



Bundlers e Compilers

Introdução

Compiladores

Entendendo o que são compiladores e como eles trabalham

Transpilador

Linguagem de saída do compilador JavaScript

Exemplos de compiladores JavaScript

JavaScript e as técnicas de compilação JIT

Babel

Babel, JSX e React

Vantagens do Uso de Compiladores:

Bundlers / Empacotadores

Módulos no JavaScript

CommomJS (CJS)

ECMAScript Modules (ESM)

CommomJS (CJS) e ECMAScript Modules (ESM)

O processo de bundling (empacotamento)

O que são bundlers (empacotadores)?

Quem realiza as otimizações de código: o bundler ou o compilador?

webpack

Entry Point (Ponto de Entrada)

Output (Saída)

Loaders (Carregadores)

Plugins

Mode (Modo)

Suporte à navegadores

esbuild

Instalação do esbuild

Por que o eslint é rápido?

Referências

Introdução

- Nem sempre os navegadores suportarão a versão mais atualizada de JavaScript que estamos utilizando;

- Diversas ferramentas foram criadas para nos permitir converter nosso código que é escrito em JavaScript mais moderno, para versões JavaScript que rodam em ambientes específicos;
 - Por exemplo, o código que você vai criar para sua aplicação nem sempre poderá rodar na versão do Internet Explorer 10.
- Não importa quão atualizados estejam os navegadores, na maioria de nossas aplicações precisaremos usar um conceito que são empacotadores e compiladores.

Compiladores

Entendendo o que são compiladores e como eles trabalham

- Um **compilador** é uma ferramenta que traduz código de uma linguagem para outra;
 - Você pode pensar em um compilador JavaScript como um tradutor de idiomas;
 - Pedir um sushi: inglês → japonês.
- O compilador pega o código JavaScript, que é escrito em uma linguagem de alto nível que é fácil para os humanos lerem e escreverem, e o converte em uma linguagem de baixo nível que é fácil para o computador entender e executar;
- No contexto do JavaScript, os compiladores geralmente traduzem código moderno (como ES6+) para versões mais antigas (como ES5) que são compatíveis com navegadores mais antigos;

Transpilador

- **Compilador** e **transpilador** são ambos tipos de ferramentas que processam código, mas eles têm diferenças importantes:
 1. **Compilador:**
 - Um compilador é uma ferramenta que **converte código-fonte de uma linguagem de alto nível para um código de baixo nível** (como código de máquina);
 - Ele produz um **artefato diretamente executável**, como um arquivo binário ou bytecode;

- Exemplos de linguagens compiladas incluem **C** (que gera código de máquina) e **Java** (que gera bytecode).

2. Transpilador:

- Um transpilador, também conhecido como **compilador de código-fonte para código-fonte**, converte código-fonte de uma linguagem para outra linguagem de **mesmo nível de abstração**;
- Ele não gera um executável diretamente, mas sim outro código-fonte em uma linguagem diferente;

-
- Vamos supor que queremos executar o código React dentro do navegador, e o navegador ainda não suporta alguns recursos do JavaScript;
 - Então precisamos converter esse código de uma versão mais atualizada do JavaScript para uma versão que o navegador entenda.
 - **Compiladores** são ferramentas essenciais no desenvolvimento JavaScript por várias razões:
 - **Compatibilidade com Navegadores:** Os navegadores têm suporte variado para diferentes versões do JavaScript;
 - O compilador pode aumentar a portabilidade e compatibilidade do seu código, permitindo que ele seja executado em diferentes navegadores ou ambientes que podem não oferecer suporte à versão original do JavaScript.
 - **Recursos Modernos:** O JavaScript evoluiu ao longo dos anos, adicionando recursos poderosos, como arrow functions, classes, destructuring e módulos;
 - Compiladores nos permitem usar esses recursos mesmo em ambientes que não os suportam nativamente.
 - **Otimização e Minificação:** Compiladores também otimizam o código, removendo espaços em branco, renomeando variáveis e reduzindo o tamanho dos arquivos;
 - A legibilidade e a manutenibilidade do seu código, permitindo que você use sintaxe e recursos mais recentes ou personalizados que tornam o seu código mais expressivo e claro;
 - Isso melhora o desempenho da aplicação nos ambientes de produção.

- **Preparação para Produção:** Antes de implantar um aplicativo em produção, é comum compilar o código para garantir que ele esteja otimizado e pronto para uso por parte dos nossos usuários.
 - Melhora o desempenho e a eficiência do seu código, o otimizando para a plataforma de destino e eliminando partes desnecessárias ou redundantes.

