A JAVA CLASS ANALYSIS PROGRAM DEVELOPMENT BY ASM LIBRARY

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Chengxiang Qiu

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Program:
Software Engineering

October 2017

Fargo, North Dakota

North Dakota State University
Graduate School

Title

A JAVA CLASS ANALYSIS PROGRAM DEVELOPMENT BY ASM LIBRARY

By

Chengxiang Qiu

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Kenneth Magel
Chair

Jun Kong

Xuehui Li

Approved:

11/14/2017	Kendall Nygard
Date	Department Chair

ABSTRACT

Class analysis is useful technique that can be used in many situations, from the syntaxes parsing, potential bugs finding, and unused code detecting to reverse engineer coding. In this paper, I write a small program classasm analysis the java class by calling the methods in ASM library. Here, the ASM name does not mean anything: it is just a reference to the “_asm_” keyword in C, which allows some functions to be implemented in assembly language. In Java, the ASM provides methods to read write and transform such byte arrays by using higher level concepts than bytes, that mean through the API in the ASM library we can analysis the class without reading the source code. ASM supply method to attract information from the compiled class file. In the research I use the simple company classes and the disconf-core-2.6.35.jar as examples, to show the results.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Kenneth Magel, for his excellent supervision and incredible encouragement. When I was a sit in student in computer science, he gave me great support and help, which encouraged me to register the master degree of software engineering. Without his help and encouragement, I won’t get a chance to switch from my previous biology experience to computer science, a brand-new area that I really interested in.

I also thank my supervisory committee Dr. Jun Kong and Dr. Xuehui Li for their kindly support.

DEDICATION

I dedicate this paper to my Mom, my wife and my two lovely sons. Without their encouragement and supporting, it is hard to imagine that I can start a brand-new career.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGEMENTS iv

DEDICATION v

LIST OF TABLES vii

LIST OF FIGURES viii

LIST OF ABBREVIATIONS ix

1. INTRODUCTION 1

2. BACKGROUND AND RELATED WORK 4

  2.1. Background 4

    2.1.1. Compiled class-file structure 4

    2.1.2. ASM 8

  2.2. Related work 9

    2.2.1. BCEL 9

    2.2.2. SERP 9

    2.2.3. JOIE 11

3. PROPOSED APPROACH 12

4. RESULT AND ANALYSIS 14

5. CONCLUSION 22

REFERENCES 23

APPENDIX A. SOURCE CODE OF JAVA 26

APPENDIX B. SOURCE CODE OF MYSQL 47

LIST OF TABLES

Table Page

1. Overall structure of a compiled class (* means zero or more) 5

2. A class file consists of a single ClassFile structure 6

3. Class access and property modifiers 7

4. Interpretation of FieldType characters 7

LIST OF FIGURES

Figure Page

1. Sequence diagram for the ChangeVersionAdapter. 8

2. Class diagram of company class. 13

3. Class structure of classasm project. 14

4. Class analysis output of Company class. 15

5. Class analysis output of disconf-core-2.6.35.jar. 16

6. Class analysis output of disconf-core-2.6.35.jar in MySQL. 18

7. Query the class analysis output of disconf-core-2.6.35.jar in MySQL. 18

8. Plot of methods' class, x is the method name, and fill is the class that the method belongs to. 19

9. Plot of methods invoke other methods, x row is method of calling, and the fill is the method of been called. 20

10. Plot of invoked methods' class, x is the invoked method, and fill is the class that the method belongs to. 21

LIST OF ABBREVIATIONS

JVM	Java virtual machine
BCEL	Byte Code Engineering Library
JOIE	Java Object Instrumentation Environment
API	Application program interface

1. INTRODUCTION

In software engineering development process, program analysis, generation and transformation are necessary and useful. Through the class analysis, potential bugs and unused code can be detected, furthermore, it is the basis for reverse engineer code, because some reassembling process can only base on the complied classes. Program generation is used in compilers such as traditional compilers, stub compilers, skeleton compilers which used for distributed programing [1]. Program transformation is also very important, which can be used to optimize or obfuscate programs. Bytecode instrumentation is one of the program transformation technology which can easily insert or delete code into applications to realize the debugging or performance improvement, and it can also monitor the code performance [2]. Bytecode instrumentation technique is one of the most often using program transformation technique, which is a valuable technique for transparently enhancing virtual execution environments for purposes such as monitoring or profiling. The advantage of working at the bytecode level is that the source code is not needed. That means we can altering java semantics via bytecode instrumentation without known the java source code and only need the compiled classes [3]. Once the bytecode instrumentation is introduced, every bytecode that is being executed in the JVM will be full covered [4].

Bytecode instrumentation includes two types, static bytecode instrumentation and dynamic instrumentation. The main difference between the two types of instrumentation is the static instrumentation inserts all instrumentation-code before the program starts execution, while dynamic instrumentation interleaves with the execution of the program which under instrumentation. The obvious advantage of the static instrumentation is that it uses less runtime overhead, because all classes are instrumented before the program is executed [3]. Another advantage is static bytecode instrumentation may use any high level bytecode engineering library such as BCEL, ASM, JOIE. And this will not perturb the measurement processes. While in the dynamic bytecode instrumentation process, an instrumentation agent will be invoked whenever a class is loaded and the loaded bytecode need to augment with instrumentation code. Compare with the static instrumentation, in order to make sure all classes will be instrumented and avoids tedious bytecode instrumentation before the program execute, this approach introduces extra overhead and will perturbate measurements due to the runtime instrumentation-process [5-9].

Interposition is a key step in the implementation of dynamic adaptable systems, which can be used not only to modify the semantics of components, but also to dynamically add and remove the additional properties such as functional or non-functional properties [6, 10]. Most of the techniques used to implement dynamically adaptable systems used this kind of interposition. The interposition requires an interposition object and the object masks are in the same type, so that the interposition tools can generate interposition code specific to the type of that objects. For example, to implement a pure Java meta object protocol, we can choose a code generation tool to generate interposition classes, or choose a code manipulation tool to inline the interposition code directly in the classes that we want to analysis [11].

Although the code generation and manipulation tools are useful to implement adaptable systems, there is one problem that produced in the manipulation process must be faced. That is the source code dependence, because most of code generation and manipulation tools finish this process depend on the support of the source code, these tools need generate source code and then dynamically compile this source code [12]. This process would add large overheads to applications both in time and in size, that means would add the time and space complexity. So, it requires the tools are as small and fast as possible to do this job. ASM is such kind of tool, it will optimize the performances of an application through optimize the most frequently executed code. Also, ASM will build a general tool to implement any class manipulation operation. In this paper, I will use the examples to show how ASM analysis the Java classes [9].

2. BACKGROUND AND RELATED WORK

2.1. Background

2.1.1. Compiled class-file structure

In Java, class is program-code-template for creating objects, it contains package and import section, also includes class attributes, annotations, methods and so on. Java program is made up by classes and these classes work together, they can realize some functions. If we analysis the classes based on the source code, we need to walk through the code line by line, check every class. While the compiled Java class is quite different from the class in the source code, it is produced by Java compiler from Java source code file (.java files). Compiled class is a file with .class filename extension, contains Java bytecode that can be executed on the JVM [4]. In the source file it generally contains more than one class, while a compiled class describes only one class which contains all information of the source code. A compiled class retains the structural information and almost all the symbols from the source code. In a compiled class, it contains: a section describing the modifiers, the name, the super class, the interfaces and the annotation of the class. There are many sections that describe the modifiers, the name, the type and annotations for each field. Each method and constructor will be described by an individual section [9, 13].

There are several differences between the source code class and the compiled class. No matter how many classes there are in the source code, it will be compiled into only one class. In this compiled class it includes all information, in the following sections: one section for the main class and the other for the inner class. The main class file contains references to its inner classes, and inner classes defined inside methods contain a reference to their enclosing method. Although it contains class, field, method, compiled class does not contain comments, it uses the code attributes to associate additional information to these elements. All the type names in the compiled class must be fully qualified, because, there is no package and import section in the compiled class. All the numeric, string and type constants will be stored in a constant pool section in the compiled class. Table 1 summarizes the overall structure of compiled class.

Table 1. Overall structure of a compiled class (* means zero or more) [9].

Modifiers, name, super class, interfaces	
Constant pool: numeric, string and type constants	
Source file name (optional)	
Enclosing class reference	
Annotation*	
Attribute*	
Inner class*	Name
	Modifiers, name, return and parameter types
Field*	Annotation*
	Attribute*
Method* Modifiers, name, return and parameter types	
Annotation*	
Attribute*	
Compiled code	

In the compiled class format, each class file contains the definition of a single class or interface. Due to the class is generated by a class loader, a class or interface need not have an external representation literally contained in a file, it will colloquially refer to any valid representation of a class or interface as being in the compiled class. For example, in Java SE platform, it has its own set of data types representing compiled class data. The types u1, u2, and u4 represent an unsigned one-, two-, or four-byte quantity, respectively. This format is supported by interfaces java.io.DataInput and java.io.DataOutput and classes such as java.io.DataInputStream and java.io.DataOutputStream. And JVM can recognize the class-file structure in the table 2, the flags in the table 3 and the FieldType in the table 4.

Table 2. A class file consists of a single ClassFile structure [14].

u4	**magic;**
u2	minor_version;
u2	major_version;
u2	constant_pool_count;
cp_info	constant_pool[constant_pool_count-1];
u2	access_flags;
u2	this_class;
u2	super_class;
u2	interfaces_count;
u2	interfaces[interfaces_count];
u2	fields_count;
field_info	fields[fields_count];
u2	methods_count;
method_info	methods[methods_count];
u2	attributes_count;
attribute_info	attributes[attributes_count];

Table 3. Class access and property modifiers [14]

Flag Name	Value	Interpretation
ACC_PUBLIC	0x0001	Declared public; may be accessed from outside its package.
ACC_FINAL	0x0010	Declared final; no subclasses allowed.
ACC_SUPER	0x0020	Treat superclass methods specially when invoked by the invokespecial instruction.
ACC_INTERFACE	0x0200	Is an interface, not a class.
ACC_ABSTRACT	0x0400	Declared abstract; must not be instantiated.
ACC_SYNTHETIC	0x1000	Declared synthetic; not present in the source code.
ACC_ANNOTATION	0x2000	Declared as an annotation type.
ACC_ENUM	0x4000	Declared as an enum type.

