International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 4, April 2014 ISSN 2319 - 4847

SKILS FRAMEWORK for Runtime Java Compilation

Prof.S.R.Gote[1], Ishwari Doshi [2], Lipi Kalita[3], Kaushal Acharya[4]

[1,2,3&4]Sinhgad Institute of Technology and Science

Abstract

Generally Java Program has to be compiled to generate .class file after editing it by using JAVAC command. SKILS framework is an Interactive Editor/Framework which can generate bytecode at the time of editing the java program. It will reduce java compilation time, because there is no need to compile a java program. So SKILS framework skips compilation phase of java. For on the fly bytecode generation SKILS uses JAVASSIST package which consists of several classes and libraries which are helpful for bytecode manipulation. Javassist provides two levels of API, source level and bytecode level. Unlike other similar systems, Javassist provides source-level abstraction; programmers can modify a class file without detailed knowledge of the Java bytecode.

Keywords- SKILS framework, javassist package, source level abstraction, on the fly bytecode generation

1. INTRODUCTION

SKILS framework is basically useful for REDUCING JAVA COMPILE TIME. The SKILS framework is specially designed for the purpose of generating Byte code without using Javac command. This framework acts as a Smart Editor which can perform operations like writing of code, on the fly Byte code, creation of .class File, Save, Log Details [Name, Date, and Time].

Java Developer Kit contains tools needed to develop the Java programs, and JRE (Java Runtime Environment) to run the programs. The tools include compiler (javac.exe), Java application launcher (java.exe), Applet viewer etc Compiler converts source java code into byte code. Java application launcher opens a JRE, loads the class, and invokes its main method. You need JDK, if at all you want to write your own programs, and to compile them. For running java programs, JRE is sufficient. JRE is targeted for execution of Java files i.e. JRE = JVM + Java Packages Classes (like util, math, lang, awt, swing etc) +runtime libraries. JDK is mainly targeted for java development. That means you can create a Java file (with the help of Java packages), compile a Java file and run a java file. Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc. Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE. If you want to run any java program, you need to have JRE installed in the system. To avoid compilation time the technique to be used is Javassist (Java programming assistant) which is a load-time reflective system for Java. It is a class library for editing byte codes in Java; it enables Java programs to define a new class at runtime and to modify a class file before the JVM loads it. Unlike other similar systems, Javassist provides source-level abstraction; programmers can modify a class file without detailed knowledge of the Java bytecode. They do not have to even write an inserted bytecode sequence; Javassist instead can compile a fragment of source text on line (for example, just a single statement). This ease of use is a unique feature of Javassist against other tools.

The Javaassist system consists of two components: the Javassist compiler and the Javassist engine. The Javassist compiler processes annotated import declarations and the Javassist engine compiles CtClass objects. In the current implementation, the Javassist engine executes an external Java compiler such as javac to compile a CtClass. In a future version, the Javassist engine will be a stand-alone component not using an external compiler. This version will achieve faster compilation because Javassist does not require the full capability to compile a Java program. Especially, Javassist does not allow programmers to directly specify a method body. The methods added to a CtClass object perform nothing except calling trap Method call() on a metaobject. This makes code generation for method bodies very simple and fast. Note that the Javassist engine does not compile a metaobject. It is separately compiled by a regular Java compiler in advance. Javassist is a programming tool for assisting Java programmers. Java programmers can use Javassist to automate some class definitions.

2. LITERATURE SURVEY

Java is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others. You can use Java to write computer applications that play games, store data or do any of the thousands of other things computer software can do. Compared to other programming languages, Java is most similar to C. However although Java shares much of C's syntax, it is not C. What's most special about Java in relation to other programming languages is that it lets you write special programs called applets that can be downloaded from the Internet and played safely within a web browser.

We dig out several options such as ASM toolkit, javassist, GNU & JIT Compiler. In which we found that javassist is the best approach to proceed further. It contains two Levels of APIs unlike other Editors. We have used & gone through the functionality of many classes in javassist packages such as Ctclass & ClassPool. Ctclass is obtained from ClassPool which extends the other classes defined in the same program. In future for code optimization, we are going to use "Soot" which is an open source java bytecode library.

