Quando estamos codificando um sistema, nem sempre nos preocupamos com o trabalho feito por trás daquela programação. No entanto, saber qual método de tradução pode ajudar bastante na hora de distribuir um projeto, especialmente quando se trata de portabilidade ou desempenho.

É nessas horas que um conhecimento prévio sobre qual técnica de implementação pode ser usada pra linguagem escolhida pode ajudar. Linguagens compiladas e interpretadas têm muito por trás do que apenas converter o código para linguagem de máquina.

A pergunta de hoje é: qual utilizar na minha aplicação? Leia o artigo e descubra!

## Qual deles eu devo usar?

Interpretador ou compilador, então? Pode parecer descuidada ou sem preocupação com as peculiaridades específicas de um sistema, mas a resposta básica é assustadoramente simples e completa: tanto faz. Quê?

Isso mesmo, tanto faz. Na verdade, qualquer linguagem pode ser compilada ou interpretada, pois esses conceitos não são próprios de uma linguagem em si, ou seja, não é um atributo ou característica intrinsecamente ligada a ela.

Ambos são técnicas de implementação para executar (ou, melhor ainda, traduzir) o seu código fonte e podem ser aplicados para qualquer linguagem, desde que se tenha o tradutor necessário para tal.

Uma atitude comum a alguns programadores que reforça essa ideia é a de construir seu código em um editor de texto comum, tipo o bloco de notas. Depois de construído, o arquivo era usado em um programa para executado (quem nunca montou um HTML dessa forma?).

## Interpretador x compilador

[](https://imasters.com.br/devsecops/interpretador-x-compilador-o-que-e-melhor/)

Basicamente, um compilador é um programa (ou um conjunto deles) como qualquer outro, porém seu objetivo principal é o de **traduzir todas as suas linhas de código para outra linguagem** – normalmente, uma de alto nível para outra de baixo nível (Assembly ou linguagem de máquina). Delphi, Rust, C++ e Swift figuram na lista de compiladas.

Um interpretador também é um programa, mas, ao contrário do compilador, ele não converte o código todo para linguagem de máquina de uma vez. Ele executa diretamente cada instrução, passo a passo. MATLAB, Lisp, Perl e PHP são apontadas como interpretadas.

Em suma, a grande diferença está na forma de execução. Enquanto um compilador analisa todo o código a fim de traduzi-lo de uma vez (muitas vezes, o resultado é um arquivo executável ou uma biblioteca), o interpretador faz esse trabalho de conversão aos poucos, sempre que uma declaração ou função é executada, por exemplo.

Embora isso dê a sugestão de que essa interpretação leve muito tempo para ficar pronta, o compilador também caminha bastante para ser convertido.

Alguns dos passos são: Análise léxica e semântica, pré-processamento, análise e otimização de código e, ao fim, geração do produto final.

Porém, uma vez compilado, ele não precisará fazer mais essa tradução – a menos que o código precise ser alterado.

## Prós e contras de interpretadores e compiladores

Uma das grandes vantagens dos compiladores é sua velocidade de execução, muito em função do que já falamos sobre traduzir todo o código de uma vez. Não precisar fazer a conversão toda vez que o sistema é executado dá uma eficiência muito maior do que um interpretador.

Uma compilação costuma dar resultados mais confiáveis graças às suas diversas etapas de validação e otimização. Uma checagem de tipos estáticos, por exemplo, é comum em compiladores, e identifica diversos erros de programação antes do executável ser gerado.

Por sua vez, enquanto uma linguagem compilada precisa fazer essa tradução para cada plataforma destinada (como versões específicas do Adobe Photoshop para Windows, Linux e Mac), a interpretação, por poder rodar em tempo de execução, é independente.

Não importa se é Linux ou Mac; basta ter o Python instalado na máquina que ela vai rodar seu código em Python.

