

**UNIVERSIDADE DE ITAÚNA**

DAVI VENTURA CARDOSO PERDIGÃO  
ERIC HENRIQUE DE CASTRO CHAVES

**Trabalho Prático - Linguagens de Programação**  
Introdução ao WebAssembly

**ITAÚNA**  
**2022**

## SUMÁRIO

1. INTRODUÇÃO.....	2
2. OBJETIVO.....	2
3. COMO APLICAR?.....	2
4. MITOS SOBRE WEBASSEMBLY.....	5
5. CONCLUSÃO.....	6
6. REFERÊNCIAS BIBLIOGRÁFICAS.....	6

## 1. INTRODUÇÃO

WebAssembly é uma tecnologia emergente, mas que terá um impacto enorme na Web em um futuro próximo. Agora apoiado por todos os principais navegadores da Web, bem como Node, WebAssembly permitirá uma diversidade de linguagens na Web, e não apenas aquelas que podem transpilar para JavaScript.

Linguagens de baixo nível como C e Rust podem compilar para WebAssembly, que é um formato binário para arquivos de tamanhos menores e tempo de execução mais rápido. O desempenho próximo do nível nativo pode ser obtido com o código compilado do WebAssembly, muitas vezes muito mais rápido que o JavaScript equivalente. É provável que vejamos bibliotecas JavaScript começando a aproveitar WebAssembly para sessões de desempenho crítico de código em um futuro próximo.

## 2. OBJETIVO

- **Ser rápido, eficiente e móvel:** o código WebAssembly pode ser executado a velocidades próximas de nativas entre diferentes plataformas, tirando vantagem das capacidades comuns de hardware;
- **Ser compreensível e debuggable:** WebAssembly é uma linguagem assembly de baixo nível, mas ela tem um formato de texto compreensível para os humanos (especificação pela qual ainda está sendo finalizado) que permite que o código seja escrito, visto e debugado à mão;
- **Manter a segurança:** WebAssembly é especificado para ser executado num ambiente seguro e controlado. Como outros códigos web, ele reforçará as mesmas políticas de origem e permissões dos browsers;
- **Não quebrar a web:** WebAssembly foi pensado de maneira que ele seja executado em harmonia com outras tecnologias web, mantendo a compatibilidade retroativa.

## 3. COMO APLICAR?

Parcel torna **extremamente fácil** começar com WebAssembly. Supondo que tenhamos um arquivo `.wasm`, podemos apenas importar como de costume. As importações síncronas e assíncronas são suportadas.

```
// importação síncrona
import { add } from './add.wasm'
console.log(add(2, 3))

// importação assíncrona
const { add } = await import('./add.wasm')
console.log(add(2, 3))
```

Ao importar sincronicamente um arquivo `.wasm`, o Parcel gera automaticamente código extra para pré-carregar o arquivo antes de executar seu pacote JavaScript. Isso significa que o arquivo WebAssembly binário não é embutido em seu JavaScript como uma sequência de caracteres. Desta forma, o seu código ainda funciona sincronicamente, mas Parcel cuida de carregar dependências para você.

Tudo isso é habilitado pelo suporte interno do Parcel para carregadores de pacotes, que são módulos de tempo de execução que sabem como carregar um formato de arquivo específico de forma assíncrona. Em versões anteriores, haviam carregadores de pacotes codificados para JavaScript e CSS, que habilitaram o suporte dinâmico de importação. No Parcel v1.5.0, isso é **completamente plugável**, sendo possível definir seus próprios carregadores de pacotes em plugins. Isso permitirá diversas funcionalidades no futuro para formatos binários personalizados.

### **A plataforma web pode ser dividida em duas partes:**

- Uma máquina virtual (VM) que executa o código de uma aplicação Web, por exemplo códigos JavaScript que enriquecem suas aplicações.
- Um conjunto de Web APIs que um Web app pode invocar para controlar funcionalidades web browser/device (dispositivo) e fazer as coisas acontecerem (DOM, CSSOM, WebGL, IndexedDB, Web Audio API, etc.).