3. KEY CONCEPTS

A. JAVA COMPILATION

Generally Java Program has to be compiled to generate .class file after editing it. We are developing Interactive Editor/Framework which can generate .class file directly once java program is saved. Following figure (Fig 1) shows the process of compilation. The Virtual machine code is not machine specific. The machine specific code is generated by Java interpreter by acting as an intermediary between the virtual machine and real machines shown below (Fig. 2). Java Object Framework act as the intermediary between the user programs and the virtual machine which in turn act as the intermediary between the operating system and the Java Object Framework.

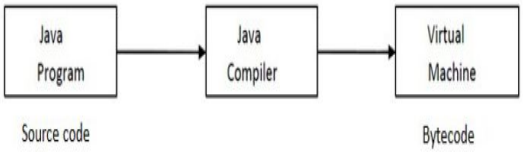

Fig. 3.1: Compilation Part-1

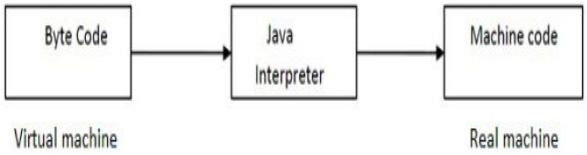

Fig. 3.2: Compilation Part-2

B. JAVA VIRTUAL MACHINE

Compiler converts java code into byte code. Java application launcher opens a JRE, loads the class, and invokes its main method. You need JDK, if at all you want to write your own programs, and to compile them. For running java programs, JRE is sufficient. JRE is targeted for execution of Java files i.e. JRE = JVM + Java Packages classes (like util, math, lang, awt, swing etc)+runtime libraries. JDK is mainly targeted for java development. i.e. you can create a Java file (with the help of Java packages), compile a Java file and run a java file. Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc. Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE. If you want to run any
Java program, you need to have JRE installed in the system.

C. PARSING

Parsing is one of the major phase in SKILS framework as it accomplish the task of fetching each instruction of java program at the time of editing of it and divide that fetched instruction into several number of tokens. SKILS identifies the end of any instruction by detecting next line character which is "\n". After that SKILS parse this fetched instruction into tokens where a single token is a string of words which is separated by another string of words by a simple blank space which is " ".

The input of this parsing phase is fetched instruction of java program and the output is number of tokens which are obtained by parsing that instruction and which are meaningful. But in this parsing phase there maybe some tokens which are not necessary for bytecode generation, so SKILS avoids such tokens which are not required. This parsing phase forwards the essential tokens to the next phase which is generation of bytecode.

D. GENERATION OF BYTECODE

This phase accomplish the task of on the fly bytecode generation for accepted tokens from the previous phase. For this purpose SKILS uses "javassist" package of java programming language. Javassist is load time reflective system of java. This javassist is collection of classes and libraries which are useful for bytecode manipulation. It enables SKILS to define a new class at runtime and to modify a class when JVM(Java Virtual Machine) loads it. It can compile just single statement of source text on line.

Bytecode is present in .class file. Each class file contains one java class or interface. CtClass(compile time class)is an abstract representation of class file. ClassPool object is a container of CtClass object representing a class file. To modify

the definition of class SKILS obtains a reference to CtClass object representing a class from ClassPool object. In short ClassPool is a hash table of CtClass objects which uses the class names as the key. Once you obtain the CtClass object you can modify corresponding class file. The change is reflected on original class file when write File() method of CtClass is called.

4. SYSTEM MODEL

A. SYSTEM ARCHITECTURE

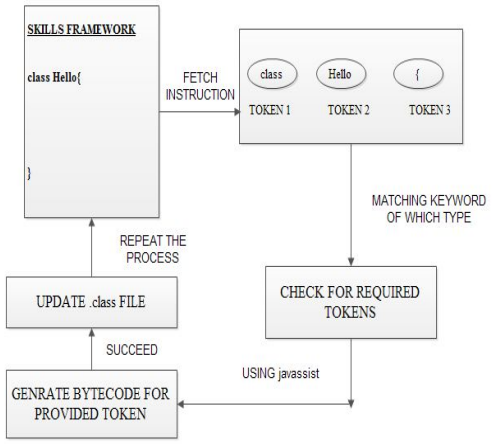

Fig. 4.1 System Model

Our proposed architecture contains 3blocks. First block contains two steps which is used for creating .class file & corresponding generation of bytecode. Programmer will write the code. At the time of writing using javaassist package helps to convert in the bytecode generation using ctclass and ctpool class. Third block at the end goes on continuing, waiting for fetching next instructions. Above figure shows the proposed architecture of the system.