Table 4. Interpretation of FieldType characters [14].

BaseType Character	Type	Interpretation
B	byte	signed byte
C	char	Unicode character code point in the Basic Multilingual Plane, encoded with UTF-16
D	double	double-precision floating-point value
F	float	single-precision floating-point value
I	int	integer
J	long	long integer
L ClassName ;	reference	an instance of class ClassName
S	short	signed short
Z	boolean	true or false
[	reference	one array dimension

As we know, in Java, class file has its special format, it consists of a stream of 8-bit bytes. That means all 16-bit, 32-bit and 64-bit data are constructed by reading in 2, 4 and 5 consecutive 8-bit bytes, respectively. In the table 2, the types u1, u2 and u4 represent an unsigned one-, two, and four- byte quantity, respectively. The values of the minor_version and major_version items represent the minor and major version numbers of this class file. Table 3 shows the ACC_INTERFACE flag rules. By different values attached to the flags, class file can define different property of the class such as public, final, supper, abstract a class. While, compiled class file interpretation the field type use only single character, as in the table 4. By single character such as B, C, D which can represent different data type in the field.

2.1.2. ASM

In Java ASM provides bunches of API to manipulate the compiled class file, there are three core components: the ClasReader, ClassWriter and the ClassVisitor. The ClassReader class parses a compiled class given as a byte array, and calls the corresponding visitXxx methods on the ClassVisitor instance passed as argument to its accept method. It can be seen as an event producer. The ClassVisitor class delegates all the method calls it receives to another ClassVisitor instance. It can be seen as an event filter. The ClassWriter class is a subclass of the ClassVisitor abstract class that builds compiled classes directly in binary form. It produces as output a byte array containing the compiled class, which can be retrieved with the toByteArray method. It be an event consumer. Figure 1. shows the sequence diagram of modified class version.

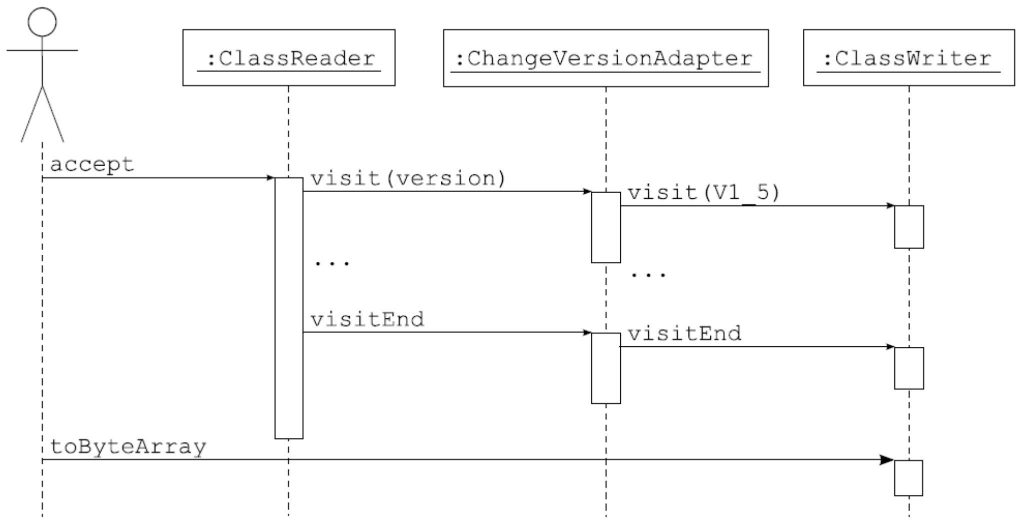


Figure 1. Sequence diagram for the ChangeVersionAdapter [9].

2.2. Related work

2.2.1. BCEL

BCEL (Byte Code Engineering Library) is the Java class manipulation tool which intended to give users a convenient way to analyze, create and manipulate Java class files. A Java class generally includes the attributes, fields and methods which are represented by objects. These objects can be loaded from an existing file at static model, that mean we can read the objects from the file and transformed them if necessary at run-time and then output them into a file. We can also create classes from scratch at run-time. The BCEL is also important to help as to understand the JVM and the format of Java .class files. There are three phases of these process, in the first phase, the class is transformed and represented by byte array, and correspondingly an object model related to the structure of this class is constructed in memory. In such a structure, at different level of bytecode instruction, one object is created for each node. In the second phase, the object model that constructed in previous phase were manipulated. And then in the third phase, this modified object will be graphed into a new byte array [15]. Through such three phases process, BCEL successfully solved the serialization and deserialization problems.

The users do not need to see all the serialization and deserialization details can solve the serialization and deserialization problems, because the BCEL provide the functions to finish this. And it also solves the symbol table management problems. While it has to explicitly give the indexes of the constants in the symbol table, and have to manually update these indexes if they become invalid.

2.2.2. SERP

SERP is an open source framework for manipulating java bytecode by a similar approach as BCEL [16]. It has lots of advantages:

- High-level APIs in the SERP let all normal bytecode are easy to be modified.
- There is a large methods library, which makes it is easy to use it as clean as possible, for example, overloads its methods to guaranty the type consistence and to make shortcuts for operations like adding default constructions.
- SERP does not hide any of the power of bytecode manipulation behind a limited set of high-level functions. Advanced users also can directly access to the low-level details of the class file and constant pool, the users can switch back and forth between low-level and high-level operations. SERP maintains complete consistency of the class structure at all times. In generally, any change to a method descriptor in the constant pool, often change the return values of all the high-level APIs that describe that method.
- In order to minimize the size of the class structures, there is a shared constant pool. SERP let user access to the constant pool directly. SERP's high-level APIs completely abstract management of the constant pool. Any time a new constant is needed, SERP will automatically add it to the pool while ensuring that no duplicates ever exist.
- Most SERP instruction representations have the ability to change their underlying low-level opcodes on the fly as the users modify the parameters of the instruction.

Although SERP has lots of advantages, it is not ideally suited to all applications, here are some disadvantages:

- SERP is not built for speed, it is a quit slow process. It costs a lot of time when the classes are loaded.

- Because SERP tries to keep the full-time consistency between the low and high-level class structures, it slows down both access and mutates methods.
- SERP's high-level structures for representing class bytecode are very memory-hungry.
- The SERP toolkit is not thread-safe, for the multiple threads cannot safely make modifications to the same classes the same time.
- Although there are plans to implement synchronize, as well as plans to allow operations to modify bytecode based on specified patterns, similar to aspect-oriented programming. While at the project-level modifications, changes made in one class in a SERP project are not yet automatically propagated to other classes.

2.2.3. JOIE

JOIE (Java Object Instrumentation Environment) is a framework for safe Java bytecode transformation. It provides both low-level and high-level functionality to extend or adapt compiled Java classes. The low-level interface allows manipulating the bytecodes itself whereas the high-level interface provides methods for inserting new interfaces, fields, methods or whole code splices. Unfortunately, this tool is not maintenance any more, the last update is 2003 [17, 18].

3. PROPOSED APPROACH

The goal of this paper is to analysis the class, more specifically, is to analysis the methods in the class. Which methods have invoked a method in the class, or a method invokes how many methods inside or outside of some classes. In this paper, I use the asm bytecode analysis package to analysis the class. ClassVisitor is an important class of the package. It supply the two constructors ClassVisitor(int api) and ClassVisitor(int api, ClassVisitor cv) , which with different parameters, both of them can construct a new ClassVisitor. It also supply with the methods, AnnotationVisitor and FieldVisitor through which, they can visit the annotation and the field of a class respectively. MethodVisitor class is another important class; a visitor can visit a Java method. The methods of this class must be called in the following order: [visitAnnotationDefault] (visitAnnotation | visitParameterAnnotation | visitAttribute)* [visitCode (visitFrame | visitXInsn | visitLabel | visitTryCatchBlock | visitLocalVariable | visitLineNumber)* visitMaxs] visitEnd. In addition, the visitXInsn and visitLabel methods must be called in the sequential order of the bytecode instructions of the visited code, visitTryCatchBlock must be called before the labels passed as arguments have been visited, and the visitLocalVariable and visitLineNumber methods must be called after the labels passed as arguments have been visited [19]. A label represents a position in the bytecode of a method. Labels are used for jump, goto, and switch instructions, and for try catch blocks. A label designates the instruction that is just after. The label class combines with the type class make it easier to manipulate type and method descriptors.

In this paper, I will use three simple test classes: company class, employee class and people class which includes a few simple test methods to test the code, and the company class diagram showed in Figure 2.