Historicamente, a VM tem tido permissão para carregar apenas JavaScript. Isto tem funcionado bem para nós, já que o JavaScript é poderoso o suficiente para resolver a maioria dos problemas da Web atualmente. No entanto, temos enfrentado problemas de performance, quando tentamos usar o JavaScript para tarefas mais intensivas como games em 3D, realidades virtual e aumentada, visão de computador, edição de imagens ou vídeos e um sem número de outros domínios que demandam performance nativa. Adicionalmente, o custo de baixar, parsear e compilar aplicações JavaScript muito grandes, é proibitivo. Plataformas mobile e outras de recursos restritos, podem ampliar ainda mais estes gargalos de performance.

WebAssembly é uma linguagem diferente do JavaScript, mas não foi pensada para ser sua substituta. Ao contrário, foi pensada para complementar e trabalhar lado a lado com o JavaScript, permitindo aos desenvolvedores web tirarem vantagem dos pontos fortes das duas linguagens:

- JavaScript é uma linguagem de alto nível, flexível e expressiva o suficiente para escrever aplicações web. Ela tem muitas vantagens — é dinamicamente tipada, não necessita ser compilada e tem um enorme ecossistema que disponibiliza poderosos frameworks, bibliotecas (libs) e outros recursos.
- WebAssembly é uma linguagem de baixo nível do tipo assembly com um formato binário compacto, que é executado com performance próximo à nativa, que disponibiliza linguagens com modelos de memória de baixo nível como C++ e Rust, com uma compilação-alvo, assim podendo ser executados na web. (Note que o WebAssembly tem uma meta futura de suporte de alto nível para linguagens com modelos de memória garbage-collected.)

Com o advento do WebAssembly nos browsers, a máquina virtual a que nos referimos antes, vai carregar e executar dois tipos de código (JavaScript e WebAssembly).

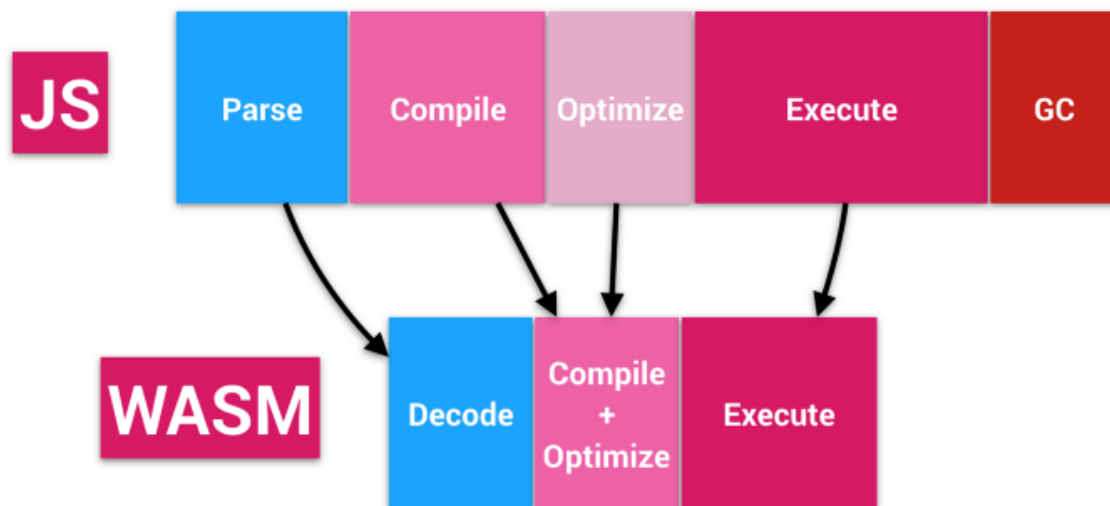
Os diferentes tipos de códigos podem invocar um ao outro conforme necessário — o WebAssembly JavaScript API encapsula código WebAssembly exportado com funções JavaScript que podem ser invocados normalmente, e código WebAssembly pode importar de forma síncrona, funções normais de JavaScript. Na verdade, a unidade básica do código WebAssembly é chamado de módulo, e os módulos WebAssembly são semelhantes em vários níveis aos módulos de ES2015.