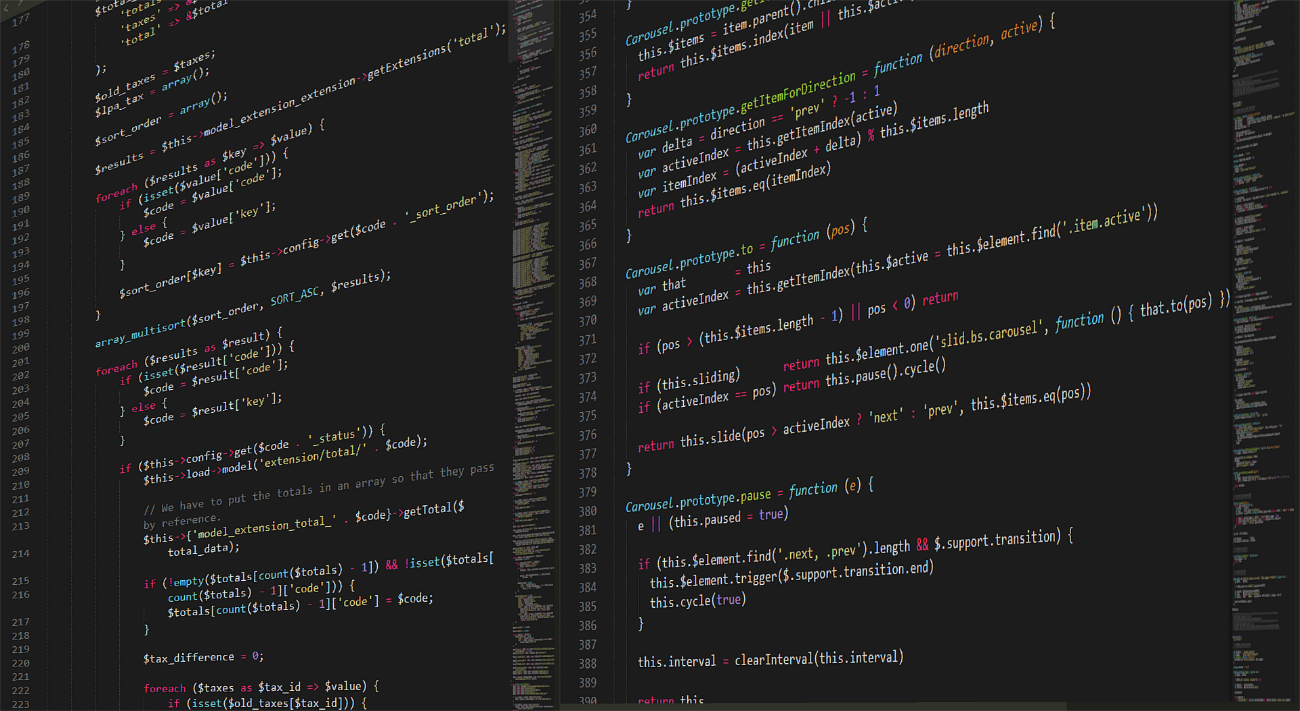
Além disso, a capacidade de execução em ***runtime*** permite utilizar [**reflexão**](http://thesocietea.org/2016/02/programming-concepts-type-introspection-and-reflection/) (ou seja, examinar e modificar sua própria estrutura em tempo de execução) e tipagem dinâmica (capacidade de escolher dinamicamente o tipo de uma variável, não exigindo uma declaração), uma característica básica do PHP, por exemplo.

Verificar e modificar o código de uma linguagem interpretada também é mais fácil, já que basta abrir o arquivo e ver o que tem escrito. Para fazer o mesmo com uma biblioteca compilada, é preciso utilizar um descompilador (o [Java Decompiler](http://jd.benow.ca/) já salvou minha vida quando precisei abrir uma biblioteca de um projeto herdado).

Isso, no entanto, também pode ser visto como uma desvantagem, pois qualquer pessoa com um mínimo de conhecimento pode ver a implementação de um JavaScript embutido numa página web ou até mesmo realizar uma injeção de código.

Por isso mesmo, a segurança é sempre uma preocupação em momentos assim.

## E quanto aos scripts?

[](https://imasters.com.br/devsecops/aplicar-script/)

Essa preocupação com a segurança e com a flexibilidade das aplicações, entre outros motivos, foram responsáveis a dar vida para aqueles pequenos trechos de código (ou scripts) baseados em Java, transformando-os em uma linguagem propriamente dita ([linguagem de script](http://www.tecmundo.com.br/programacao/1185-o-que-e-script-.htm)) como o JavaScript.

Por outro lado, sua principal característica ainda é a interpretação do código pelo lado do cliente, através do navegador (sim, uma das funções básicas de um browser é interpretação), sem que precise ser executado no servidor.

Mesmo assim, isso não impede que um script seja compilado. Na verdade, os games fazem uso disso com certa frequência, seja para adicionar novas funcionalidades ou corrigir problemas com a aplicação de atualizações.

Lua também é uma linguagem de script extremamente poderosa, bem famosa pela construção de *macros* e *add-ons* compilados por [jogadores de World of Warcraft](http://books.google.com.br/books?id=rAmNCqpmunIC&dq=compiling+addon+lua&hl=pt-BR). Porém, como ela gera bytecodes como resultado, além de ter tipagem dinâmica e gerenciamento automático de memória, é mais comum sua associação com interpretadores.

## Compiladores + Interpretadores

Por terem características diferentes (e, de certa forma, complementares), algumas linguagens fazem uso dos dois conceitos, sendo o Java o maior exemplo disso.

Implementações do Java costumam compilar o código a fim de gerar um [bytecode](http://www.javaworld.com/article/2077233/core-java/bytecode-basics.html" \t "_blank), ao invés de instruções em linguagem de máquina. A partir dessa conversão, o bytecode pode ser interpretado por uma JVM (*Java Virtual Machine*), que pode ser instalada em qualquer máquina. Assim, o Java combina a confiabilidade e a otimização da compilação com a flexibilidade da interpretação.

