JIT (Just In Time) – Compilação de Programas Java

Moisés Jardim Pinheiro
Universidade Católica de Pelotas
pinheiro@ucpel.tche.br

Resumo

O presente artigo apresenta uma visão geral da linguagem de programação Java e as fases em que um programa Java passa até ser executado, bem como o seu processo de compilação e interpretação. Também é mostrado uma das formas que melhora o desempenho de Java, que é o compilador JIT (Just In Time), sendo esse o tema foco abordado no artigo.

1. Introdução

Os sistemas java geralmente consistem em várias partes: um ambiente, a linguagem, a interface de programas aplicativos (API) Java e várias bibliotecas de classes. Os programas Java passam por 5 fases para serem executados. São as seguintes: *edição, compilação, carga, verificação e execução.*

Fase 1 – EDIÇÃO

A fase 1 consiste em editar um arquivo. Isso é realizado com um programa editor. O programador digita um programa em Java utilizando o editor e faz correções se necessário. Também pode-se utilizar de ambientes de desenvolvimento integrado (IDEs) Java como Forté for Java Community Edition, NetBeans, o JBuilder da Borland, o Visual Cafe da Symantec e o Visual Age da IBM, entre outros, têm editores embutidos que são tranparentemente integrados ao ambiente de programação.

Fase 2 – COMPILAÇÃO

O programador emite o comando *javac* para compilar o programa. O compilador Java traduz o programa Java para *bytecodes* (a linguagem entendida pelo interpretador Java). Sempre que o programa compilar corretamente, o compilador produz um arquivo *.class,* que é o arquivo que contém os bytecodes que serão interpretados durante a fase de execução.

Fase 3 – CARGA

O programa deve ser primeiramente colocado na memória antes de ser executado. Isso é feito pelo carregador de classe, que pega o arquivo (ou arquivos) . class que contém os bytecodes e o transfere para a memória. O arquivo .class pode ser carregado a partir de um disco ou através de uma rede. Há dois tipos de programas para os quais o carregador de classe carrega arquivos .class - *aplicativos e applets*. O aplicativo é um programa que normalmente é armazenado e executado a partir do computador local, já o applet é um programa pequeno que normalmente é armazenado em um computador remoto que o usuário conecta através de um web browser. Os applets são carregados no navegador a partir de um computador remoto, são executados no navegador e descartados quando se completa a execução. Para executar um applet novamente, o usuário deve direcionar o browser para a localização apropriada e recarregar o programa novamente. Os aplicativos são carregados na memória e executados com o interpretador Java (JVM) através do comando *java*. Por exemplo, ao executar um aplicativo chamado TESTE, o comando invoca a JVM para o aplicativo TESTE e faz com que o carregador de classe carregue as informações utilizadas no programa TESTE. O carregador de classe também é executado quando um web browser carrega um applet java.

Fase 4 – VERIFICAÇÃO

Antes que o interpretador Java execute os bytecodes em um applet, os bytecodes são verificados pelo *verificador de bytecode*. Isso assegura que os bytecodes para as classes que são baixadas da internet (conhecidas como classes baixadas) são válidos e não violam as restrições de segurança do Java. O Java impõe intensa segurança porque os programas Java baixados da rede não devem ser capazes de causar danos aos arquivos e ao sistema do computador (como acontece com os vírus de computadores).

Fase 5 – EXECUÇÃO

O computador, sob o controle de sua CPU, interpreta o programa, um bytecode por vez, realizando assim a ação especificada pelo programa. Você já ouviu dizer que Java é uma linguagem portável e que os programas escritos em Java podem rodar em muitos computadores diferentes. Para a programação em geral, portabilidade é um objetivo vago. Por exemplo, o documento-padrão ANSI de C contém uma longa lista de questões de portabilidade; e foram escritos livros inteiros que discutem portabilidade. Embora seja mais fácil escrever programas portáveis em

Java do que em outras linguagens de programação, há diferenças entre compiladores, interpretadores e computadores que podem tornar difícil alcançar a portabilidade. Simplesmente escrever programas em Java não garante a portabilidade. Ocasionalmente, o programador precisará lidar diretamente com variações de compilador e computador.

2. Passos de um ambiente Java típico

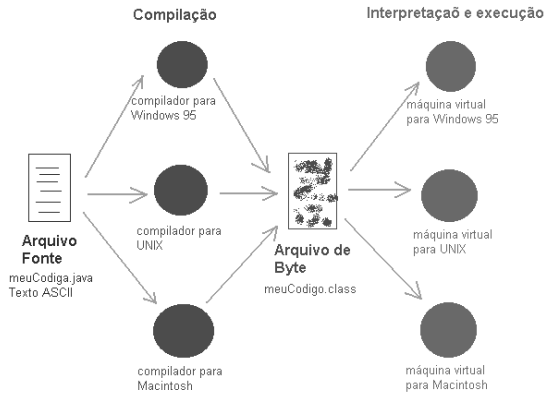
Fase 1 - O programa é criado no editor e armazenado em disco.