Linguagem de saída do compilador JavaScript

- Não existe uma única linguagem para a qual todos os compiladores mudam seu código;
- Diferentes compiladores podem transformar seu código em diferentes linguagens, dependendo do que você quer fazer e onde deseja executá-lo;
- Alguns compiladores podem transformar seu código em uma linguagem que usa apenas 0s e 1s, que o computador pode ler muito rapidamente;
 - Isso pode fazer com que seu código seja executado mais rapidamente e use menos recursos;
 - Um compilador que faz isso é o V8, que é usado pelo Google Chrome e Node.js.
- Outros compiladores podem converter seu código em uma linguagem que pode ser modificada novamente ou lida por outra ferramenta;
 - Faz com que o código funcione em diferentes navegadores ou dispositivos que podem não suportar a versão mais recente do JavaScript;
 - Exemplo Babel: converte o código em uma versão mais antiga do JavaScript para que os navegadores antigos entendam.
- Existem compiladores que transformam seu código novamente em JavaScript, mas com diferentes maneiras de escrever ou fazer as coisas;
 - Código mais fácil de ler e escrever;
 - Permite o uso de novas ou personalizadas formas de escrever ou fazer as coisas;
 - Exemplo: TypeScript, que pode ser convertido em código JavaScript simples que funciona em qualquer navegador.
- O compilador JavaScript opera em estágios.
 - Inicialmente, ele interpreta o código JavaScript para executá-lo rapidamente;

- À medida que o código é executado repetidamente ou conforme certas regras, o compilador identifica partes do código que são frequentemente executadas ou que exigem otimização;
- Essas partes são então compiladas em um código de máquina altamente otimizado, o que melhora significativamente o desempenho da execução do programa.

Exemplos de compiladores JavaScript

- O **Babel** é um compilador amplamente utilizado no ecossistema JavaScript;
 - Ele permite que você escreva código moderno e o compila para uma versão compatível com navegadores mais antigos;
 - Isso é especialmente útil quando você está usando recursos avançados do ES6+.
- O TypeScript pode ser compilado em código JavaScript simples que pode ser executado em qualquer navegador.
 - Superset do JavaScript que adiciona tipos opcionais e outras funcionalidades.
 - Um "superset" é um termo usado em linguagens de programação para descrever uma linguagem que contém todos os recursos da linguagem base (subset), além de oferecer funcionalidades adicionais.

JavaScript e as técnicas de compilação JIT

- JavaScript é uma linguagem de programação que é, em sua maioria, **interpretada**;
 - Diferentemente das linguagens compiladas, onde o código fonte é traduzido para uma representação de código de máquina antes da execução, as linguagens interpretadas são traduzidas e executadas linha por linha em tempo real.
- Em uma linguagem interpretada, o código fonte não é traduzido diretamente pela máquina de destino;
 - Em vez disso, um programa diferente, o interpretador, lê e executa o código;
 - Portanto, os interpretadores passam por um programa linha por linha e executam cada comando.

- Um compilador JIT é responsável por traduzir o código fonte do programa em código de máquina executável;
 - Ao invés de compilar todo o código de uma vez, o compilador JIT utiliza informações sobre o contexto de execução para decidir quais partes do código devem ser compiladas e executadas em tempo real;
 - Essa abordagem permite que o compilador JIT identifique partes do código que são executadas com frequência e otimize-as para um desempenho máximo;
 - Modernas implementações de JavaScript, como o V8 do Google Chrome, usam técnicas de compilação Just-In-Time (JIT) para melhorar o desempenho.
- Portanto, embora o JavaScript seja geralmente classificado como uma linguagem interpretada, na prática, ele utiliza tanto a interpretação quanto a compilação JIT.
- Essa combinação de interpretação e compilação dinâmica permite que o JavaScript equilibre entre:
 - Tempos de inicialização rápidos e velocidade de execução otimizada;
 - Fornecendo um ambiente de tempo de execução flexível e eficiente para aplicações web.

Babel

- Babel é um dos compiladores/transpiladores JavaScript mais populares;
- Babel é uma cadeia de ferramentas usada principalmente para converter código ECMAScript 2015+ em uma versão compatível com versões anteriores do JavaScript em navegadores ou ambientes atuais e mais antigos;
- Babel é um transpilador, que é um tipo especial de compilador, então ambos os termos estão tecnicamente corretos;
- Principais recursos do Babel:
 1. **Transformar Sintaxe:**
 - O Babel é capaz de transformar a sintaxe de código JavaScript de uma versão mais recente para uma versão mais antiga;
 - Recursos como arrow functions, classes, template literals e outras características introduzidas em versões mais recentes do

ECMAScript podem ser convertidos para uma forma que seja compreendida por navegadores.

