
Barkovska Olesia[2]

[1,2]Kharkiv National University of Radio Electronics, 14 Nauky Ave, Kharkiv UA-61166, Ukraine, david.dvinskykh@nure.ua

Abstract. *The major goal of research is to analyze and propose a new approach for static analysis of method calls in Java code and finding the class configurations using auxiliary libraries for reading and byte code search.*

Keywords: *byte-code, callgraph, challenges, java, jdk, jgrapht, jre, static analysis.*

I. Introduction and Problem statement

The software developer uses up to 90% of the time to read the code. For checking the places of creation and configuration in different classes, the developer manually checks each place using the usual file search, but due to human factor, there is a probability of missing some places [1]. To correct the situation, a tool is needed for automatic analyzing the code for the connection between its various parts. Many tools are free or paid, but with little functionality or many bugs [2].

The purpose of the study is to develop an approach for static analysis of method calls in Java and to find class configuration places.

Existing analogs (Java-callgraph, Javadepend) and the criteria against which they are compared are shown in Table 1.

Java-callgraph is a console application that requires the presence of Java Virtual Machine [3].

Javadepend is part of the proprietary JArchitect software product

Table 1. Table comparing existing software solutions

Criterias	java-callgraph	javadepend
Cost	Free	Paid
Open source	Open	Closed
License type	2-clause BSD license	N/A
Profiling	Exist	Non-exist
Output type	One	Many
CI/CD	Exist	Non-exist
Search with criteria	Non-exist	Exist

II. Problem solution and Results

The proposed analysis approach consists of three steps, shown in Fig. 1.

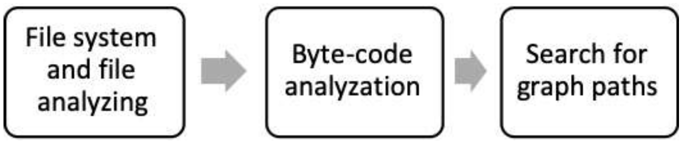

Figure 1. General approach to analyze calls

The first component should be used to collect files that have a class extension, then a list of all the files found is formed and passed to the input of the second component of the program, which will deal with the analysis and construction of vertices of the graph and the dependencies between them.

After constructing the entire graph, the third component begins its work, which looks all the way between the initial group of methods and the final group of methods. If filter criteria are specified, then in each path from start to finish, the program tries to find the first vertex that matches the filter.

Input parameters:

– paths to the files to be analyzed;
– names of packages start and end nodes of the graph;
– package names for the filter criterion;
– output format.

Each vertex that corresponds to the filter will be printed in the console or duplicated in a file. Each point has a name for the class, method, and archive where it was found. Javaagent is one of the JVM settings that allows you to specify an agent to run before launching the application. The agent executable is a standalone application that provides access to the byte code manipulation mechanism (java.lang.instrument) in the runtime.

Byte code can be parsed using the Apache BCEL library, and graphing is a third-party JGraphT library [4].

Criterion for finding paths in a graph - is the full or partial name of packages that contain classes with target methods.

The program should take the following paths:

– the path to the JAR file;
– the path to the WAR file;
– the path to the CLASS file;
– the path to a folder that contains any type of file, namely JAR, or WAR, or CLASS files.

III. Conclusions

Compared to other software solutions support for the filter criterion, additional output path, more output information for support for CI / CD processes were added.

IV. References

[1] Morenets S. Ideal code [Electronic resource] / Sergey Morenets // dou. - 2016. - Resource access mode: https://dou.ua/lenta/articles/perfect-code/.

[2] Martovytskyi, V. O., Kolodochkyn L. L."Stvorennia kros-platformnoi systemy zakhystu Web-servisiv i dodatkiv na osnovi XML-failiv dlia tekhnolohii ASP. NET." Systemy ozbroiennia i viiskova tekhnika 2 (2015): 122-123.

[3] Gousios G. java-callgraph: Java Call Graph Utilities [Online resource] / Georgios Gousios - Resource access mode: https://github.com/gousiosg/java-callgraph/blob/master/README.md.

[4] Explore Existing Architecture - Resource Access Mode:https://www.jarchitect.com/dependenciesview.