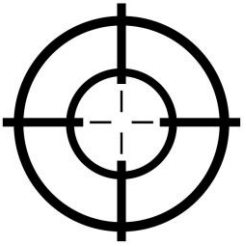


Instalação

# O que é o TypeScript?

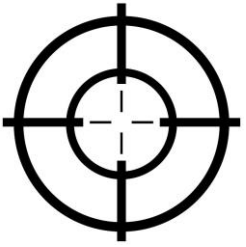


TypeScript é uma linguagem de programação desenvolvida e mantida pela Microsoft.

É um superconjunto sintático estrito de JavaScript, que adiciona tipagem estática opcional à linguagem. O TypeScript foi projetado para o desenvolvimento de grandes aplicativos e transcompilações para JavaScript.

Como TypeScript é um superconjunto de JavaScript, os programas JavaScript existentes também são programas TypeScript válidos.

# Instalação...



Existem duas maneiras principais de instalar o TypeScript, uma via npm a outra instalando os plugins do VS para a linguagem.

→ `npm install -g typescript`

Instalando o módulo como global, assim fica disponível para qualquer projeto.

Pode-se usar outros gerenciadores de dependências como yarn ou pnpm para baixar e instalar o TypeScript.

# TypeScript para desenvolvedores Java ou C#...



TypeScript é uma escolha popular para programadores acostumados com outras linguagens com tipagem estática, como C# e Java.

Embora o TypeScript forneça muitos recursos familiares para esses programadores, vale lembrar que o JavaScript (e, portanto, o TypeScript) difere das linguagens orientadas à objetos tradicionais.

# Vantagens do TypeScript...



TypeScript apresenta tipagem estática, é uma linguagem fortemente tipada. Isso significa que pode-se declarar uma variável com um tipo específico, uma vez declarada, uma variável não muda seu tipo e pode assumir apenas alguns valores.

Por isso, o compilador alerta os desenvolvedores sobre erros relacionados ao tipo, evitando que eles cheguem a fase de produção. Este comportamento não é possível no JavaScript.

# Vantagens do TypeScript...



O TypeScript pode ser usado para desenvolver aplicativos JavaScript para execução do lado do cliente (client side) e do lado do servidor (server side), com softwares como com node.js ou deno.

O TypeScript oferece suporte a conceitos de programação orientada a objetos, como classes, interfaces, herança e muito mais.

# Principal desvantagem...



Os navegadores não podem interpretar o código TypeScript. Por isso, precisa-se da transcompilação para JavaScript antes de executá-lo.

No entanto, esse processo é altamente automatizado e não requer muito tempo adicional. No total, a desvantagem dessa etapa é muito menos significativa do que seus benefícios.

# No windows...



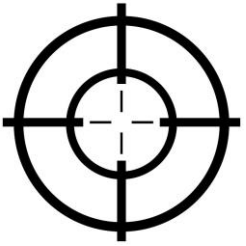
As vezes, antes de iniciar a compilação de arquivos .ts, é necessário habilitar a execução de scripts.

Para isso, deve-se solicitar permissão no powershell, digitando "Set-ExecutionPolicy Unrestricted" e optar por permitir. A execução do powershell precisa ser em modo administrador.

Geralmente, sistema Windows bloqueiam a execução do compilador (transcompilador) do TypeScript.



# Desenvolvendo em TypeScript...



A extensão padrão para um código fonte TypeScript é “.ts”.

→ `somador.ts`

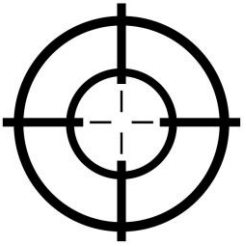
```
class Somador {  
    public somar(numero1: number, numero2: number) {  
        return numero1 + numero2  
    }  
}
```

```
let somador = new Somador()
```

```
console.log(somador.somar(10, 5))
```

Método público, que recebe como parâmetro duas variáveis `number` - que aceitam apenas valores numéricos.