2. Preencher Recursos Ausentes:

- Em alguns casos, certos recursos do JavaScript podem não estar disponíveis em todos os navegadores ou ambientes de execução;
- O Babel pode preencher essas lacunas por meio de polyfills de terceiros, como o core-js.
 - *Polyfills* são pedaços de código que implementam funcionalidades JavaScript ausentes em navegadores antigos.

3. Transformações de Código-Fonte (Codemods):

- O Babel pode realizar transformações mais avançadas no código-fonte, não apenas alterando a sintaxe, mas também reestruturando ou refatorando o código de maneira significativa;
- Essas transformações podem ser realizadas automaticamente com o uso de codemods, que são scripts ou plugins que automatizam mudanças em larga escala no código-fonte, como renomear variáveis, extrair funções, entre outras tarefas.

4. E Muito Mais:

- Babel oferece uma gama de outras capacidades, incluindo:
 - Suporte a diferentes formatos de módulos;
 - Integração com ferramentas de construção (como webpack);
 - Otimizações de código e muito mais.
 - Uma das principais vantagens do Babel é seu extenso ecossistema de plugins;
 - Os desenvolvedores podem personalizar o comportamento do Babel adicionando ou configurando plugins;
 - Permite a adaptação do processo de compilação às suas necessidades específicas.

```
// Babel Input: ES2015 arrow function
[1, 2, 3].map(n => n + 1);

// Babel Output: ES5 equivalent
```

```
[1, 2, 3].map(function(n) {  
  return n + 1;  
});
```

Babel · Babel

The compiler for next generation JavaScript

 <https://babeljs.io/>

Babel, JSX e React

- **Presets do Babel:** são conjuntos predefinidos de plugins que configuram o Babel para lidar com determinados tipos de transformações de código;
 - Eles são usados para simplificar o processo de configuração do Babel;
 - Agrupa conjuntos comuns de plugins que são frequentemente usados juntos para atender a necessidades específicas de desenvolvimento;
 - Esses presets ajudam os desenvolvedores a evitar a necessidade de configurar manualmente cada plugin individualmente, oferecendo uma configuração pronta para uso que é adaptada para cenários comuns de desenvolvimento.
- O preset `@babel/preset-react` é um conjunto de plugins específicos que permitem ao Babel compilar código JSX para JavaScript puro, tornando-o executável nos navegadores.

Instalando o preset-react

```
npm install --save-dev @babel/preset-react
```

```
yarn add --dev @babel/preset-react
```

Vendo o funcionamento do Babel na prática.

Vantagens do Uso de Compiladores:

1. Melhoria de Desempenho
2. Portabilidade
3. Otimização de Código
4. Segurança Aprimorada
5. Independência de Linguagem
6. Verificação de Erros
7. Executáveis Autônomos
8. Melhor Utilização de Recursos

Bundlers / Empacotadores

Módulos no JavaScript

- Quando as aplicações do JavaScript começaram, eles eram muito pequenos, onde a maior parte do que se aplicava o JS era para scripts isolados, ou para fornecer o mínimo de interatividade com as páginas web;
- Isso fazia com que grandes scripts não fossem necessários de se utilizar;
- Ao passar dos anos, com o JavaScript sendo utilizado em vários contextos diferentes, o JavaScript começou a ser utilizado em sistemas completos, trazendo scripts muito maiores, mais “pesados” e complexos;
- Com isso, surgiu a necessidade de dividir as aplicações feitas em JavaScript em módulos separados, que podem ser importados quando necessário;
 - É como se nossas aplicações se dividissem em peças que, quando se juntam, criam algo maior.
- Cada módulo tem um escopo e responsabilidade menor do que um programa completo, tornando a verificação, depuração e teste trivial;
 - Propósito claro dentro da aplicação como um todo.
- O Node.js tem suportado a programação modular quase desde o seu início;
- Demorou, mas hoje em dia, a maioria dos navegadores modernos começaram a dar suporte nativamente para essa funcionalidade de módulos;

- Torna o código mais eficiente do que ter que usar uma biblioteca e fazer todo esse processamento do lado do cliente de maneira menos performática.