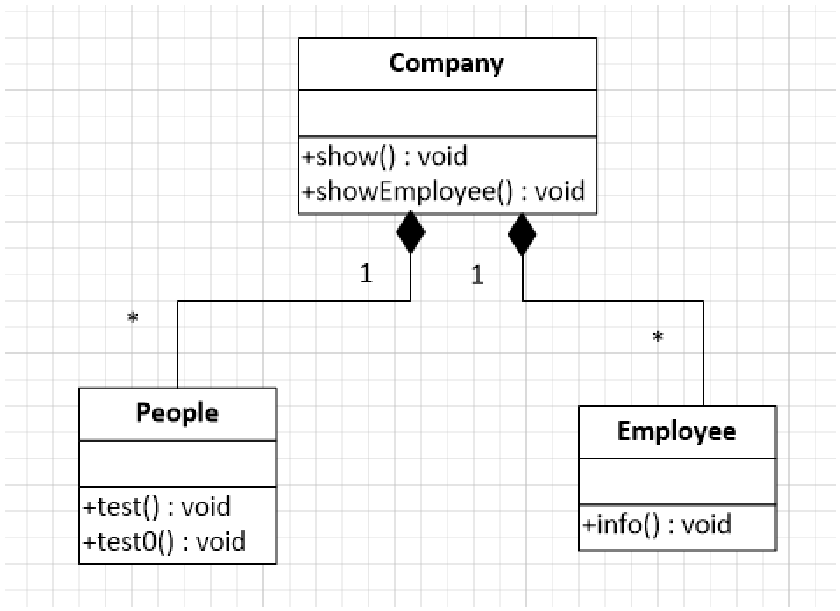
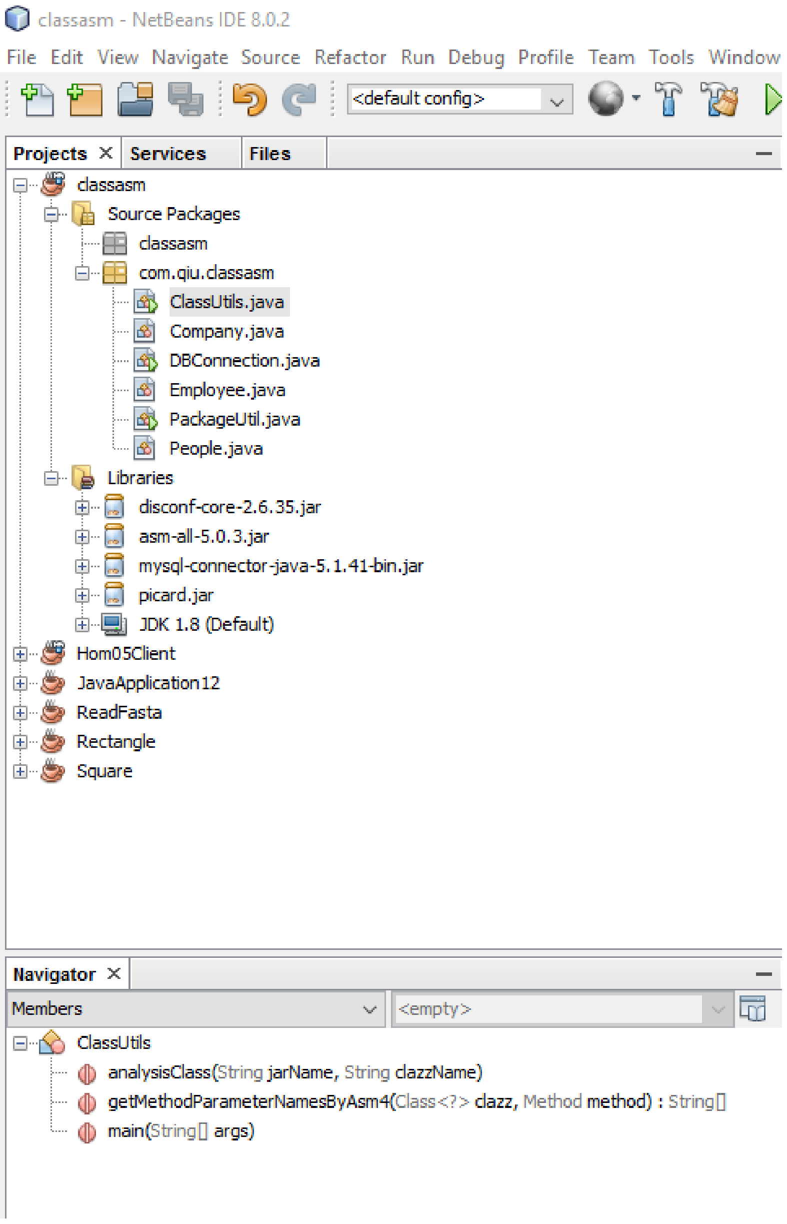


Figure 2. Class diagram of company class.

Show () method in the company class will call the method test () and test0() in the class People, and also call the method info () in the class Employee. I the result part I will show these methods invocations analyzed by asm bytecode package. In addition to this simple class test, I also tested a complex jar file disconf-core-2.6.35.jar which is a Distributed Configuration Management Platform package [20]. This package is developed by Baidu Company. I will use my program to analysis the methods invocation in this package. And I also use the MySQL to establish the database named result, in the database, all the methods analysis result will output into the database.

4. RESULT AND ANALYSIS

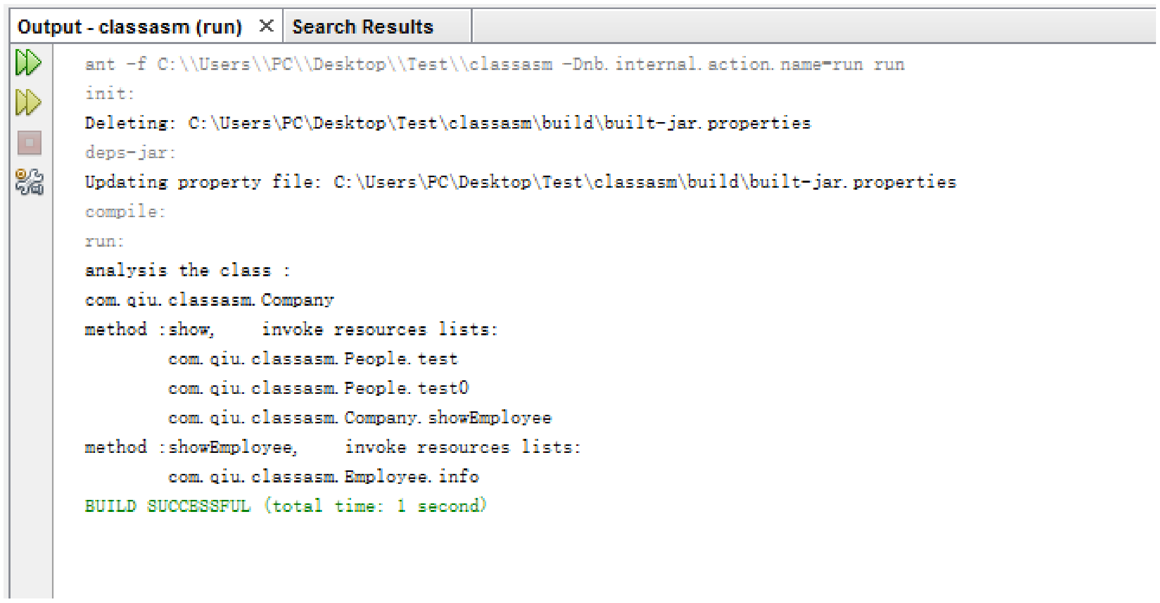

classasm - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window
<default config>
Projects
Services
Files
Navigator
Members
<empty>

```
classasm
    Source Packages
        classasm
        com.qiu.classasm
            ClassUtils.java
            Company.java
            DBConnection.java
            Employee.java
            PackageUtil.java
            People.java
    Libraries
        disconf-core-2.6.35.jar
        asm-all-5.0.3.jar
        mysql-connector-java-5.1.41-bin.jar
        picard.jar
        JDK 1.8 (Default)
Hom05Client
JavaApplication12
ReadFasta
Rectangle
Square
```

```
ClassUtils
    analysisClass(String jarName, String clazzName)
    getMethodParameterNamesByAsm4(Class<?> clazz, Method method) : String[]
    main(String[] args)
```

Figure 3. Class structure of classasm project.

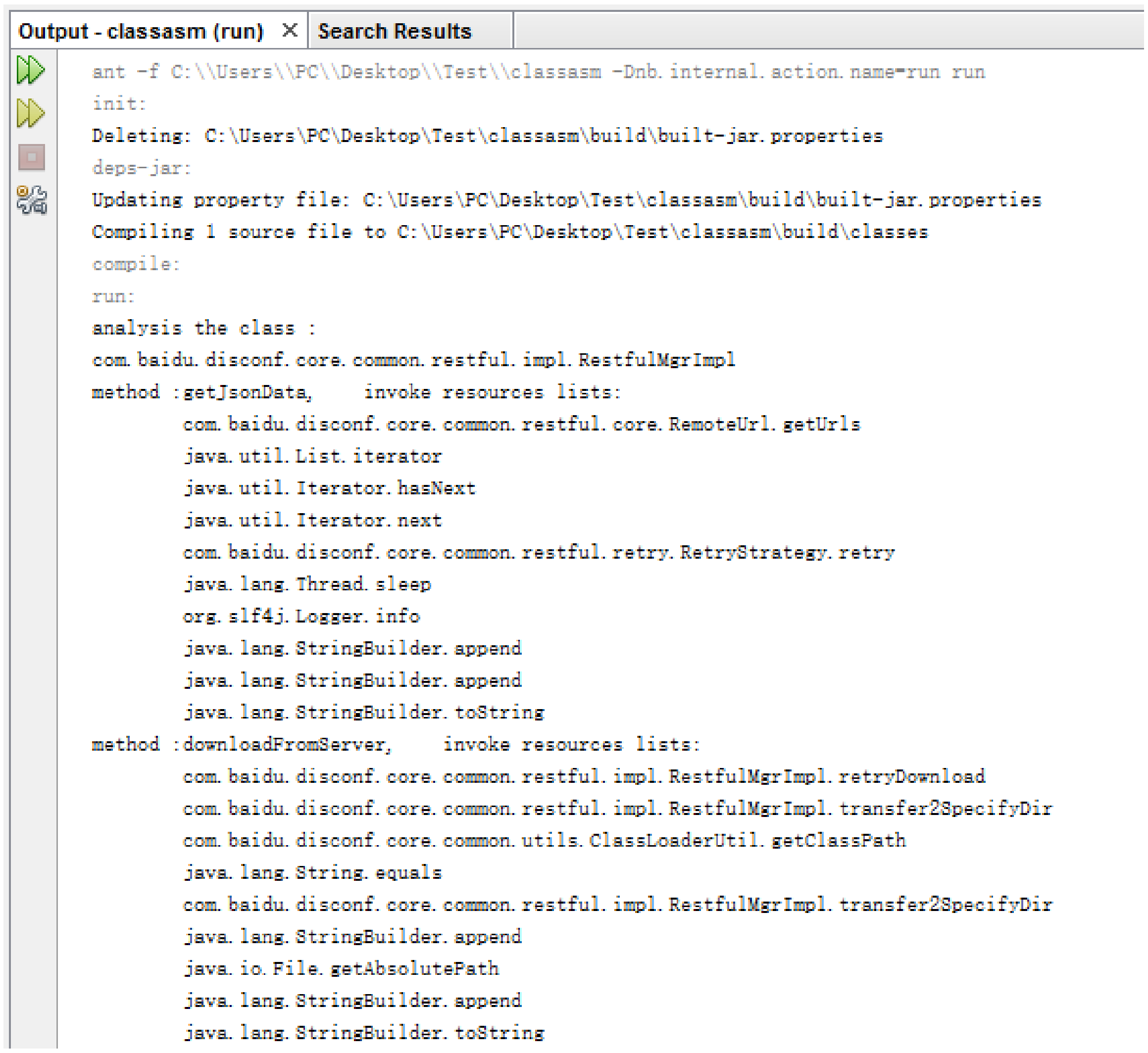
In my project, there are 6 classes as Figure 3, the company.java, employee.java and people.java are three simple classes used as example to test the code. And ClassUtils.java, PackageUtil.java and DBConnection.java are function classes that realize the class analysis. DBConnection.java is to connect to MySQL database.



```
ant -f C:\\Users\\PC\\Desktop\\Test\\classasm -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\PC\Desktop\Test\classasm\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\PC\Desktop\Test\classasm\build\built-jar.properties
compile:
run:
analysis the class :
com.qiu.classasm.Company
method :show,    invoke resources lists:
        com.qiu.classasm.People.test
        com.qiu.classasm.People.test0
        com.qiu.classasm.Company.showEmployee
method :showEmployee,    invoke resources lists:
        com.qiu.classasm.Employee.info
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 4. Class analysis output of Company class.

Figure 4 is the output of a demo class analysis by classasm. In the company class, there are two methods, show () and showEmployee(), the show() method invokes two methods in People class, which are test() and test0(), and the showEmployee() method invokes one method info() in Employee class. The output result is consistence with my design and goal.