B. MAIN ARCHITECTURE DIAGRAM

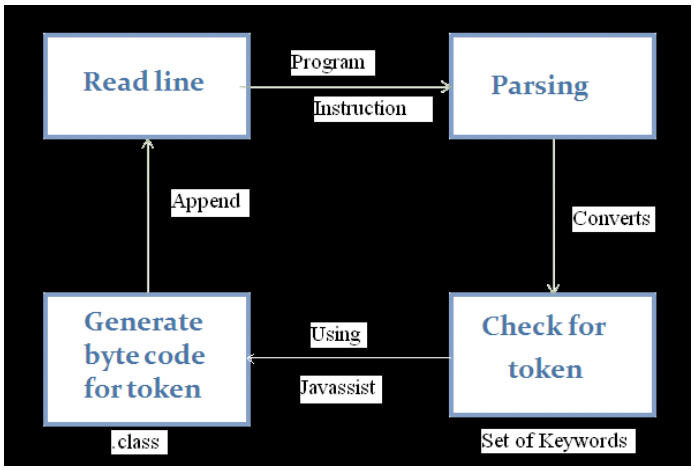

Fig. 4.2 Architecture Diagram

As per given in Fig. 4.2 the conversion takes places from one block to another i.e. from taking input from read line ,parse it through program instruction and convert it into tokens as per given in Fig. 4.1. This process is goes on continuing.

C. SYSTEM WORKING

After fetching the instruction SKILS framework analyses the fetched instruction and then decides type of fetched instruction and parsing is done and at the time of writing the code, line by line each token get converted into bytecode in the corresponding class file. Using tokenization in parsing step class name gets identified and by regular expression we can create .class file for corresponding name in the corresponding path at the time of execution.

In Fig. 4.3 for example, if it is class declaration instruction then SKILS replace class keyword by blank and create .class file of remaining word and then skils wait for next line instruction for on the fly bytecode generation. Using javassist package this runtime creation of bytecode in .class file can be done.

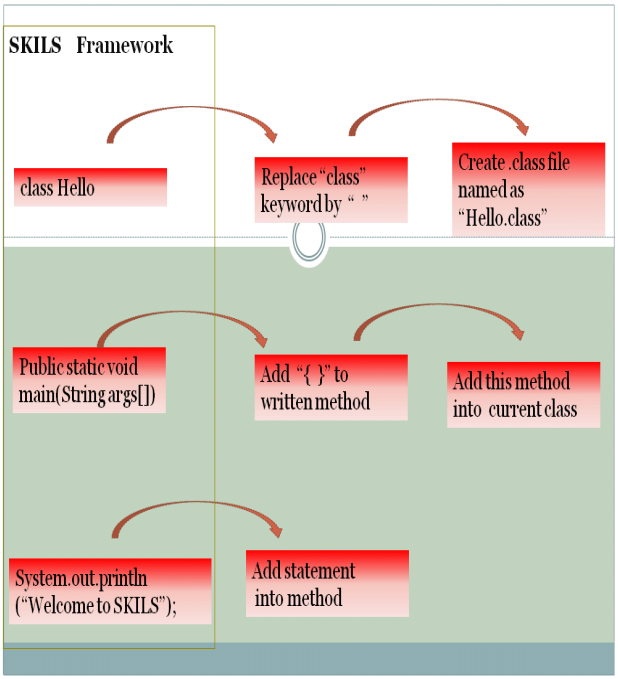

Fig. 4.3 System Working

5. REFRENCES

[1] "A GCC-based Java Implementation" (1997, IEEE), was presented at IEEE Compcon, Spring 1997
[2] The Java Virtual Machine Specification by Tim Lindholm and Frank Yellin, Addison-Wesley, Second edition, 1999.Stephen Gilmore, Javier Esparza, February 6, 2003.
[3] SCJP 9th edition https://java.net/downloads/jfjug/SCJP.
[4] JIT related concepts http://stackoverow.com.
[5] Javassist (Java Programming Assistant) http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/
[6] Bytecode Assembler, http://www.meurrens.org/ipLinks/Java/codeEngineering/blackDown/jas.html
[7] Parser Generator, http://www.meurrens.org/ip-Links/Java/codeEngineering/blackDown/jell.html.
[8] A Lexical Analyzer Generator for Java, http://www.cs.princeton.edu/~appel/modern/java/JLex .