CommomJS (CJS)

- **CommonJS (CJS)** é um projeto que visa padronizar o ecossistema de módulos para **JavaScript** fora dos navegadores da web, como em servidores ou aplicativos nativos de desktop;
- Os módulos CommomJS são a forma original de empacotar o código JavaScript para o Node.js;
- Ele usa a sintaxe `require()` para importar módulos e `module.exports` para exportar funcionalidades;

ECMAScript Modules (ESM)

- Sistema padrão de módulos do JavaScript/ECMAScript;
 - O termo "ECMAScript" se refere à especificação oficial do JS.
- Representa uma abordagem para organizar e modularizar código nos projetos;
- Mais comumente utilizado em ambientes de navegador modernos e em algumas ferramentas de desenvolvimento front-end
- Deve substituir o CommomJS (CJS);
- O CommonJS usa `require()` e `module.exports`, enquanto os módulos ES (ou ES6 modules) usam `import` e `export`.

```
// add.js
export function add(a, b) {
  return a + b;
}

// main.js
import { add } from "./add.js";
```

- O ESM foi padronizado em 2015, mas levou tempo para ser amplamente suportado pelos navegadores e pelo Node.js;
- Traz vantagens como:
 - Importação e exportações mais claras;
 - Suporte nativo para importações assíncronas.
- No React, permite que realize a separação dos componentes da nossa aplicação em diferentes módulos de forma que:
 - Auxilia na organização do código;
 - Fornece importações e exportações claras;
 - Fornece suporte nativo para importações assíncronas;
 - Os módulos ESM suportam importações assíncronas para realizar o carregamento dinamicamente das partes do código em aplicações React;
 - Ajuda na otimização do desempenho.
 - Tem boa integração com ferramentas como:
 - TypeScript;
 - Babel;
 - ESLint.

CommonJS (CJS) e ECMAScript Modules (ESM)

- Para navegadores mais antigos que não suportam ESM, o CommonJS ainda pode ser útil;
- Os navegadores modernos oferecem suporte nativo aos ESM;
- O ESM são preferíveis porque têm uma sintaxe mais clara e suportam importações assíncronas;
- Se você precisa dar suporte a navegadores mais antigos (como o Internet Explorer), pode usar ferramentas como o Babel, para converter código CommonJS em módulos ES.

O processo de bundling (empacotamento)

- Nós, desenvolvedores, quebramos nossas aplicações em **módulos**, **componentes** e **funções** que podem ser usados para construir pedaços da nossa aplicação, como se fosse um lego;
- Exportando e importando esses módulos, criamos uma dependência entre esses arquivos;
- **Bundling** (ou agrupamento), é um processo que resolve as dependências desses arquivos e junta esses módulos dentro de pacotes para o navegador, para poder **reduzir o número de requisições por arquivo** quando um usuário abre a página;
- Ao invés do navegador realizar a requisição de 6 arquivos separadamente quando um usuário acessar a nossa página, ele irá realizar a consulta de apenas um arquivo JavaScript;
 - Em um arquivo apenas, nós temos todo o código do nosso sistema.
- Porém, conforme nossa aplicação cresce, o nosso *bundle* (pacote) também cresce;
 - Isso pode acontecer quando instalamos mais libs de terceiros, ou simplesmente quando estamos adicionando mais funcionalidades ao nosso sistema.
- Quanto maior o nosso *bundle* (pacote), mais tempo o nosso site ou aplicação demora pra ser carregado para que o nosso usuário possa ver nossa página e interagir.

-
- Na prática é como se nós possuíssemos dois arquivos de código JavaScript:

```
import { add } from './math.js';  
  
console.log(add(16, 26)); // 42
```

```
export function add(a, b) {  
  return a + b;  
}
```

- Após o processo de empacotamento, nós teremos o seguinte resultado:

```
function add(a, b) {  
  return a + b;  
}  
  
console.log(add(16, 26)); // 42
```

O que são bundlers (empacotadores)?

- Ferramentas que ajudam a gerenciar e organizar o código JavaScript, permitindo que os desenvolvedores modularizem seus projetos e os distribuam de maneira mais eficiente;
- Essas ferramentas são especialmente úteis em projetos grandes e complexos, nos quais os desenvolvedores escrevem código em vários arquivos diferentes (módulos);

Características que tornam os bundlers necessários:

1. Complexidade do Desenvolvimento:

- Imagine escrever um site complexo usando apenas esses três componentes;
 - Idealmente, dividiríamos nosso código JS em diferentes arquivos e os incluiríamos no HTML;
 - No entanto, isso resultaria em várias solicitações HTTP separadas para carregar todos os arquivos.
 - Para evitar esse problema, podemos combinar todos os arquivos em um único arquivo, mas isso se torna complexo de gerenciar manualmente.

