Aula 1 - Introdução a Java

Docupedia Export

Author:Silio Leonardo (CtP/ETS)
Date:29-Apr-2024 14:45

Table of Contents

- Linguagens Compiladas e Interpretadas
- Ahead-Of-Time Compilation, Representações Intermediárias e Linguagens Híbridas
- Java e Just-In-Time Compilation
- Tipagem
- Paradigmas de Programação
- Ecossistema Java
- Java CLI e Java Compiler
- Introdução a Codificação com o Java

1 Linguagens Compiladas e Interpretadas

Para que um programa seja executado em um computador é necessário que exista alguma forma de transformar o texto do código fonte em algo que o computador compreenda, o então chamado código de máquina.
Para isso, existem o que chamamos de **Compilador**. Um Compilador é um programa que recebe arquivos de código fonte de uma linguagem de programação e faz a sua conversão para código de máquina.
É importante perceber que isso significa que o programador entra com um código e recebe como saída um executável (.exe) que é capaz de ser executado em um computador. Mas note que:

- Cada com sistema operacional pode ter dependências (bilbiotecas base) e ainda ser 32 ou 64 bits (veremos isso mais a frente).
- Cada processador tem um código de máquina único.

Assim você precisa compilar para cada arquitetura computacional. Quando trabalhar com linguagens compiladas você precisa ter isso em mente. Chamamos o alvo da compilação simplismente de target.
Exemplos de linguagens compiladas: C, C++, Fortran, Cobol.
Por outro lado existe também outra técnica de tradução que é o **Interpretador**. A ideia é simples porém brilhante: Faça um programa que leia um código fonte e execute imediatamente. Por exemplo: Ao ler uma linha de código 'print("ola")' o programa imediatamente executa o 'printf' da linguagem C, por exemplo, tendo como parâmetro 'ola'. Se ele ler uma linha 'x = 4', ele imediatamente salva em algum lugar o valor 4 apontando que o mesmo se encontra numa variável chamada 'x'.
A ideia é inteligente pois podemos fazer um Interpretador para cada arquitetura e um mesmo programa pode rodar em diferentes arquiteturas, pois será interpretado por diferentes programas para diferentes arquiteturas mas que tem um mesmo funcionamento.
Por isso, linguagens como Python rodam em todo lugar: Basta ter um interpretador Python (que é escrito em C) compilador para a arquitetura destino.
O mesmo acontece com JavaScript, Css e Html (as duas últimas não são linguagens de programação, apenas linguagens de estilização e estruturação, respectivamente). Um navegador não é nada mais, nada menos, que um interpretador que recebe código online e apresenta o seu processamento.
A desvantagem das linguagens interpretadas é que costumam ser mais lentas, afinal, é preciso ler cada linha de código, interpretar o que ela faz e então executar o código de máquina equivalente escrito em C. Além disso, para que uma aplicação funcione no seu cliente, é necessário que você exponha o código original do seu software para ele.
Outra vantagem é que na linguagem interpretada você não precisa passar por um processo para que ela se torne uma linguagem executável (.exe) toda vez que quiser testá-la, tornando-a, em geral, mais ágil.

Característica	**Compilada**	**Interpretada**
Velocidade de Compilação	Lenta	Inexistente
Velocidade de Execução	Rápida	Mais Lenta
Compatibilidade sem Recompilação	Não	Sim

2 Ahead-Of-Time Compilation, Representações Intermediárias e Linguagens Híbridas

Uma técnica poderosa utilizada pelas linguagens modernas é a **Compilação Antecipada**, Ahead-Of-Time, ou simplismente AOT. Está técnica consiste em traduzir um código mais alto-nível (mais inteligível para humanos) em um código mais baixo-nível (mais próximo ao código de máquina).
Ou seja, você pode transformar JavaScript em C++, por exemplo. Mas é muito mais do que isso. A utilização mais comun é criar uma representação intermediária.
Essa representação intermediária (IR) do código pode se utilizada para diminuir o trabalho da compilação.
Por exemplo, você pode ter várias linguagens que se transformam em uma mesma IR, assim todas elas seriam compatíveis, ao passo que seria necessário apenas transformar o IR em código de máquina uma única vez.
Entenda: Se nós temos 3 linguagens e 5 arquiteturas desejadas, seguindo os conceitos tradicionais precisariamos desenvolver 15 compiladores, 1 para cada linguagem sendo convertida para cada arquitetura. Usando AOT, precisamos de 3 compiladores, de cada linguagem para a IR, e 5 compiladores, da IR para cada arquitetura. Assim 8 no total e, de quebra, ganhamos alta compatibilidade entre as linguagens.
E o mais poderoso: Podemos construir um compilador para transformar da linguagem fonte na IR e criarmos um interpretador para a IR executar. Assim temos linguagens de compilação **Híbridas**.