Uma evolução desse modelo está no conceito de compilação *just-in-time* (JIT). O JIT nada mais é que uma compilação feita em tempo de execução, ao invés de antes de rodar a aplicação (*ahead-of-time*). Ela pode ser feita por arquivo, função ou até fragmentos de código, traduzindo dinamicamente essas partes e executando diretamente na memória.

Essa técnica ainda apresenta algumas desvantagens como o atraso na inicialização (“*startup delay*”), pois ainda é necessário carregar os primeiros blocos do código para serem compilados. Assim, quanto mais o JIT for otimizado, melhor o código gerado, mas também esse atraso fica maior.

Uma boa dica para conhecer mais sobre o assunto está no artigo “[Compilação Just-In-Time: Histórico, Arquitetura, Princípios e Sistemas](https://pdfs.semanticscholar.org/914d/b8a17227605384dff1133888ed843915c463.pdf)“, escrito pelos então estudantes George Souza Oliveira e Anderson Faustino da Silva, da Universidade Federal do Maringá.

## Quando usar cada uma?

Como dito acima, as linguagens podem ser tanto interpretadas quanto compiladas, então vai da necessidade da aplicação o método de execução, independentemente da linguagem escolhida.

**Sistemas web frequentemente usam linguagens interpretadas** para serem feitos pela facilidade de manutenção e pela necessidade de portabilidade. Um problema que até recentemente afetava bastante essas páginas era a falta de padronização do mercado: um site que era bem visualizado no Google Chrome podia ficar com um layout esquisito no Internet Explorer.

Muitas vezes, os programadores tinham que recorrer a scripts, alterações de layout ou de código diversas vezes, então imagina só ter que lidar com diversas distribuições de compilação. Ao contrário, realizar as modificações necessárias e vê-las aplicadas pouco tempo depois é bem mais interessante para desenvolvedores e usuários.

**Em contrapartida, aplicações compiladas são bem mais seguras e rápidas, e são recomendadas para quem exige um desempenho considerável.** Mas também traz alguns incômodos, como ter que reinstalar o sistema quando uma versão mais atualizada fica disponível.

Além disso, várias das linguagens mais aceitas atualmente utilizam podem gerar bytecode como formato intermediário – como Java, C#, Python e Ruby – sem problemas. Assim, é possível ficar em um meio-termo, garantindo as melhores vantagens de cada um para seu sistema.

\*\*\*

Artigo original publicado na *[GeekHunter](http://geekhunter.com.br/)* e republicado com a autorização da autora: [*Métodos de Tradução: interpretador x compilador*](http://blog.geekhunter.com.br/metodos-de-traducao-compiladores-ou-interpretadores/)

Compiladores e interpretadores são formas de tradução de um código geralmente de alto nível para um código que o computador (a máquina) interprete. Este tradutor tem a função de traduzir uma linguagem abstrata para uma linguagem binária.

**Compilador:**

Compilador é um programa ou um grupo de programas que escrito por uma linguagem (esta sendo necessária de compilação para sua execução) ao ser compilado gera outro código que é interpretado pelo computador. Este código compilado é chamado de código objeto, podendo ser um arquivo executável que é reproduzido em um sistema operacional, por exemplo. Este tipo de tradutor é um dos mais utilizados.  
  
Os compiladores analisam o código em três partes, de forma sintática ou hierárquica, análise léxica ou linear e anâlise semântica.  
  
Alguns compiladores contam com um pré-processamento. Este pré-processamento é responsável por modificar o código-fonte conforme necessidades que o compilador identifique que sejam necessárias, por exemplo, otimização de código.  
  
**Vantagens:**

* O código compilado é mais rápido de ser acessado;
* Impossibilita ou pelo menos dificulta ser quebrado e visualizado o código-fonte original;
* Permite otimização do código por parte do compilador;
* Compila o código somente se estiver sem algum erro.

**Desvantagens:**

* Para ser utilizado o código precisa passar por muitos níveis de compilação;
* Assim como vantagem a possibilidade de não poder visualizar o código-fonte, pode ser uma desvantagem;
* Processo de correção ou alteração do código requer que ele seja novamente recompilado.

**Interpretador:**

O interpretador ao contrário do compilador roda o código-fonte escrito como sendo o código objeto, ele traduz o programa linha a linha, o programa vai sendo utilizado na medida em que vai sendo traduzido. Cada execução do programa precisa ser novamente traduzido e interpretado.  
  
O interpretador analisa sintaticamente e semanticamente o código, se estas duas etapas forem realizadas e executadas de forma correta o código está pronto para funcionar.    
  
**Vantagens:**

* Correções e alterações são mais rápidas de serem realizadas;
* Código não precisa ser compilado para ser executado;
* Consomem menos memória.

**Desvantagens:**

* Execução é mais lenta do programa;
* Necessita sempre ser lido o código original para ser executado;