2. Dependências e Bibliotecas:

- Projetos modernos frequentemente dependem de bibliotecas externas, como o caso da própria biblioteca React;
- Essas bibliotecas podem, por sua vez, depender de outras bibliotecas.
 - Gerenciar todas essas dependências manualmente durante a construção do projeto é trabalhoso.

3. Otimização de Carregamento:

- Se mesclarmos todos os arquivos manualmente, podemos criar um arquivo JS enorme, mas isso aumenta o tempo de carregamento do site;
- Um **bundler** resolve esse problema, otimizando o código e criando um único arquivo que contém tudo o que precisamos;
- Os bundlers podem aplicar otimizações:
 - Removendo trechos de código que não são utilizados;
 - Minificando o código (removendo espaços em branco e renomeando variáveis para reduzir o tamanho do arquivo);
 - Realizando outras otimizações para melhorar o desempenho do aplicativo.

4. Gestão de Recursos:

- Além do código JavaScript, os bundlers também gerenciam recursos estáticos, como estilos CSS, imagens e fontes;
- Garante que sejam incluídos e otimizados no pacote de saída.

Quem realiza as otimizações de código: o bundler ou o compilador?

- A otimização geralmente é realizada pelo próprio bundler, embora o compilador também possa desempenhar um papel importante, dependendo do contexto específico do projeto e das ferramentas utilizadas.
- **Bundler:** possuem recursos embutidos para otimização de código;
 - Exemplo: Webpack, um dos bundlers mais populares para projetos JavaScript;
 - Inclui recursos para minificação de código;
 - Remoção de partes do código que não são usadas;
 - Divisão de código assíncrono (dividindo o código em pedaços menores para carregamento mais eficiente);
 - Etc.
 - Essas otimizações são realizadas durante o processo de empacotamento do código.
- **Compilador:** o compilador também pode desempenhar um papel na otimização do código.

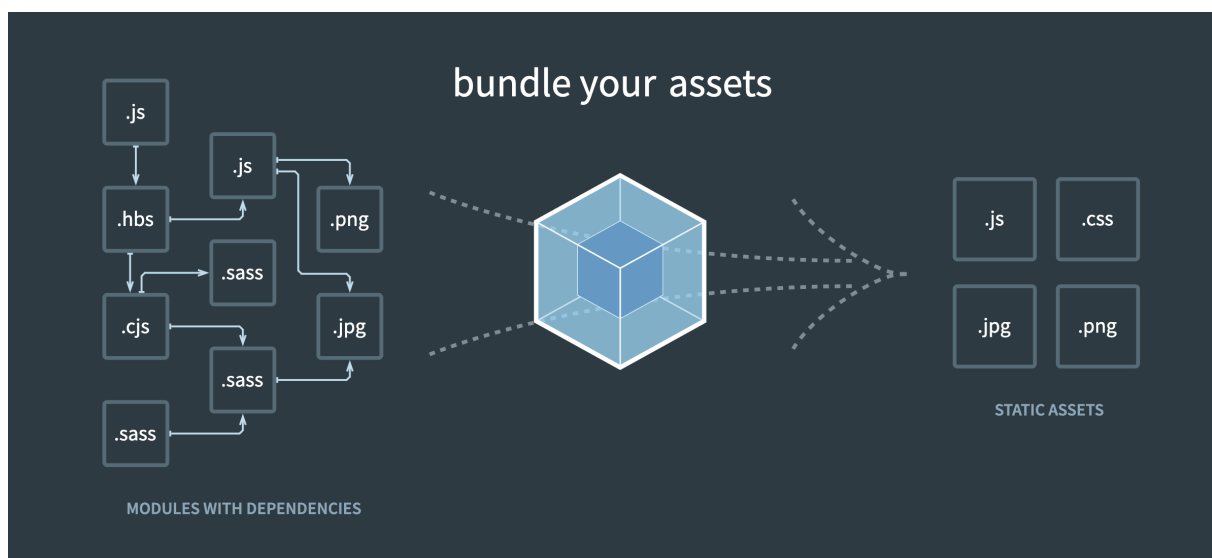
- Exemplo: Babel que, embora não seja um bundler, ele pode realizar otimizações simples, como:
 - Remoção de espaços em branco desnecessários;
 - Renomear variáveis para reduzir o tamanho do arquivo.
- Em muitos casos, bundlers e compiladores são usados juntos em um projeto para fornecer uma série de otimizações que melhoram o desempenho e a eficiência do código;
- No entanto, é importante observar que os bundlers geralmente se concentram em otimizações específicas para o processo de empacotamento e carregamento do código;
- Já os compiladores podem realizar otimizações mais abrangentes durante o processo de transpilação ("tradução") do código-fonte.