```
ant -f C:\\Users\\PC\\Desktop\\Test\\classasm -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\PC\Desktop\Test\classasm\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\PC\Desktop\Test\classasm\build\built-jar.properties
Compiling 1 source file to C:\Users\PC\Desktop\Test\classasm\build\classes
compile:
run:
analysis the class :
com.baidu.disconf.core.common.restful.impl.RestfulMgrImpl
method :getJsonData,     invoke resources lists:
        com.baidu.disconf.core.common.restful.core.RemoteUrl.getUrls
        java.util.List.iterator
        java.util.Iterator.hasNext
        java.util.Iterator.next
        com.baidu.disconf.core.common.restful.retry.RetryStrategy.retry
        java.lang.Thread.sleep
        org.slf4j.Logger.info
        java.lang.StringBuilder.append
        java.lang.StringBuilder.append
        java.lang.StringBuilder.toString
method :downloadFromServer,     invoke resources lists:
        com.baidu.disconf.core.common.restful.impl.RestfulMgrImpl.retryDownload
        com.baidu.disconf.core.common.restful.impl.RestfulMgrImpl.transfer2SpecifyDir
        com.baidu.disconf.core.common.utils.ClassLoaderUtil.getClassPath
        java.lang.String.equals
        com.baidu.disconf.core.common.restful.impl.RestfulMgrImpl.transfer2SpecifyDir
        java.lang.StringBuilder.append
        java.io.File.getAbsolutePath
        java.lang.StringBuilder.append
        java.lang.StringBuilder.toString
```

Figure 5. Class analysis output of disconf-core-2.6.35.jar.

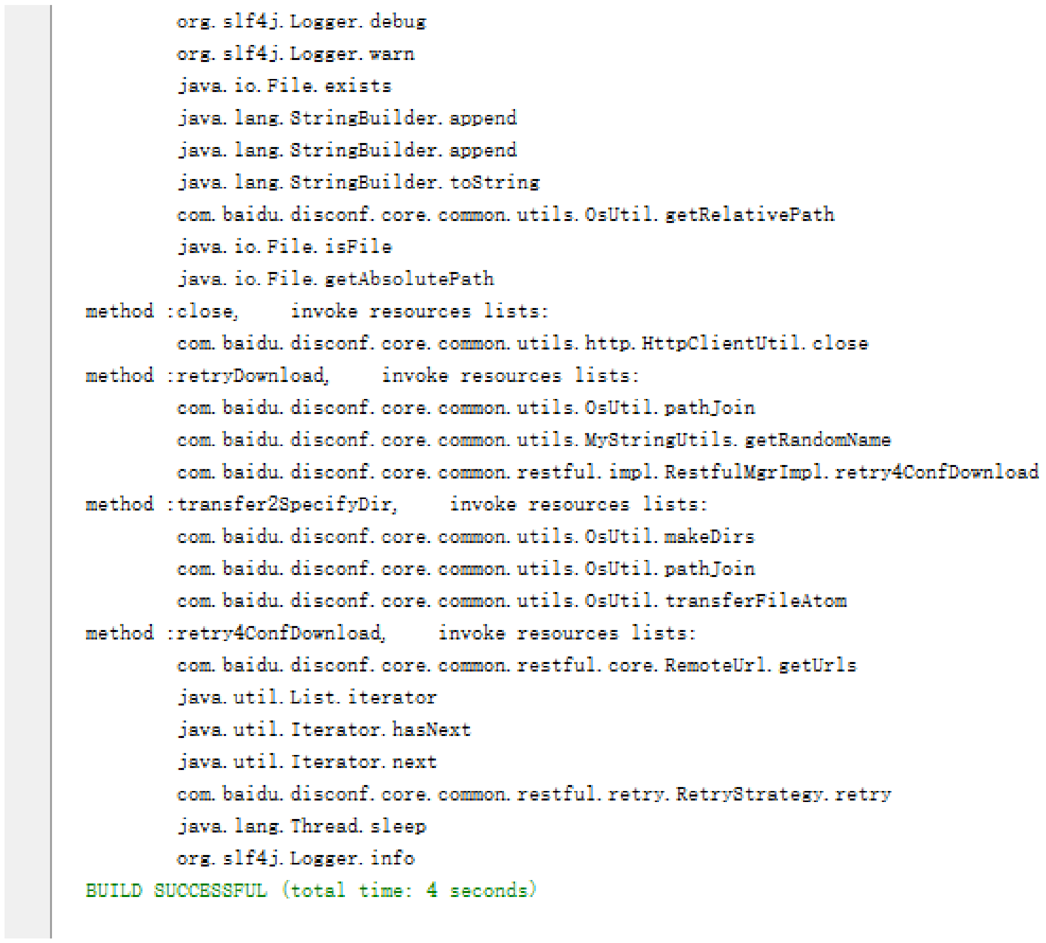
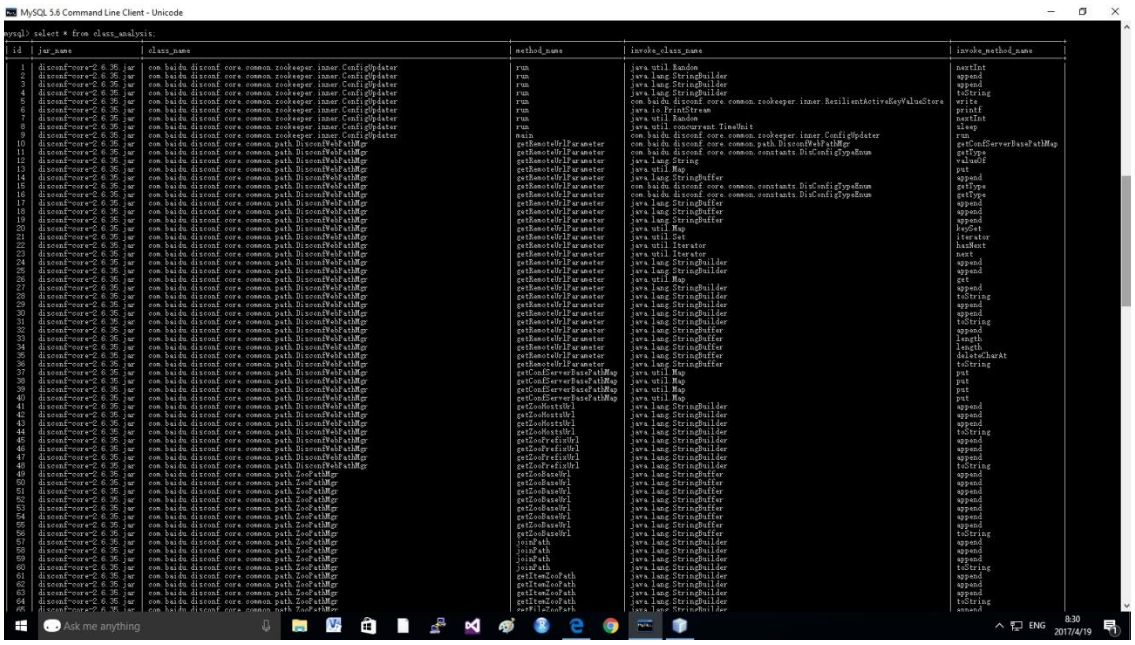