# Compilando o código...



O node.js, assim como outros interpretadores, não executam código TypeScript por padrão. Por isso é preciso compilar para JavaScript.

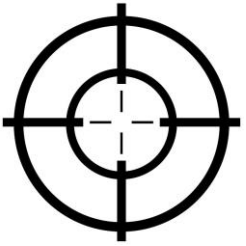
Para isso, utiliza-se o compilador da linguagem.

```
tsc somador.ts
```

```
npx tsc somador.ts
```

O npx é uma ferramenta utilizada para executar pacotes do npm, geralmente, utilizada quando o pacote não está instalado globalmente.

# Comparando os códigos...



A compilação de um código “.ts”, resulta em um outro “.js”.

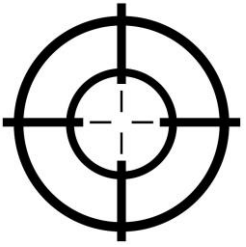
```
class Somador {  
    public somar(numero1: number, numero2: number) {  
        return numero1 + numero2  
    }  
}
```

```
let somador = new Somador()
```

```
console.log(somador.somar(10, 5))
```

```
var Somador = /** @class */ (function () {  
    function Somador() {  
    }  
    Somador.prototype.somar = function (numero1, numero2) {  
        return numero1 + numero2;  
    };  
    return Somador;  
})();  
var somador = new Somador();  
console.log(somador.somar(10, 5));
```

# Executando o código...



A compilação de um código “.ts”, resulta em um outro “.js”. Por padrão, o node.js não consegue executar arquivos .ts. Por isso, após a compilação deve-se executar o .js gerado. Lembre-se, o node.js é o executor de arquivos .js.

```
var Somador = /** @class */ (function () {  
    function Somador() {  
    }  
    Somador.prototype.somar = function (numero1, numero2) {  
        return numero1 + numero2;  
    };  
    return Somador;  
})();  
var somador = new Somador();  
console.log(somador.somar(10, 5));
```

node somador.js

# Configurações do compilador

TS...



Assim como em um módulo, que possui um arquivo de configuração chamado `package.json`, um projeto de TypeScript pode ter um arquivo de configuração, chamado `tsconfig.json`.

A presença de um arquivo `tsconfig.json`, em um diretório, indica que esse diretório é a raiz do projeto TypeScript.

Projetos JavaScript podem ter um arquivo `jsconfig.json`, que tem quase o mesmo propósito.

# Para que serve o tsconfig.json?

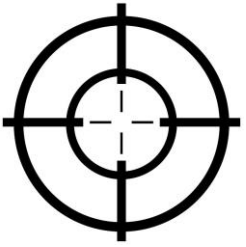


O arquivo tsconfig.json especifica os arquivos raiz e as configurações de compilação necessárias para o projeto.

Entende-se como arquivos raiz, os arquivos de código fonte, ou seja, os arquivos .ts.

Além dos arquivos raiz, o tsconfig.json também pode especificar diretórios de armazenamento de arquivos raiz e o diretório de saída após a compilação, onde estarão os arquivos .js.

# Apontando o tsconfig.json para o compilador...



Existem duas formas de “mostrar” para o tsc onde está o arquivo de configuração.

A primeira é invocando o tsc sem os arquivos de entrada como argumentos, nesse caso o compilador procura o arquivo tsconfig.json, começando no diretório atual e continua em suas subpastas.

