What are advantages of bytecode over native code? [closed]

Asked 16 years, 3 months ago Modified 11 years, 6 months ago

Viewed 34k times

41

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 11 years ago.

It seems like anything you can do with bytecode you can do just as easily and much faster in native code. In theory, you could even retain platform and language independence by distributing programs and libraries in bytecode then compiling to native code at installation, rather than JITing it.

So in general, when would you want to execute bytecode instead of native?

java .net bytecode

Share

Improve this question

Follow

edited Sep 7, 2008 at 4:46

asked Sep 7, 2008 at 4:36

Jay Conrod

29.6k ● 20 ● 99 ● 110

8 Answers

Sorted by: Highest score (default)

33

Hank Shiffman from SGI said (a long time ago, but it's till true):

> There are three advantages of Java using byte code instead of going to the native code of the system:
>
> 1. **Portability**: Each kind of computer has its unique instruction set. While some processors include the instructions for their predecessors, it's generally true that a program that runs on one kind of computer won't run on any other. Add in the services provided by the operating system, which each system describes in its own unique way, and you have a compatibility problem. In general, you can't write and compile a program for one kind of system and run it on any other without a lot of work. Java gets around this limitation by inserting its virtual

> machine between the application and the real environment (computer + operating system). If an application is compiled to Java byte code and that byte code is interpreted the same way in every environment then you can write a single program which will work on all the different platforms where Java is supported. (That's the theory, anyway. In practice there are always small incompatibilities lying in wait for the programmer.)
>
> 2. **Security**: One of Java's virtues is its integration into the Web. Load a web page that uses Java into your browser and the Java code is automatically downloaded and executed. But what if the code destroys files, whether through malice or sloppiness on the programmer's part? Java prevents downloaded applets from doing anything destructive by disallowing potentially dangerous operations. Before it allows the code to run it examines it for attempts to bypass security. It verifies that data is used consistently: code that manipulates a data item as an integer at one stage and then tries to use it as a pointer later will be caught and prevented from executing. (The Java language doesn't allow pointer arithmetic, so you can't write Java code to do what we just described. However, there is nothing to

> prevent someone from writing destructive byte code themselves using a hexadecimal editor or even building a Java byte code assembler.) It generally isn't possible to analyze a program's machine code before execution and determine whether it does anything bad. Tricks like writing self-modifying code mean that the evil operations may not even exist until later. But Java byte code was designed for this kind of validation: it doesn't have the instructions a malicious programmer would use to hide their assault.
>
> 3. **Size**: In the microprocessor world RISC is generally preferable over CISC. It's better to have a small instruction set and use many fast instructions to do a job than to have many complex operations implemented as single instructions. RISC designs require fewer gates on the chip to implement their instructions, allowing for more room for pipelines and other techniques to make each instruction faster. In an interpreter, however, none of this matters. If you want to implement a single instruction for the switch statement with a variable length depending on the number of case clauses, there's no reason not to do so. In fact, a complex instruction set is an advantage for a web-based language: it means that the same

> program will be smaller (fewer instructions of greater complexity), which means less time to transfer across our speed-limited network.

So when considering byte code vs native, consider which trade-offs you want to make between portability, security, size, and execution speed. If speed is the only important factor, go native. If any of the others are more important, go with bytecode.

I'll also add that maintaining a series of OS and architecture-targeted compilations of the same code base for every release can become very tedious. It's a huge win to use the same Java bytecode on multiple platforms and have it "just work."

Share Improve this answer Follow

answered Sep 7, 2008 at 4:46

Sean
4,670 ● 1 ● 21 ● 18

7 4 years later... Portability: compilers that produce native code can cross-compile, like gc (The official Go compiler) which makes it as simple. Security: Native Client runs native code in a sandbox thus limiting it's permissions. Size: rarely an issue these days, even for mobile devices. – Zippo Aug 1, 2012 at 14:08

3 @Zippoxer What's with the four years? Cross-compilation is a very old concept. But you still have to compile the code for each platform separately. Virtualization is also not a new concept, but virtualizing code written for native execution is not the same as virtualizing code which is specifically designed to be run in a sandbox. As for the size, I actually

wouldn't call Java bytecode CISC at all. The same goes for CIL. – Malcolm Nov 6, 2012 at 2:54

16

The performance of essentially any program will improve if it is compiled, executed with profiling, and the results fed back into the compiler for a second pass. The code paths which are actually used will be more aggressively optimized, loops unrolled to exactly the right degree, and the hot instruction paths arranged to maximize I$ hits.