webpack

webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/>



- O **Webpack** é um **empacotador de módulos** utilizado em projetos de desenvolvimento web;
- Responsável por **agrupar e otimizar** recursos como JavaScript, imagens, fontes e CSS;
- Ele cria um **gráfico de dependência** que mapeia cada módulo e suas relações;
 - Rastreia quais módulos dependem de quais outros módulos.
- Após o processamento, o Webpack gera um ou mais pacotes;
 - Quando você executa o Webpack, ele analisa essas dependências e **empacota** todos os módulos necessários em um ou mais arquivos de saída;
 - Isso permite que você **carregue apenas o código necessário** para a funcionalidade específica que está sendo usada, otimizando o desempenho da aplicação.
- Vantagens na utilização do Webpack:
 - **Otimização:** Permite otimizar processos, mesmo em projetos de diferentes tamanhos;
 - **Desenvolvimento Independente:** Equipes podem trabalhar em módulos separados;
 - **Controle de Dependências:** Ajuda a gerenciar as dependências entre os módulos.
 - Módulos da aplicação com interdependência;
 - Por exemplo, um arquivo JavaScript pode importar uma função de outro arquivo.
 - Trata de como esses módulos se relacionam e como suas dependências são gerenciadas.

Entry Point (Ponto de Entrada)

- Um ponto de entrada indica qual módulo o webpack deve usar para começar a construir seu grafo de dependência;
- O webpack irá determinar quais outros módulos e bibliotecas esse ponto de entrada depende (direta e indiretamente).
 - Por padrão, o valor utilizado é `./src/index.js`;


- É possível especificar um arquivo diferente, ou então vários arquivos como ponto de entrada (entry point).

```
module.exports = {  
  entry: './path/to/my/entry/file.js',  
};
```

Mais detalhes em:

Entry Points | webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/concepts/entry-points/>



Output (Saída)

- A propriedade de saída (output) indica ao webpack para qual pasta os arquivos que passaram pelo processo de empacotamento devem estar e como nomear esses arquivos;
 - Por padrão, é `./dist/main.js` para o arquivo de saída principal e `./dist` para qualquer outro arquivo gerado.

```
const path = require('path');  
  
module.exports = {  
  entry: './path/to/my/entry/file.js',  
  output: {  
    path: path.resolve(__dirname, 'dist'),  
    filename: 'my-first-webpack.bundle.js',  
  },  
};
```

- `output.filename`: nome do arquivo que será gerado;

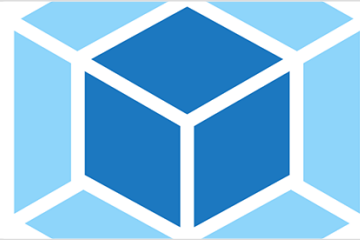
- `output.path`: pasta de destino após realizar o processo de *bundler*.

Mais detalhes em:

Output | webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/concepts/output/>



Loaders (Carregadores)

- Por padrão, o webpack apenas entende arquivos JavaScript e JSON;
- Os loaders permitem que o webpack processe outros tipos de arquivos e os converta em módulos válidos que podem:
 - Ser consumidos por sua aplicação;
 - Adicionados ao grafo de dependência.

```
const path = require('path');

module.exports = {
  output: {
    filename: 'my-first-webpack.bundle.js',
  },
  module: {
    rules: [{ test: /\.txt$/, use: 'raw-loader' }],
  },
};
```

- Esta configuração do webpack diz que todos os arquivos com extensão `.txt` serão processados pelo `raw-loader`, que carrega esses arquivos como texto bruto, sem fazer qualquer modificação no seu conteúdo;
 - `module`: Indica que estamos configurando as regras para o sistema de módulos do webpack;

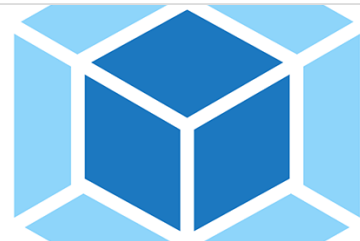
- `rules` : É um array de objetos que descrevem como os diferentes tipos de arquivos devem ser tratados.
- `test` : É uma expressão regular que define quais arquivos serão processados por esta regra;
 - Neste caso, `/\.txt $ /` seleciona todos os arquivos com extensão `.txt`.
- `use` : Indica qual loader (carregador) deve ser usado para processar os arquivos que correspondem ao teste definido acima;
 - Aqui, `'raw-loader'` é o nome do loader a ser utilizado;
 - Este loader é responsável por carregar arquivos de texto brutos sem modificar seu conteúdo.