```
        org.slf4j.Logger.debug
        org.slf4j.Logger.warn
        java.io.File.exists
        java.lang.StringBuilder.append
        java.lang.StringBuilder.append
        java.lang.StringBuilder.toString
        com.baidu.disconf.core.common.utils.OsUtil.getRelativePath
        java.io.File.isFile
        java.io.File.getAbsolutePath
method :close,    invoke resources lists:
        com.baidu.disconf.core.common.utils.http.HttpClientUtil.close
method :retryDownload,    invoke resources lists:
        com.baidu.disconf.core.common.utils.OsUtil.pathJoin
        com.baidu.disconf.core.common.utils.MyStringUtils.getRandomName
        com.baidu.disconf.core.common.restful.impl.RestfulMgrImpl.retry4ConfDownload
method :transfer2SpecifyDir,    invoke resources lists:
        com.baidu.disconf.core.common.utils.OsUtil.makeDirs
        com.baidu.disconf.core.common.utils.OsUtil.pathJoin
        com.baidu.disconf.core.common.utils.OsUtil.transferFileAtom
method :retry4ConfDownload,    invoke resources lists:
        com.baidu.disconf.core.common.restful.core.RemoteUrl.getUrls
        java.util.List.iterator
        java.util.Iterator.hasNext
        java.util.Iterator.next
        com.baidu.disconf.core.common.restful.retry.RetryStrategy.retry
        java.lang.Thread.sleep
        org.slf4j.Logger.info
BUILD SUCCESSFUL (total time: 4 seconds)
```

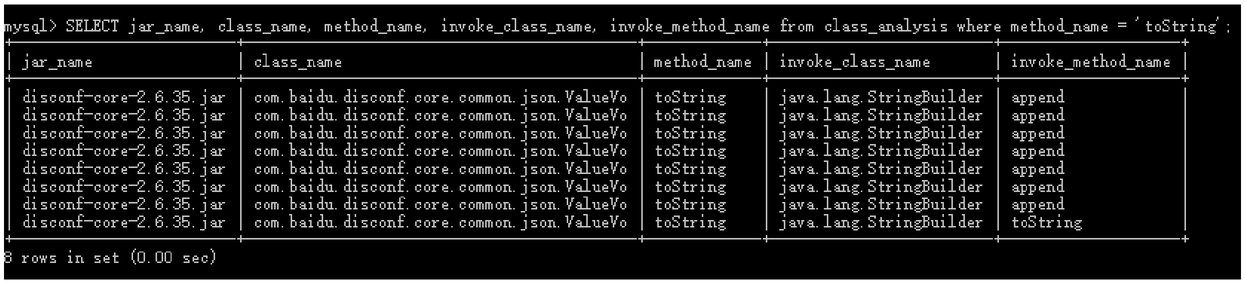
Figure 5. Class analysis output of disconf-core-2.6.35.jar (continued).

Figure 5 is screenshot of the program output in the NetBeans, it shows the invoking relationship between the methods and classes of disconf-core-2.6.35.jar.



```
ysql> select * from class_analysis;
+----+-------------------------+-------------------------------------------------------------+--------------------------+----------------------------------------------------------------------------+--------------------------+
| id | jar_name                | class_name                                                  | method_name              | invoke_class_name                                                          | invoke_method_name       |
+----+-------------------------+-------------------------------------------------------------+--------------------------+----------------------------------------------------------------------------+--------------------------+
|  1 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.util.Random                                                           | nextInt                  |
|  2 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.lang.StringBuilder                                                    | append                   |
|  3 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.lang.StringBuilder                                                    | append                   |
|  4 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.lang.StringBuilder                                                    | toString                 |
|  5 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | com.baidu.disconf.core.common.zookeeper.inner.ResilientActiveKeyValueStore | write                    |
|  6 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.io.PrintStream                                                        | printf                   |
|  7 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.util.Random                                                           | nextInt                  |
|  8 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | run                      | java.util.concurrent.TimeUnit                                              | sleep                    |
|  9 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater | main                     | com.baidu.disconf.core.common.zookeeper.inner.ConfigUpdater                | run                      |
| 10 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | com.baidu.disconf.core.common.path.DisconfWebPathMgr                       | getConfServerBasePathMap |
| 11 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | com.baidu.disconf.core.common.constants.DisConfigTypeEnum                  | getType                  |
| 12 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.String                                                           | valueOf                  |
| 13 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.util.Map                                                              | put                      |
| 14 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | append                   |
| 15 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | com.baidu.disconf.core.common.constants.DisConfigTypeEnum                  | getType                  |
| 16 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | com.baidu.disconf.core.common.constants.DisConfigTypeEnum                  | getType                  |
| 17 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | append                   |
| 18 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | append                   |
| 19 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | append                   |
| 20 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.util.Map                                                              | keySet                   |
| 21 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.util.Set                                                              | iterator                 |
| 22 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.util.Iterator                                                         | hasNext                  |
| 23 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.util.Iterator                                                         | next                     |
| 24 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | append                   |
| 25 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | append                   |
| 26 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.util.Map                                                              | get                      |
| 27 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | append                   |
| 28 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | toString                 |
| 29 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | append                   |
| 30 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | append                   |
| 31 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuilder                                                    | toString                 |
| 32 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | append                   |
| 33 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | length                   |
| 34 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | length                   |
| 35 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | deleteCharAt             |
| 36 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getRemoteUrlParameter    | java.lang.StringBuffer                                                     | toString                 |
| 37 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getConfServerBasePathMap | java.util.Map                                                              | put                      |
| 38 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getConfServerBasePathMap | java.util.Map                                                              | put                      |
| 39 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getConfServerBasePathMap | java.util.Map                                                              | put                      |
| 40 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getConfServerBasePathMap | java.util.Map                                                              | put                      |
| 41 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooHostsUrl           | java.lang.StringBuilder                                                    | append                   |
| 42 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooHostsUrl           | java.lang.StringBuilder                                                    | append                   |
| 43 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooHostsUrl           | java.lang.StringBuilder                                                    | append                   |
| 44 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooHostsUrl           | java.lang.StringBuilder                                                    | toString                 |
| 45 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooPrefixUrl          | java.lang.StringBuilder                                                    | append                   |
| 46 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooPrefixUrl          | java.lang.StringBuilder                                                    | append                   |
| 47 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooPrefixUrl          | java.lang.StringBuilder                                                    | append                   |
| 48 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.DisconfWebPathMgr        | getZooPrefixUrl          | java.lang.StringBuilder                                                    | toString                 |
| 49 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 50 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 51 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 52 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 53 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 54 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 55 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | append                   |
| 56 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getZooBaseUrl            | java.lang.StringBuffer                                                     | toString                 |
| 57 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | joinPath                 | java.lang.StringBuilder                                                    | append                   |
| 58 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | joinPath                 | java.lang.StringBuilder                                                    | append                   |
| 59 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | joinPath                 | java.lang.StringBuilder                                                    | append                   |
| 60 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | joinPath                 | java.lang.StringBuilder                                                    | toString                 |
| 61 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getItemZooPath           | java.lang.StringBuilder                                                    | append                   |
| 62 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getItemZooPath           | java.lang.StringBuilder                                                    | append                   |
| 63 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getItemZooPath           | java.lang.StringBuilder                                                    | append                   |
| 64 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getItemZooPath           | java.lang.StringBuilder                                                    | toString                 |
| 65 | disconf-core-2.6.35.jar | com.baidu.disconf.core.common.path.ZooPathMgr               | getFileZooPath           | java.lang.StringBuilder                                                    | append                   |
```

Figure 6. Class analysis output of disconf-core-2.6.35.jar in MySQL.