Fase 2 - O compilador cria bytecodes e os armazena em disco.

Fase 3 - O carregador de classe coloca bytecodes na memória.

Fase 4 - O verificador de bytecodes confirma que todos os bytecodes são válidos e não violam restrições de segurança de java.

Fase 5 - A JVM lê os bytecodes e os traduz para uma linguagem que o computador pode entender (linguagem de máquina), possivelmente armazenando valores dos dados enquanto executa o programa.



3. Java é lento?

Benchmarks mostram que Java não executa tão rápido quanto programas C++, por exemplo. Enquanto Java é compilado e interpretado, o que torna uma linguagem híbrida, C++ é apenas compilado. Mas "lentidão" é um termo relativo, e talvez a velocidade do seu programa Java não seja problema na aplicação que você estiver construindo. Tudo depende de fatores como quantos usuários você planeja atender ao mesmo tempo, quão poderoso é o seu hardware e quanto realmente sua aplicação consome processamento.

Não pense que ninguém está tentando melhorar a performance do Java. Hoje, muitas VMs possuem compiladores JIT (Just In Time) embutidos nelas. Um compilador JIT traduz o bytecode - quando ele está para ser processado - para o código de máquina nativo dos computadores. Este código de máquina é então mantido (em um buffer ou cache) de forma que ele possa ser reusado se o bytecode correspondente for executado mais de uma vez.

4. Compilação de bytecode em tempo de execução

O módulo que obtém o programa objeto e o converte em instruções de máquinas específicas para cada sistema operacional em tempo de execução, é o método de compilação chamado de JIT (Just In Time), usado pela linguagem Java da Sun e outras, para de forma otimizada melhorar significamente a performance das aplicações.

Um compilador lê um programa escrito em uma linguagem fonte e traduz o programa equivalente em uma linguagem alvo. Java foi projetado para ser compacta, independente de plataforma e para utilização em rede, o que levou a decisão de ser interpretada através do esquema de bytecode, mas também não podemos esquecer que Java é compilado, que é a fase de geração dos bytecodes para posterior interpretação dos mesmos. Em uma linguagem interpretada a performance é razoável, não podendo ser comparada a velocidade de execução do código nativo, foi criada uma técnica para superar esta limitação, a JVM utiliza um compilador Just In Time, que compila os bytecodes para código nativo durante a execução, desta forma deixando-a otimizada, o que melhora significamente a performance dos programas Java.

5. Conclusão

O compilador Java não gera código para a CPU real, mas sim para uma CPU fictícia. Chamamos a linguagem de máquina desta CPU fictícia de bytecode.

O bytecode deve ser executado (interpretado) por um emulador desta CPU, ao qual chamamos máquina virtual Java, ou JVM.

Portanto podemos esperar que Java tenha uma performance semelhante a outros ambientes interpretados, tais como visual basic, perl ou php.

Para reduzir a diferença de performance entre Java e linguagens compiladas tradicionais (C++, Pascal...) o bytecode pode ser transformado em código nativo no momento da sua carga na JVM.

O compilador JIT (parte da JVM) troca maior velocidade de execução por um maior tempo de carga.

A compilação JIT só apresenta benefícios reais para código que será executado várias vezes.

Apesar do JIT, aplicações Java ainda costumam ser mais lentas do que aplicações tradicionais, sejam elas compiladas ou interpretadas.

Java segue a mesma linha evolutiva de todas as linguagens de programação até hoje: fazer a máquina trabalhar mais para que o homem trabalhe menos. Se um aplicativo escrito em Java é visivelmente mais pesado do

que um aplicativo em C ou Pascal, ele também é escrito em menos tempo e contém menos bugs.

6. Referências

[1] H. M. Deitel e P. J. Deitel, "Java: Como Programar", 4ª Edição, Editora Bookman, 2002, 1386 pág.

[2] Lozano, Fernando, GNU Compiler for Java GCJ. Disponível em http://www.lozano.eti.br/, (18/11/2004).

[3] Página Oficial da linguagem. Disponível em http://java.sun.com/, (18/11/2004).

[4] Funari de Freitas, André. Compilador JIT. Disponível em http://atlas.ucpel.tche.br/~barbosa/consico/consico2/artigos/a5.pdf (18/11/2004).

[5] Franca Einhardt, Luciana e Alves Nickel, Kelen. Java Virtual Machine. Disponível em http://atlas.ucpel.tche.br/~barbosa/consico/consico2/artigos/a7.pdf (18/11/2004).

[6] GUJ, Grupo de Usuários Java. JIT. Disponível em http://www.guj.com.br/ (18/11/2004.