3 Java e Just-In-Time Compilation

A grande vantagem das linguagens Híbridas são linguagens como o **Java**. Muitas linguagens como Java, Kotlin, Scala, são convertidas na mesma IR que é levada ao cliente e interpretada em diferentes máquinas. Basta ter um software de tempo de execução chamada JVM ou Java Virtual Machine, instalada na arquitetura alvo.
Isso também permite que o mesmo código uma vez compilado (IR) execute em praticamente qualquer arquitetura, basta existir uma implementação da JVM para ela.
Além disso, o Java utliza uma ténica chamada de **Just-In-Time** ou JIT. O JIT é uma forma de interpretação aperfeiçoada: Ela compila a representação intermediária no momento em que ela deveria ser executada, possibilitando a execução diretamente na CPU do computador, realizando ainda optimizações que não poderiam ser feitas em outro momento, possibilitando que as desvantagens de linguagens interpretadas sejam liquidadas, desempenhando muito bem em diferentes arquiteturas.
Além disso, apenas o código usado é convertido, partes do código não executadas não são convertidas e as partes convertidas ficam salvas, fazendo com que em futuras execuções o desempenho da aplicação melhore ainda mais.
Ou seja, JIT é poderoso, contudo, complexo de se implementar, mas dá um alto nível de flexibilidade e compatibilidade.

4 Tipagem

Um tópico interessante da Ciência da Computação que nos ajuda a compreender melhor as características da linguagem C#, bem como outras, é a **Tipagem**.
A tipagem é como um plano Cartesiano que classifica as linguagens em 2 eixos:

- Tipagem Dinâmica vs Tipagem Estática
- Tipagem Forte vs Tipagem Fraca

Uma Tipagem Estática é uma linguagem onde os tipos de todas as variáveis são definidos ainda na fase de compilação. Ou seja, se a variável 'x' contém um número, isso será identificado ainda na compilação e 'x' só poderá receber um número. Isso acontece em linguagens como C ou Java.
Uma Tipagem Dinâmica é uma linguagem onde as variáveis podem mudar de tipos o tempo todo. Como Python e JavaScript, por exemplo. Então escrever 'x = 2' e em seguida 'x = "oi"' é aceitável numa linguagem Dinâmica, mas não em uma Estática.
Já um Tipagem Forte é onde os dados tem tipo bem definido. Por exemplo, vamos supor que criamos um variável 'x' valendo 4. Ao tentar obter o resulado da seguinte expressão 'x[0]' teríamos um erro pois 4 não é um vetor então não podemos acessar a posição zero dele.
Para uma Tipagem Fraca a expressão retornaria algum valor, mesmo que o mesmo seja tratado como 'indefinido'. O fato de não aparecer um erro faz com que não percebamos que algo deu errado. Isso pode trazer flexibilidade mas aumenta a quantidade de bugs.
Abaixo uma pequena tabela com exemplos:

Linguagem	Força	Dinamicidade
C K&R (bem antigo)	Fraca	Estática
C Ansi (mais moderno)	Forte	Estática
Java	Forte	Estática
C++	Forte	Estática
JavaScript	Fraca	Dinâmica
PHP	Fraca	Dinâmica
Python	Forte	Dinâmica
Ruby	Forte	Dinâmica

Nota: Aqui você percebe que Java e JavaScript tem muito pouco em comum.