### **Conceitos-chave do WebAssembly:**

- **Módulo:** Representa o binário do WebAssembly que foi compilado pelo browser, em código executável pela máquina. Um módulo não tem estado e, tal qual um Blob, pode ser explicitamente compartilhado entre janelas e workers (via `postMessage()`). Um Módulo declara imports e exports, assim como um módulo ES2015.
- **Memória:** Um `ArrayBuffer` redimensionável que contém um array linear de bytes, lidos e escritos pelas instruções de memória de baixo nível do WebAssembly.
- **Tabela:** Um array tipado de referências redimensionável (por exemplo para funções) que, em outra situação, não poderia ser armazenado como bytes puros na Memória (por questões de segurança e portabilidade).
- **Instância:** Um Módulo pareado com todo o estado utilizado durante a execução, incluindo uma Memória, Tabela e um conjunto de valores importados. Uma Instância é como um módulo ES2015 que foi carregado em um global específico com um conjunto de importações específico.

## 4. MITOS SOBRE WEBASSEMBLY

- **WASM vai matar o JavaScript:** Não é esperado que o WebAssembly seja usado no lugar do JavaScript, ele foi criado para complementá-lo onde a performance da aplicação web é crítica.



Comparação teórica da execução de um código JavaScript e WASM.

Fonte: Medium, 2018.

- **WASM é uma nova linguagem de programação:** Lembre-se, WASM é um formato binário, intermediário, que serve como *Compiler Target* para linguagens de programação como C, C++ e Rust. Embora exista um formato de texto para representação do seu código, não é esperado que se escreva código nesta representação.
- **Só sendo programador C ou Rust para programar para WASM:** O suporte para outras linguagens de programação virá assim que novas funcionalidades forem adicionadas ao WebAssembly, como suporte ao *Garbage Collector*. Além disso, existem projetos de novas linguagens e até sub/superset do JavaScript que compila para WASM.

## 5. CONCLUSÃO

Como vimos, os desenvolvedores em algum momento da sua carreira serão afetados pelo WebAssembly. Isso porque essa abordagem revolucionária de formato binário para a web veio para ficar. Contudo, podemos concluir que o WebAssembly é definitivamente uma melhoria para o que temos hoje, que é o desenvolvimento em JavaScript. Além disso, é um grande avanço também para o navegador.

Basicamente, podemos esperar o início de uma nova era com menos códigos, melhor desempenho e eliminar bugs. As tecnologias oferecem muitas possibilidades, o WebAssembly não é exceção e pode ser um aliado ou um inimigo. É muito claro que tem muitas vantagens, mas pode fornecer novos caminhos para explorar as fraquezas em diferentes casos. Enquanto os desenvolvedores se esforçam para integrar recursos de segurança, os usuários devem tomar alguns cuidados, mantendo os navegadores atualizados com plug-ins que bloqueiam a execução dinâmica de JavaScript, como o NoScript.

## 6. REFERÊNCIAS BIBLIOGRÁFICAS

- What is WebAssembly (em inglês): <https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts>.
- AST Binário, link da explicação da proposta (em inglês): <https://github.com/binast/ecmascript-binary-ast>.
- Javascript startup performance (em inglês): <https://medium.com/reloading/javascript-start-up-performance-69200f43b201>.
- Turboscript - um superset do JavaScript que compila para WASM: <https://github.com/01alchemist/TurboScript>.
- Walt - sintaxe JavaScript-like para WebAssembly: <https://github.com/ballercat/walt>
- WebAssembly - MDN Web Docs - Mozilla : <https://developer.mozilla.org/pt-BR/docs/WebAssembly>