```
ysql> SELECT jar_name, class_name, method_name, invoke_class_name, invoke_method_name from class_analysis where method_name = 'toString';
+-------------------------+--------------------------------------------+-------------+-------------------------+--------------------+
| jar_name                | class_name                                 | method_name | invoke_class_name       | invoke_method_name |
+-------------------------+--------------------------------------------+-------------+-------------------------+--------------------+
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | append             |
| disconf-core-2.6.35.jar | com.baidu.disconf.core.common.json.ValueVo | toString    | java.lang.StringBuilder | toString           |
+-------------------------+--------------------------------------------+-------------+-------------------------+--------------------+
8 rows in set (0.00 sec)
```

Figure 7. Query the class analysis output of disconf-core-2.6.35.jar in MySQL.

Figure 6. is the output result of disconf-core-2.6.35.jar in MySQL, based on this result, we can query any method invocation in this class. Figure 7. is an example of method toString() invocation. In this class, method toString() has been invoked 8 times, 7 times was invoked by the method append() from the class java.lang.StringBuilder, and one time was invoked by the method toString() from the class java.lang.StringBuilder. During the executing of the class RestfulMgrImpl, 153 times calling has happened, these calling is from 31 independent methods which belong to 12 differently class (Figure 8). The most call is 28 times from the class com.baidu.disconf.core.common.path.DisconfWebPathMgr. Figure 9 is the list of the methods that the 31 methods calling. It shows, there are 52 differently methods has been invoked, among which, the append () is the high frequent called method, it has been called 49 times. These 52 methods come from 33 different classes. 64 times invoked the methods belong to the java.lang.stringBuilder class (Figure 10).

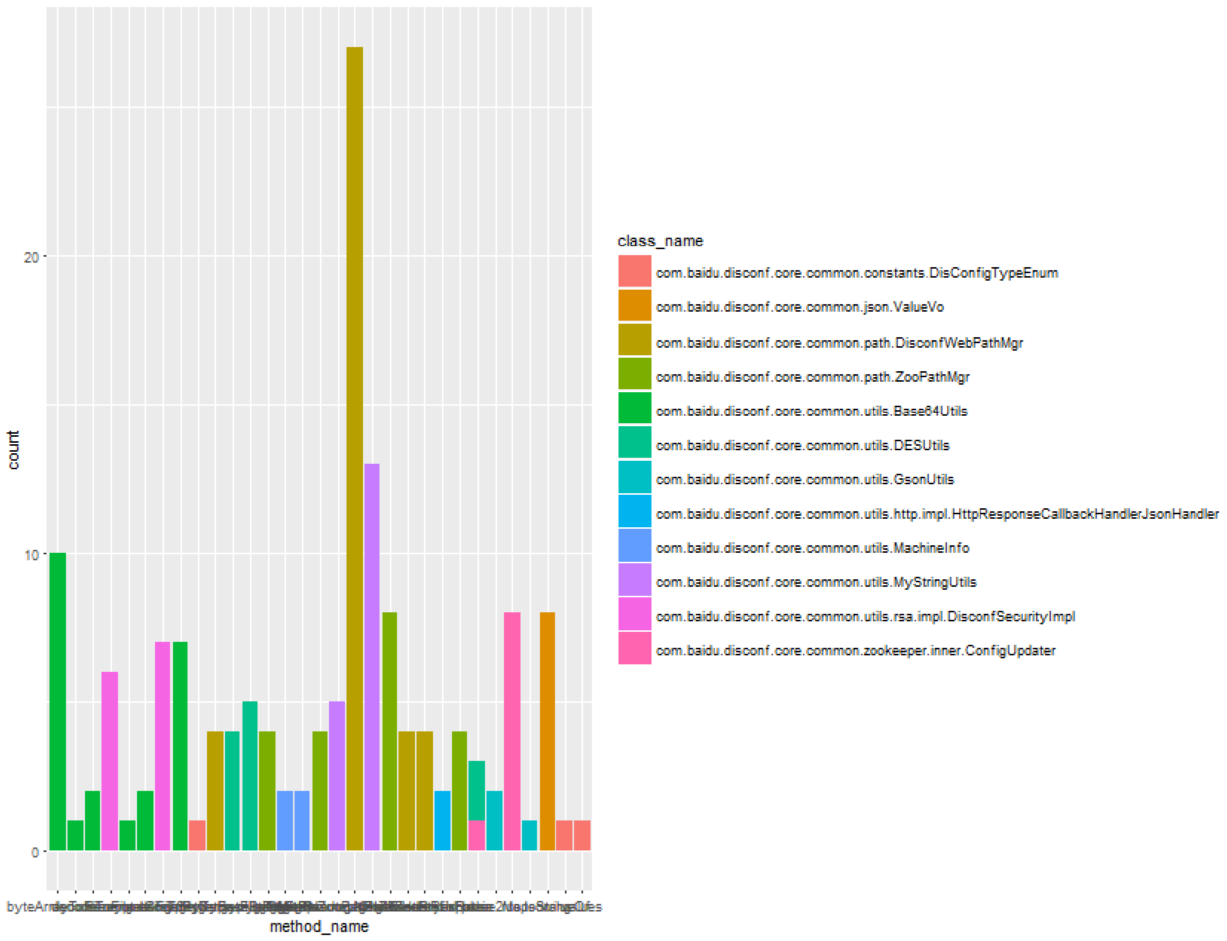


Figure 8. Plot of methods' class, x is the method name, and fill is the class that the method belongs to.

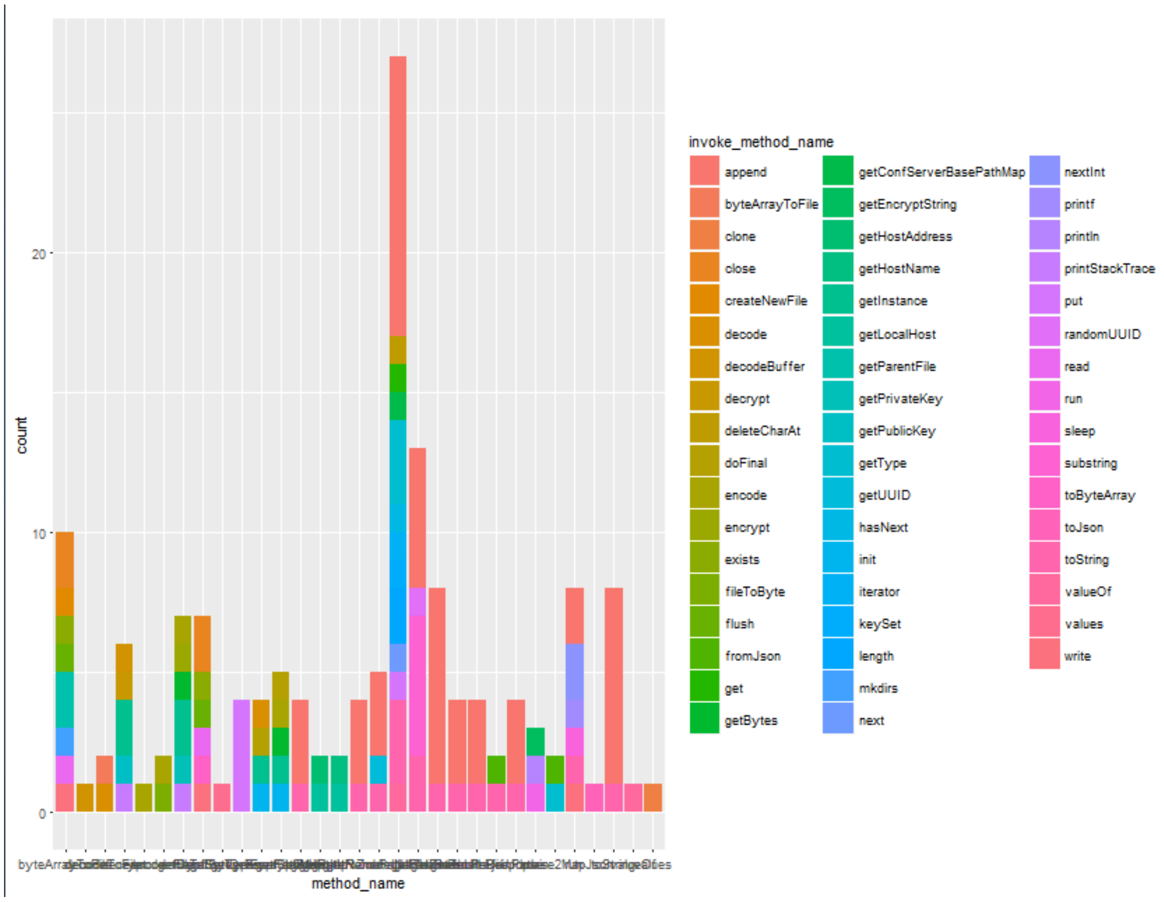


Figure 9. Plot of methods invoke other methods, x row is method of calling, and the fill is the method of been called.

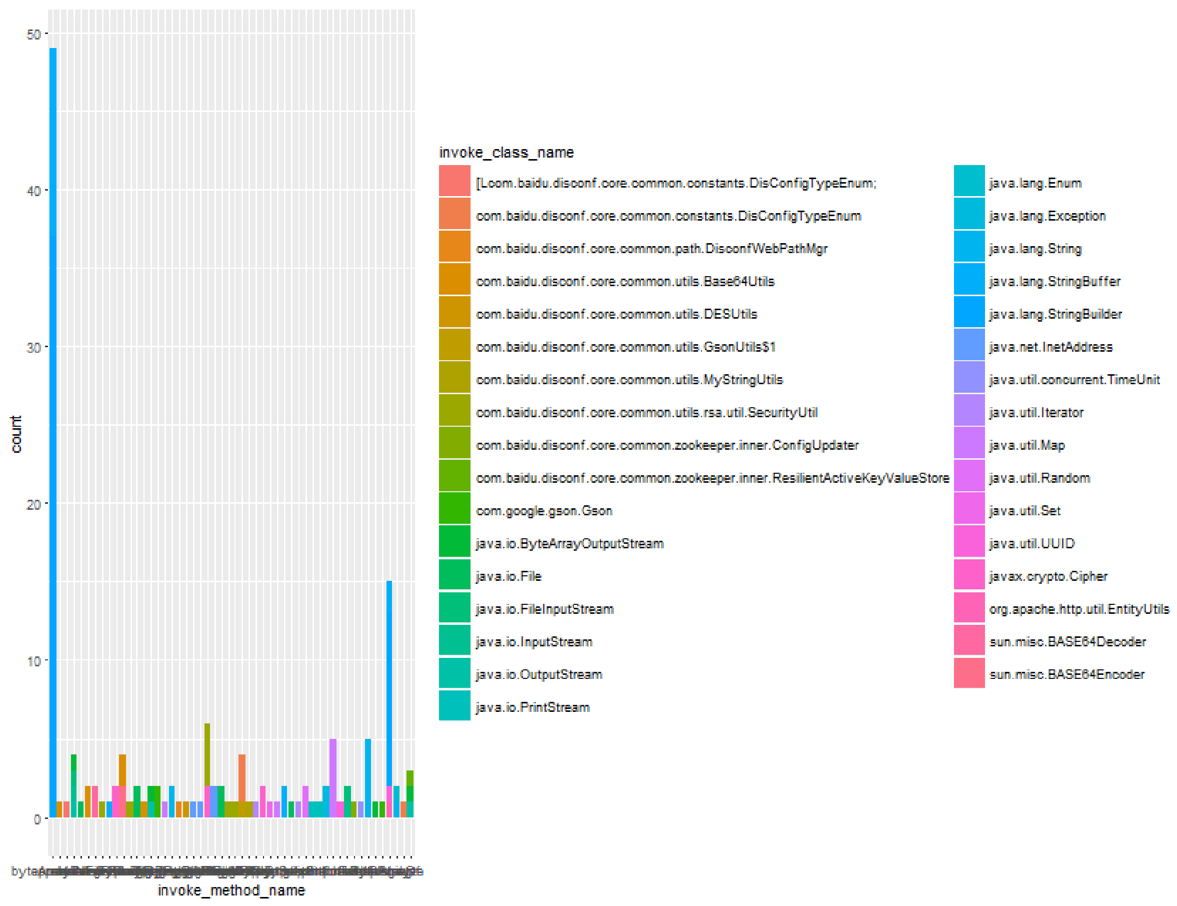


Figure 10. Plot of invoked methods' class, x is the invoked method, and fill is the class that the method belongs to.

The project classasm make it easy to analysis class. We can load any class source code or jar package and set the class name as the class that we want to analysis, run the program, we can get the clearly result of all methods invocation information in that class. The information includes the methods come from which class, calling which methods and which classes do these invoked methods belong to.

5. CONCLUSION

Although there are bunch of tools that can be used to analysis the Java class either at source code level or at compiled class level, while these tools only supply with the powerful library to use, there is no such a software package that can directly analysis the class as we want to show the methods invocation, and relationship between the methods and the classes these methods belong. To realize these functions, we need to develop a software. In this paper, we based on the ASM library, developed a new software package classasm, which can analysis both simple java class file and complexity jar package. It can output the analyzed result on the console and can output the results into MySQL database which can be conveniently queried by user. The classasm successfully realize the class analysis and data persistent into database.

REFERENCES

1. Bruneton, E., Lenglet, R., & Coupaye, T. (2002). ASM: a code manipulation tool to implement adaptable systems. *Adaptable and extensible component systems*, *30*(19).

2. Blasciak, A., & Parets, G. (1993). *U.S. Patent No. 5,265,254*. Washington, DC: U.S. Patent and Trademark Office.

3. Binder, W., Hulaas, J., & Moret, P. (2007, September). Advanced Java bytecode instrumentation. In *Proceedings of the 5th international symposium on Principles and practice of programming in Java* (pp. 135-144). ACM.

4. Lindholm, T., Yellin, F., Bracha, G., & Buckley, A. (2014). *The Java virtual machine specification*. Pearson Education.

5. Irvine, S. A., Pavlinic, T., Trigg, L., Cleary, J. G., Inglis, S., & Utting, M. (2007, September). Jumble java byte code to measure the effectiveness of unit tests. In *Testing: Academic and industrial conference practice and research techniques-MUTATION, 2007. TAICPART-MUTATION 2007* (pp. 169-175). IEEE.

6. Vasseur, A. (2004, March). Dynamic aop and runtime weaving for java-how does aspectwerkz address it. In *DAW: Dynamic Aspects Workshop* (Vol. 23).

7. Cohen, G. A., & Kaminsky, D. (1998, June). Automatic Program Transformation with JOIE. In *USENIX annual technical conference* (Vol. 98).

8. Kniesel, G., Costanza, P., & Austermann, M. (2001). Jmangler-a framework for load-time transformation of java class files. In *Source Code Analysis and Manipulation, 2001. Proceedings. First IEEE International Workshop on* (pp. 98-108). IEEE.

9. Bruneton, E. (2007). ASM 3.0 A Java bytecode engineering library. *URL: http://download. forge. objectweb. org/asm/asmguide. pdf*.

10. Müller, A. (2010, September). Bytecode analysis for checking Java access modifiers. In *Work in Progress and Poster Session, 8th Int. Conf. on Principles and Practice of Programming in Java (PPPJ 2010), Vienna, Austria*.

11. Binder, W., Hulaas, J., & Moret, P. (2007, September). Reengineering standard Java runtime systems through dynamic bytecode instrumentation. In *Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on* (pp. 91-100). IEEE.

12. Dahm, M. (1999). Byte code engineering. In *JIT'99* (pp. 267-277). Springer, Berlin, Heidelberg.

13. Muller, G., Moura, B., Bellard, F., & Consel, C. (1997, June). Harissa: A Flexible and Efficient Java Environment Mixing Bytecode and Compiled Code. In *COOTS* (pp. 1-20).

14. Lindholm, T., Yellin, F., Bracha, G., & Buckley, A, Chapter 4. The class file format. (2013, February 28). Retrieved October 2, 2017, from https://docs.oracle.com.

15. Dahm, M. (2001). Byte Code Engineering Library (BCEL) Description and usage manual. *Freie Universität Berlin, Institut für Informatik, Technischer Report B-17-98*.

16. Lee, H. B., & Zorn, B. G. (1997, December). BIT: A Tool for Instrumenting Java Bytecodes. In *USENIX Symposium on Internet technologies and Systems* (pp. 73-82).

17. Cohen, G. A., & Kaminsky, D. (1998, June). Automatic Program Transformation with JOIE. In *USENIX annual technical conference* (Vol. 98).

18. Duke University, The Java Object Instrumentation Environment. (2003, May 1). Retrieved October 2, 2017, from https://www.cs.duke.edu.

19. Class, ASM documentation 4.0. (2014, July 4). Retrieved October 2, 2017, from http://asm.ow2.org

20. Liao, Q., Distributed Configuration Management Platform. (December 10, 2016). Retrieved October 2, 2017, from https://github.com.

APPENDIX A. SOURCE CODE OF JAVA