Mais detalhes em:

Loaders | webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/concepts/loaders/>



Plugins

- Enquanto os *loaders* (carregadores) são utilizados para transformar certos tipos de módulos, os plugins podem ser aproveitados para realizar uma gama mais ampla de tarefas, como:
 - Otimização de pacotes;
 - Gerenciamento de ativos;
 - Injeção de variáveis de ambiente.
- A maioria dos plugins é personalizável por meio de opções;
 - Você pode usar um plugin várias vezes em uma configuração para diferentes fins.

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const webpack = require('webpack'); //to access built-in plugins
```


```
module.exports = {
  module: {
    rules: [{ test: /\.txt$/, use: 'raw-loader' }],
  },
  plugins: [new HtmlWebpackPlugin({ template: './src/index.html'
});
```

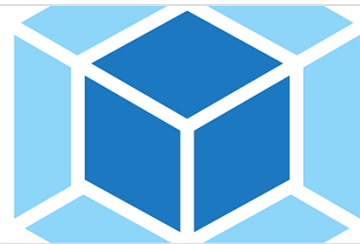
No exemplo acima, o `html-webpack-plugin` gera um arquivo HTML para sua aplicação e automaticamente injeta todos os seus pacotes gerados neste arquivo.

Lista de plugins disponíveis para o Webpack:

Plugins | webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/plugins/>



Mode (Modo)

- Ao definir o parâmetro mode como `development`, `production` ou `none`, você pode habilitar as otimizações integradas do webpack que correspondem a cada ambiente;
 - O valor padrão é `production`.

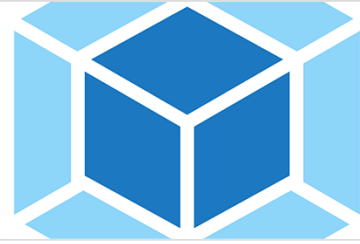
```
module.exports = {
  mode: 'production',
};
```

Mais detalhes em:

Mode | webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/configuration/mode/>



Suporte à navegadores

- O Webpack suporta todos os navegadores compatíveis com ES5;
 - IE8 e anteriores não são suportados.
- Se você deseja oferecer suporte a navegadores mais antigos, será necessário carregar um polyfill antes de usar essas expressões.



Um "polyfill" é uma peça de código (geralmente em JavaScript) que fornece funcionalidades modernas em navegadores mais antigos que não as suportam nativamente.

O termo "*polyfill*" é uma combinação de "poly", que significa "muitos" ou "vários", e "fill", que significa "preencher".


Em essência, um polyfill preenche as lacunas em recursos que não são suportados pelos navegadores mais antigos, permitindo que os desenvolvedores usem funcionalidades mais recentes em todos os navegadores de forma consistente.

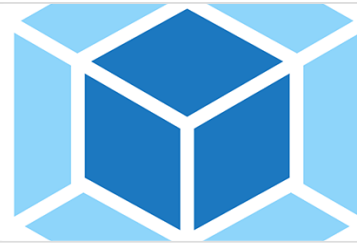
Por exemplo, se um navegador mais antigo não suporta o método `forEach()` de arrays do JavaScript, um polyfill pode ser usado para adicionar essa funcionalidade ao navegador, garantindo que o código seja executado corretamente em todos os navegadores.

Mais detalhes em:

Shimming | webpack

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or

 <https://webpack.js.org/guides/shimming/#loading-polyfills>



esbuild

- É um empacotador e minificador JavaScript;
- É de 10 a 100 vezes mais rápido do que seus concorrentes como webpack e Rollup;
- Seu objetivo é melhorar significativamente o processo de construção de aplicações JavaScript;
- **Empacotamento de JavaScript e CSS:**
 - O esbuild empacota o código JavaScript em um único arquivo;
 - Suas principais funções incluem:
 - Resolver módulos;
 - Relatar problemas de sintaxe;
 - Tree-shaking (remover funções não utilizadas);
 - Eliminar declarações de log e depuração;
 - Minificar o código;
 - Etc.
 - Para CSS, o esbuild também empacota o código em um único arquivo;
 - Lida com pontos como:
 - Problemas de sintaxe;
 - Aninhamento;
 - Codificação de ativos embutidos;
 - Incorporação de recursos (como imagens, fontes ou outros arquivos) diretamente no código CSS ou JavaScript;
 - Por exemplo, em vez de carregar uma imagem externa por meio de uma URL, você pode codificar a imagem diretamente no arquivo CSS ou JavaScript usando **base64**. Isso reduz a

