Java Virtual Machine Compilers Java (programming language) +2

In Java, what exactly will the JVM interpreter and the JIT compiler do with the bytecode?

This question previously had details. They are now in a comment.

Answer Interesting · 7 Request 1

2 Answers

Vicky Singh, Learning Java
Answered Dec 3, 2015

The Just-In-Time (JIT) compiler is a component of the Java™ Runtime Environment that improves the performance of Java applications at run time. Java programs consists of classes, which contain platform-neutral bytecodes that can be interpreted by a JVM on many different computer architectures. At run time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation. The additional processor and memory usage during interpretation means that a Java application performs more slowly than a native application. The JIT compiler helps improve the performance of Java programs by compiling bytecodes into native machine code at run time.

The JIT compiler is enabled by default, and is activated when a Java method is called. The JIT compiler compiles the bytecodes of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it. Theoretically, if compilation did not require processor time and memory usage, compiling every method could allow the speed of the Java program to approach that of a native application.

JIT compilation does require processor time and memory usage. When the JVM first starts up, thousands of methods are called. Compiling all of these methods can significantly affect startup time, even if the program eventually achieves very good peak performance.

In practice, methods are not compiled the first time they are called. For each method, the JVM maintains a call count, which is incremented every time the method is called. The JVM interprets a method until its call count exceeds a JIT compilation threshold. Therefore, often-used methods are compiled soon after the JVM has started, and less-used methods are compiled much later, or not at all. The JIT compilation threshold helps the JVM start quickly and still have improved performance. The threshold has been carefully selected to obtain an optimal balance between startup times and long term performance.

After a method is compiled, its call count is reset to zero and subsequent calls to the method continue to increment its count. When the call count of a method

Related Questions

- If a JIT compiler compiles the bytecode into machine code, what is the interpreter doing? Does it simply run the machine code? Do JIT and the ...
- What is the difference between a JVM and JIT compiler (just in time compiler) in Java?
- How does the Java interpreter (JVM) convert bytecode into machine code?
- Why do we call it "JIT compiler" and not "JIT interpreter" to refer to the thing that converts the Java bytecode to the machine code?
- Is Java a compiled language or interpreted? What is the difference? What is the JIT compiler?
- Is it possible to compile C#-Bytecode into a real "assembler-binary" (No JIT-Compilation)?
- For what purpose Java bytecode was designed to be interpreted?
- The JVM interprets bytecode. In general, is every interpreter a virtual machine?
- Is JVM the interpreter of Java?
- What are the differences between bytecode and machine code? Is bytecode specific to Java only?

+ Ask New Question

More Related Questions

Question Stats

7 Publicly Interested

3,465 Views

Last Asked Sep 2, 2015

Edits

compilation. This process is repeated until the maximum optimization level is reached. The busiest methods of a Java program are always optimized most aggressively, maximizing the performance benefits of using the JIT compiler. The JIT compiler can also measure operational data at run time, and use that data to improve the quality of further recompilations.
The JIT compiler can be disabled, in which case the entire Java program will be
reted. Disabling the JIT compiler is not recommended except to diagnose
k around JIT compilation problems.

Source: IBM Knowledge Center

2.2k Views · View Upvoters

Your response is private.

Is this answer still relevant and up to date? Yes No

Upvote · 19 Downvote

Add a comment... Recommended All

Promoted by DigitalOcean

Deploy your Python project - free $100/60-day credit.

Rapidly provision thousands of cloud servers in seconds with firewalls, monitoring and alerts included.

Learn more at try.digitalocean.com

Umang Galaiya, Making my computer do stuff for me, just so I can be lazy.
Answered Sep 2, 2015

I read a lot about Java in the last week, so I hope I can answer this as simply as I can.

For a complied language, everything is converted into '0's and '1's, or binary, if you will. Because the computer only understands binary. Binary strings can be converted to hexadecimal strings by using converting four binary digits to a one hexadecimal digit, and padding the binary string with zeroes on the left if need be. Execution of a complied program is essentially faster, because the computer knows which sequence of digits belong to what operation, and the operations that are performed are almost always optimized to run as fast as possible.

On the other hand, in interpreted languages, each statement is parsed (humans read a sentence and understand what it means, computers parse it) before it is executed. This produces a substantial overhead as each statement has to be parsed every time the program is executed.

Java implements a combination of both of these. It compiles the program into bytecode. The JVM interprets the bytecode and performs the operations. So basically, we have a combination of both, compilation, and interpretation.

Moving on to your question, if you open a .class file that is compiled from a .java