```
/*
 * This is a class analysis program, in this class it will supply the method to analysis
 * any java class that loaded in .jar form, and will output the detail of methods execution;
 * invoking, as well as the class appending, extending and so on.
 */
package com.qiu.classasm;

/**
 *
 * @author Chengxiang Qiu
 */
import com.google.common.html.HtmlEscapers;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.util.Arrays;

import org.objectweb.asm.ClassReader;
import org.objectweb.asm.ClassVisitor;
import org.objectweb.asm.Label;
import org.objectweb.asm.MethodVisitor;
import org.objectweb.asm.Opcodes;
import org.objectweb.asm.Type;
```

```
public class ClassUtils {


    /**
     * Obtain the the method name and parameter name of the class
     *
     * @param clazz The class name of the methods that belong to
     * @param method The method that get the parameter
     * @return return the parameter list, if there is no parameter, it will
     * return null;
     */
    public static String[] getMethodParameterNamesByAsm4(Class<?> clazz, final Method
method) {
        final Class<?>[] parameterTypes = method.getParameterTypes();
        if (parameterTypes == null || parameterTypes.length == 0) {
            return null;
        }
        final Type[] types = new Type[parameterTypes.length];
        for (int i = 0; i < parameterTypes.length; i++) {
            types[i] = Type.getType(parameterTypes[i]);
        }
        final String[] parameterNames = new String[parameterTypes.length];

        String className = clazz.getName();
        int lastDotIndex = className.lastIndexOf(".");
        className = className.substring(lastDotIndex + 1) + ".class";
```

```
InputStream is = clazz.getResourceAsStream(className);
try {
    ClassReader classReader = new ClassReader(is);
    classReader.accept(new ClassVisitor(Opcodes.ASM4) {
        @Override
        public MethodVisitor visitMethod(int access, String name, String desc, String signature,
                String[] exceptions) {
            // Only execute the methods that required;
            Type[] argumentTypes = Type.getArgumentTypes(desc);
            if (!method.getName().equals(name) || !Arrays.equals(argumentTypes, types)) {
                return null;
            }
            return new MethodVisitor(Opcodes.ASM4) {
                @Override
                public void visitLocalVariable(String name, String desc, String signature, Label start,
                        Label end, int index) {
                    // In the static method the first parameter is the mehtod parameter, while in the non-static method the parameter is "this"
                    if (Modifier.isStatic(method.getModifiers())) {
                        parameterNames[index] = name;
                    } else if (index > 0 && index <= parameterTypes.length) {
                        parameterNames[index - 1] = name;
                    }
                }
```

```
                    @Override
                    public void visitMethodInsn(int opcode, String owner,
                            String name, String desc, boolean itf) {
                        System.out.println("owner:" + owner + "\t name= " + name);
                        super.visitMethodInsn(opcode, owner, name, desc, itf);
                    }

                };

            }
        }, 0);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return parameterNames;
}

public static void analysisClass(final String jarName, final String clazzName) throws Exception
{
    Class clazz = Class.forName(clazzName);
    String className = clazz.getName();
    int lastDotIndex = className.lastIndexOf(".");
    className = className.substring(lastDotIndex + 1) + ".class";
    InputStream is = clazz.getResourceAsStream(className);
    System.out.println("analysis the class :\n" + clazz.getCanonicalName());
    final String fclassName = clazz.getCanonicalName();//TODO
    String sql = "insert into class_analysis (jar_name,class_name,method_name,
invoke_class_name,invoke_method_name) values ";

    final StringBuilder sb = new StringBuilder();

    try {

      ClassReader classReader = new ClassReader(is);

      classReader.accept(new ClassVisitor(Opcodes.ASM4) {

        @Override

        public MethodVisitor visitMethod(int access, String name, String desc, String signature,

            String[] exceptions) {

          // Only execute the methods that required;

          if (name.equals("<init>") || name.equals("<clinit>")) {

            return null;

          }

          //System.out.println("method :"+name+ ",    invoke resources lists:");

          final String fmethod = name;//TODO

          return new MethodVisitor(Opcodes.ASM4) {


            @Override

            public void visitMethodInsn(int opcode, String owner,

                String name, String desc, boolean itf) {

              if (!name.equals("<init>")) {

//
      System.out.println("\t"+owner.replace("/", ".")+"."+ name);//TODO

                sb.append("(")

                    .append("'").append(jarName).append("'").append(",")

                    .append("'").append(fclassName).append("'").append(",")
```

```
                                .append("'").append(fmethod).append("'").append(",")
                                .append("'").append(owner.replace("/", ".")).append("'").append(",")
                                .append("'").append(name).append("'")
                                .append("),");
                        }
                        super.visitMethodInsn(opcode, owner, name, desc, itf);
                    }


                };