quantidade de solicitações de rede, mas também aumenta o tamanho do arquivo.

- Minificação;
- Mapas de origem (source maps);
 - Gerados durante o processo de compilação ou empacotamento;
 - Mapeiam o código minificado ou transpilado de volta para o código-fonte original, facilitando a depuração;
 - Quando você inspeciona um erro no navegador, os mapas de origem permitem que você veja o código-fonte real em vez do código minificado, tornando a depuração mais fácil.
- Prefixação automática.
 - Alguns navegadores ainda requerem prefixos específicos para reconhecer certas funcionalidades CSS;
 - Por exemplo, para garantir que uma propriedade `transform` funcione em todos os navegadores, você pode usar `transform: rotate(45deg)` e o esbuild automaticamente adicionará os prefixos necessários, como `-webkit-transform` e `-moz-transform`.
- **Servidor de Desenvolvimento Local:**
 - O esbuild oferece um servidor de desenvolvimento local com empacotamento automático e recarga rápida.
- Ao contrário dos empacotadores JavaScript tradicionais, o esbuild é um **executável Go compilado** que aproveita o processamento paralelo intenso;
- É significativamente mais rápido que outros, podendo economizar o tempo de desenvolvimento;
- Outros recursos incluem empacotamento e compilação integrados para JavaScript, TypeScript, JSX e CSS;
- Documentação abrangente.

Instalação do esbuild

esbuild - Getting Started

First, download and install the esbuild command locally. A prebuilt native executable can be installed using npm (which is automatically installed when you install the node JavaScript runtime):

» <https://esbuild.github.io/getting-started/#install-esbuild>

Por que o eslint é rápido?

<https://esbuild.github.io/faq/#why-is-esbuild-fast>

esbuild - An extremely fast bundler for the web

Our current build tools for the web are 10-100x slower than they could be. The main goal of the esbuild bundler project is to bring about a new era of build tool performance, and create an easy-to-use modern bundler along the way.

» <https://esbuild.github.io/>

Referências

Bundlers and Compilers

Something very common when we are starting to work with React and these more modern JavaScript development technologies, is that we realize...

» <https://medium.com/@enoquetembe/bundlers-and-compilers-b07059abfd7a>



JavaScript Compiler

How does the JavaScript compiler work?

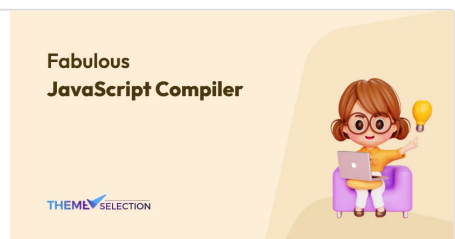
» <https://medium.com/illuminations-mirror/javascript-compiler-1b6eb09bd1a0>



Fabulous JavaScript Compilers To Use in 2024

Check out this awesome collection of the best JavaScript Compilers. Streamline your workflow using these awesome compilers for JavaScript.

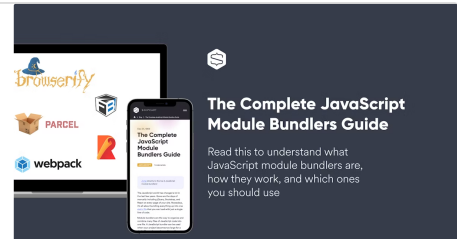
» <https://themeselection.com/javascript-compilers/>



JavaScript Bundlers: In-Depth Guide

This in-depth guide will help you understand what you need to know about JavaScript module bundlers. A list of the top 5 best JS bundlers is also included.


 <https://snipcart.com/blog/javascript-module-bundler>



JavaScript é interpretado ou compilado em tempo de execução?

Nesta outra pergunta eu perguntei a mesma coisa, mas em relação a Java. Agora pergunto sobre o JavaScript.

Pelo que eu saiba, historicamente o JavaScript sempre foi interpretado,

 <https://pt.stackoverflow.com/questions/26127/javascript-é-interpretado-ou-compilado-em-tempo-de-execução>



O que é : Just-In-Time (JIT) Optimization • Napoleon

O que é Just-In-Time (JIT) Optimization? Just-In-Time (JIT) Optimization é uma técnica utilizada no desenvolvimento de software para melhorar o desempenho e a eficiência de um

n. <https://napoleon.com.br/glossario/o-que-e-just-in-time-jit-optimization/>