All good stuff, yet it is almost never done because it is annoying to go through so many steps to build a binary.

This is the advantage of running the bytecode for a while before compiling it to native code: profiling information is automatically available. The result after Just-In-Time compilation is highly optimized native code for the specific data the program is processing.

Being able to run the bytecode also enables more aggressive native optimization than a static compiler could safely use. For example if one of the arguments to a function is noted to always be NULL, all handling for that argument can simply be omitted from the native code. There will be a brief validity check of the arguments in the function prologue, if that argument is not NULL the VM aborts back to the bytecode and starts profiling again.

Share Improve this answer Follow

answered Sep 7, 2008 at 4:56

DGentry
16.3k ● 8 ● 53 ● 66

10

Bytecode creates an extra level of indirection.

The advantages of this extra level of indirection are:

- Platform independence
- Can create any number of programming languages (syntax) and have them compile down to the same bytecode.
- Could easily create cross language converters
- x86, x64, and IA64 no longer need to be compiled as seperate binaries. Only the proper virtual machine needs to be installed.
- Each OS simply needs to create a virtual machine and it will have support for the same program.
- Just in time compilation allows you to update a program just by replacing a single patched source file. (Very beneficial for web pages)

Some of the disadvantages:

- Performance
- Easier to decompile

Share Improve this answer Follow

edited Sep 7, 2008 at 4:51

answered Sep 7, 2008 at 4:44

Brian R. Bondy
347k ● 126 ● 602 ● 640

4

All good answers, but my hot-button has been hit - performance.

If the code being run spends all its time calling library/system routines - file operations, database operations, sending windows messages, then it doesn't matter very much if it's JITted, because most of the clock time is spent waiting for those lower-level operations to complete.

However, **if** the code contains things we usually call "algorithms", that have to be fast and don't spend much time calling functions, **and** if those are used often enough to be a performance problem, then JIT is very important.

Share Improve this answer Follow

answered Jan 6, 2009 at 21:22

Mike Dunlavey
40.6k ● 15 ● 94 ● 138

2

I think you just answered your own question: platform independence. Platform-independent bytecode is produced and distributed to its target platform. When executed it's quickly compiled to native code either before execution begins, or simultaneously (Just In Time). The Java JVM and presumably the .NET runtimes operate on this principle.

Share Improve this answer Follow

answered Sep 7, 2008 at 4:38

Kyle Cronin
79k ● 45 ● 151 ● 167

2

Here:
http://slashdot.org/developers/02/01/31/013247.shtml

Go see what the geeks of Slashdot have to say about it! Little dated, but very good comments!

Share Improve this answer Follow

answered Sep 7, 2008 at 4:56

Marcio Aguiar
14.5k ● 6 ● 40 ● 42

1

Ideally you would have portable bytecode that compiles Just In Time to native code. I think the reason bytecode interpreters exist without JIT is due primarily to the practical fact that native code compilation adds complexity to a virtual machine. It takes time to build, debug, and maintain that additional component. Not everyone has the time or resources to make that commitment.

A secondary factor is safety. It's much easier to verify an interpreter won't crash than to guarantee the same for native code.

Third is performance. It can often take more time to generate machine code than to interpret bytecode for small pieces of code that only run once.

Share Improve this answer Follow

answered Sep 7, 2008 at 4:50

Josh Segall
4,283 ● 4 ● 32 ● 28

0

Portability and platform independence are probably the most notable advantages of bytecode over native code.

Share Improve this answer Follow

answered Apr 24, 2013 at 19:32

Fiery Phoenix
1,166 ● 3 ● 16 ● 30