            }
        }, 0);
        String tmp = sb.toString();
        if (!"“. equals(tmp)) {
            String finalSql = sql + tmp.substring(0, tmp.length() - 1);
            System.out.println(finalSql);
            DBConnection.getInstance().executeSQL(finalSql);
        }
//                                System.out.println(sb.toString());
      } catch (IOException e) {
          e.printStackTrace();
      }
   }


   public static void main(String [] args) {
        Class clazz = null;


        clazz = Company.class;
//                  clazz = HttpClientUtil.class;
 //      clazz = com.baidu.disconf.core.common.restful.impl.RestfulMgrImpl.class;
// clazz =  HtmlEscapers.class;
////                clazz = DisconfItemCoreProcessorImpl.class;
//                  clazz = DisconfCoreMgrImpl.class;
        String className = clazz.getName();
        int lastDotIndex = className.lastIndexOf(".");
        className = className.substring(lastDotIndex + 1) + ".class";
        InputStream is = clazz.getResourceAsStream(className);
        System.out.println("analysis the class :\n" + clazz.getCanonicalName());
        try {
            ClassReader classReader = new ClassReader(is);
            classReader.accept(new ClassVisitor(Opcodes.ASM4) {
                @Override
                public MethodVisitor visitMethod(int access, String name, String desc, String signature,
                        String[] exceptions) {
                    // Only execute the methods that required;
                    if (name.equals("<init>") || name.equals("<clinit>")) {
                        return null;
                    }
                    System.out.println("method :" + name + ",    invoke resources lists:");
                    return new MethodVisitor(Opcodes.ASM4) {


                        @Override
                        public void visitMethodInsn(int opcode, String owner,
                                String name, String desc, boolean itf) {
                            if (!name.equals("<init>")) {
                                System.out.println("\t" + owner.replace("/", ".") + "." + name);
                            }
                            super.visitMethodInsn(opcode, owner, name, desc, itf);
                        }


                    };


                }
            }, 0);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


}
/**
 * This class is to connect to de Database of MySQL, visit the database;
 */
package com.qiu.classasm;
```

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;




public class DBConnection {


	public Connection getConnection(String dbType,String url,String userName,String password){

		Connection conn = null;

		if(dbType == null || "".equals(dbType)){

			dbType = "mysql";

		}


		try {

			Class.forName("com.mysql.jdbc.Driver").newInstance();

			conn = DriverManager.getConnection(url, userName, password);

		} catch (Exception e) {

			e.printStackTrace();

		}


		return conn;

	}
	private static volatile DBConnection dbConnnection;


	private Connection conn;

	private DBConnection(){

		conn                                  =                                   getConnection("mysql",
"jdbc:mysql://127.0.0.1:3306/result?useUnicode=true&characterEncoding=utf-
8&autoReconnect=true&autoReconnectForPools=true&zeroDateTimeBehavior=convertToNull",
"root", "a0308021");

	}

	public void executeSQL(String sql ){

		try {

			final Statement stmt = conn.createStatement();

			stmt.execute(sql);

		} catch (Exception e) {

			e.printStackTrace();

		} finally{


		}

	}


	public static DBConnection getInstance(){

		if(null == dbConnnection){

			synchronized (DBConnection.class) {

				if(dbConnnection == null){

					dbConnnection =  new DBConnection();
```

```
                    }
                }
            }

            return dbConnnection;
        }

        public static void main(String[] args) {
            Connection conn =new DBConnection().getConnection("mysql",
"jdbc:mysql://192.168.130.62:3306/paytest?useUnicode=true&characterEncoding=utf-8&autoReconnect=true&autoReconnectForPools=true&zeroDateTimeBehavior=convertToNull",
"root", "a0308021");
            try {
                final Statement stmt = conn.createStatement();



            } catch (Exception e) {
                e.printStackTrace();
            }finally{

            }
        }
}
/*
 * This class supply mehtods to access the package that we want to analysis, and get all the information
 * that need for the class anaylyis, and then analysis the methods in detail
 */
package com.qiu.classasm;

/**
 *
 * @author Chengxiang Qiu
 */
import com.qiu.classasm.ClassUtils;
import java.io.File;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.List;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;

public class PackageUtil {

        public static void main(String[] args) throws Exception {
                String packageName = "com.baidu";
                String jarName = "disconf-core-2.6.35.jar";
                // List<String> classNames = getClassName(packageName);
                List<String> classNames = getClassName(packageName, true);
```

```
		List<String> nclassNames = new LinkedList<String>();
		if (classNames != null) {
			for (String className : classNames) {
				if(className.contains("$")){
					continue;
				}
				nclassNames.add(className);
//				System.out.println(className);
				try{
					ClassUtils.analysisClass(jarName,className);
				}catch(Error e){
					//System.out.println(className);
				}finally{
				}
			}
		}

		System.out.println(nclassNames.size());
	}

	/**
	 * Obtain the the method name and parameter name of the class
	 * @param packageName
	 * @return class name
	 */
	public static List<String> getClassName(String packageName) {
		return getClassName(packageName, true);
	}

	/**
	 * Obtain the the method name and parameter name of the class
	 * @param packageName
	 * @param childPackage
	 * @return class name
	 */
	public static List<String> getClassName(String packageName, boolean childPackage) {
		List<String> fileNames = null;
		ClassLoader loader = Thread.currentThread().getContextClassLoader();
		String packagePath = packageName.replace(".", "/");
		URL url = loader.getResource(packagePath);
		if (url != null) {
			String type = url.getProtocol();
			if (type.equals("file")) {
				fileNames = getClassNameByFile(url.getPath(), null,
childPackage);
			} else if (type.equals("jar")) {
				fileNames = getClassNameByJar(url.getPath(), childPackage);
			}
		} else {
			fileNames = getClassNameByJars(((URLClassLoader) loader).getURLs(),
packagePath, childPackage);
```

```
				}

				return fileNames;

			}



			/**

			 * Obtain the the method name and parameter name of the class

			 * @param filePath

			 * @param className

			 * @param childPackage

			 * @return class name

			 */

			private static List<String> getClassNameByFile(String filePath, List<String> className,
		boolean childPackage) {

				List<String> myClassName = new ArrayList<String>();

				File file = new File(filePath);

				File[] childFiles = file.listFiles();

				for (File childFile : childFiles) {

					if (childFile.isDirectory()) {

						if (childPackage) {


			myClassName.addAll(getClassNameByFile(childFile.getPath(),		myClassName,
		childPackage));

						}

					} else {

						String childFilePath = childFile.getPath();

						if (childFilePath.endsWith(".class")) {
```

```
                        childFilePath                                              =
childFilePath.substring(childFilePath.indexOf("\\classes") + 9, childFilePath.lastIndexOf("."));
                        childFilePath = childFilePath.replace("\\", ".");
                        myClassName.add(childFilePath);
                    }
                }
            }

            return myClassName;
        }

        /**
         * Obtain all class of the package from jar
         * @param jarPath
         * @param childPackage
         * @return class name
         */
        private static List<String> getClassNameByJar(String jarPath, boolean childPackage) {
            List<String> myClassName = new ArrayList<String>();
            String[] jarInfo = jarPath.split("!");
            String jarFilePath = jarInfo[0].substring(jarInfo[0].indexOf("/"));
            String packagePath = jarInfo[1].substring(1);
            try {
                JarFile jarFile = new JarFile(jarFilePath);
                Enumeration<JarEntry> entrys = jarFile.entries();
                while (entrys.hasMoreElements()) {
                    JarEntry jarEntry = entrys.nextElement();
                    String entryName = jarEntry.getName();
                    if (entryName.endsWith(".class")) {
                        if (childPackage) {
                            if (entryName.startsWith(packagePath)) {
                                entryName      =      entryName.replace("/",
".").substring(0, entryName.lastIndexOf("."));
                                myClassName.add(entryName);
                            }
                        } else {
                            int index = entryName.lastIndexOf("/");
                            String myPackagePath;
                            if (index != -1) {
                                myPackagePath   =   entryName.substring(0,
index);
                            } else {
                                myPackagePath = entryName;
                            }
                            if (myPackagePath.equals(packagePath)) {
                                entryName      =      entryName.replace("/",
".").substring(0, entryName.lastIndexOf("."));
                                myClassName.add(entryName);
                            }
                        }
                    }
                }
```

```
		} catch (Exception e) {
			e.printStackTrace();
		}
		return myClassName;
	}

	/**
	 * Search the package from jar and obtain all classes in the package;
	 * @param urls
	 * @param packagePath
	 * @param childPackage
	 * @return class name
	 */
	private static List<String> getClassNameByJars(URL[] urls, String packagePath, boolean
childPackage) {
		List<String> myClassName = new ArrayList<String>();
		if (urls != null) {
			for (int i = 0; i < urls.length; i++) {
				URL url = urls[i];
				String urlPath = url.getPath();
				// class folder that not necessary to search
				if (urlPath.endsWith("classes/")) {
					continue;
				}
				String jarPath = urlPath + "!/" + packagePath;
				myClassName.addAll(getClassNameByJar(jarPath, childPackage));
```

```
                }
            }
            return myClassName;
        }
}
package com.qiu.classasm;

public class People {

        public void test(){
                System.out.println("test");
                test0();
        }

        public void test0(){
                System.out.println("test0");
                  StackTraceElement[] stackElements = Thread.currentThread().getStackTrace();
                   if (stackElements != null) {
                       for (int i = stackElements.length - 1; i >= 0; i--) {
                           System.out.print(stackElements[i].getClassName() + "\t");
                           System.out.print(stackElements[i].getMethodName() + "\t");
                           System.out.print(stackElements[i].getFileName() + "\t");
                           System.out.println(stackElements[i].getLineNumber());
                       }
                   }
```

```
		System.out.println();
	}
}
package com.qiu.classasm;

public class Employee {

	public void info(){

	}
}
package com.qiu.classasm;

public class Company {

	public void show(){
		People p = new People();
		p.test();

		p.test0();
		this.showEmployee();
	}

	public void showEmployee(){
		Employee e = new Employee();
```

```
                e.info();
        }
}
```

APPENDIX B. SOURCE CODE OF MYSQL

```
CREATE database result;

use result;

CREATE TABLE `class_analysis` (

  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,

  `jar_name` varchar(50) NOT NULL,

  `class_name` varchar(255) NOT NULL,

  `method_name` varchar(50) NOT NULL,

  `invoke_class_name` varchar(255) NOT NULL,

  `invoke_method_name` varchar(50) NOT NULL,

  PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```