Aqui vale uma ressalva: Permitir que várias conversões automáticas aconteçam como no JS torna uma linguagem mais fraca. Por exemplo ao computar '{} + {}', um dicionário vazio mais outro dicionário vazio se obtém '[object Object][object Object]', pois o JS tenta converter para um texto apresentável. O exemplo anterior 'x[0]' ou até mesmo '4[0]' apresenta valor indefinido. Isso é muito diferente do que fazer isso em uma linguagem como o C antigamente, onde 'x[0]' é literalmente acessar a memória do computador no endereço 4, pois o C entende (ou entendia) tudo como números e mais números e não da significado para seus valores. Assim é fácil perceber que existem diferentes níves de linguagem fraca.

5 Paradigmas de Programação

Um paradigma de programação é a maneira como uma linguagem de programação se estrutura para orientar o pensamento do programador. Existem diversas abordagens para se programar; estamos mais acostumados com a programação Imperativa e Estruturada, ou seja, dizemos como o programa deve fazer o processamento de dados e estruturamos isso em várias funções e variáveis, executando estruturalmente linha a linha.
Contudo, existem outras propostas: Ainda como linguagem imperativa temos a Orientação a Objetos, que muda a forma como modularizamos o estado (variáveis) e comportamento (métodos) do nosso programa, escondendo dentro de escopos e afastando da maneira estruturada onde deixamos tudo no mesmo programa.
Por outro lado, temos os paradigmas declarativos, onde se fala o que se deseja fazer, não se importando no como. Dentro deste mundo existem paradigmas como o Funcional, onde a definição de funções, uso de funções como dados (o que sim, parece confuso e de fato é), além de vários recursos prontos tornam a programação muito diferente da que estamos acostumados, mas pode aumentar muito a produtividade.
O C# lhe permite escrever código estruturado, embora seja uma linguagem inerentemente Orientada a Objetos. E dentro da programação imperativa, temos ainda a programação Orientada a Eventos que não será abordado neste curso, mas C# da suporte a mesma. C# também tem suporte a programação funcional entre outros aspectos de programação declarativa. Ou seja, Java é uma linguagem considerada multi-paradigma, embora com grande enfoque em OO.

6 Ecossistema Java

Java é a plataforma de desenvolvimento criado pela Sun Microsystems. Ela, em si, é muito maior que somente o Java como linguagem. Vamos explorar rapidamente a fim de entender o que é o Java. Para isso vamos aprender os seus componentes:

- Java como plataforma possui 3 linguagens principais: Java, Kotlin e Scala (uma interessante linguagem funcional). Todas são convertidas para uma linguagem intermediária (ou seja, um linguagem que é uma representação intermediária/IR) chamada Java bytecode. Ou seja, as linguagens são intercambiáveis e totalmente integráveis.
- JVM, Java Virtual Machine é a máquina virtual que transforma o Java bytecode em código nativo capaz de rodar em muitas arquiteturas diferentes.
- Java SE é uma espécie de biblioteca padrão com centenas de recursos para realizar qualquer tarefa: Trabalhar com arquivos, Criptografia, Comunicação em Redes, Desenvolvimento Web e afins. Para que programas rodem você precisa ter instalado na sua máquina as bibliotecas do .NET Framework na versão desejada.
- JDK ou Java Development Kit (SDK, Software Development Kit) é o Kit de desenvolvimento do Java. É possível instalar apenas os recursos de execução, sendo um usuário, e executar programas Java sem ter o Kit de desenvolvimento. Para os desenvolvedores, basta instalar o SDK e ter acesso as ferramentas para construir novas aplicações.
- Java CLI (Command Line Interface) é um programa que utilizamos para criar projetos, testar, executar entre várias operações no momento de gerenciar projetos em Java. Com o JDK instalado, basta utilizar o comando 'java' no terminal para utilizar o Java.

7 Java CLI e Java Compiler

- javac -d pasta arquivo: Compila um arquivo java e coloca em uma pasta específica.
- java arquivo_sem_extensão: Roda um arquivo .class e apresenta seu resultado.

8 Introdução a Codificação com o Java

```java
import java.util.Scanner;

public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");

        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();

        int sum = 0;
        while (scanner.hasNextInt()) {
            int number = scanner.nextInt();
            sum += number;
        }
        System.out.printf("Soma: %d\n", sum);
    }
}
```