# Apontando o tsconfig.json para o compilador...



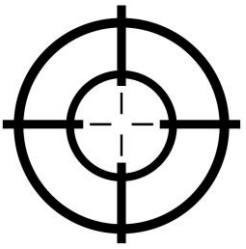
A segunda forma é invocando o tsc sem os arquivos de entrada como argumentos e a opção de linha de comando --project (ou apenas -p), que especifica o caminho para o diretório que contém o arquivo tsconfig.json ou o caminho para um arquivo .json válido, contendo as configurações.

```
tsc -p .\tsconfig.json
```





# Como é o tsconfig.json por dentro?

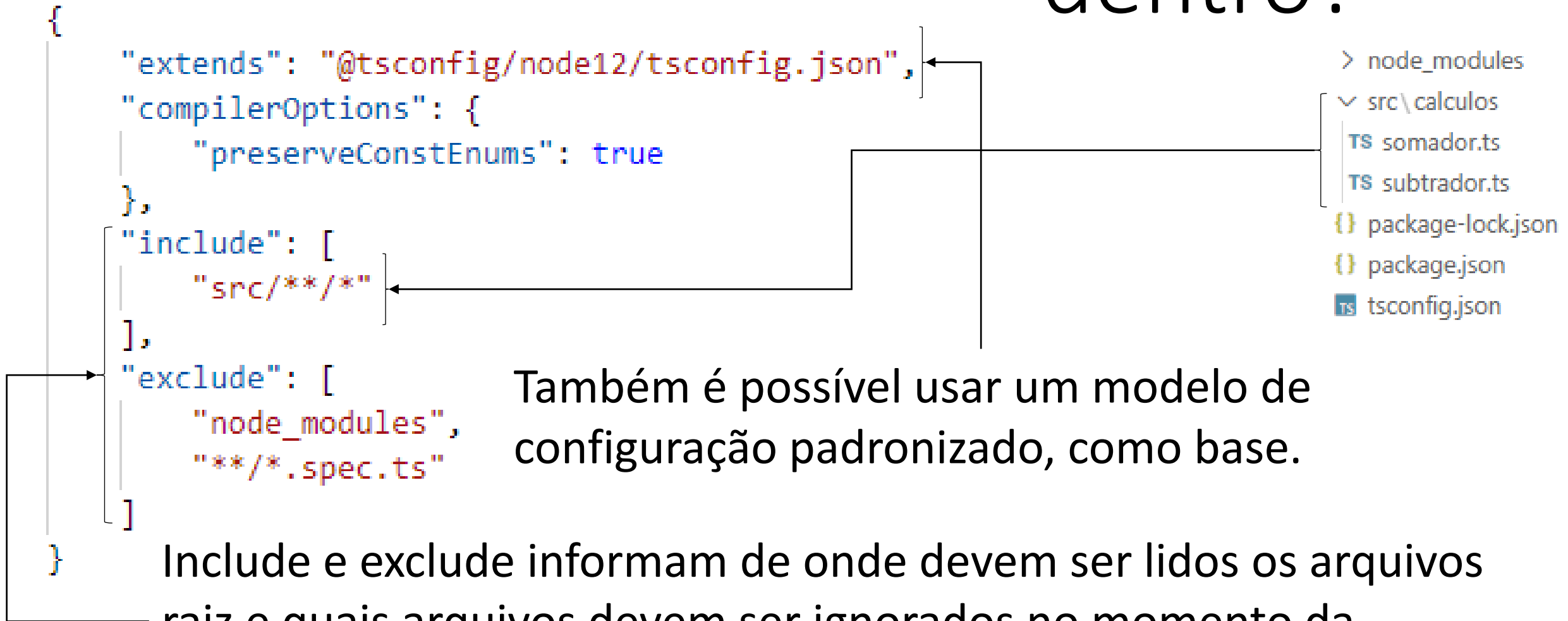
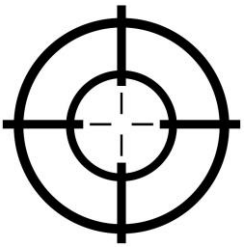


```
{
  "compilerOptions": {
    "module": "commonjs",
    "noImplicitAny": true,
    "removeComments": true,
    "preserveConstEnums": true,
    "sourceMap": true
  },
  "files": [
    "somador.ts",
    "subtrador.ts"
  ]
}
```

Configurações que informam se o tsc deve fazer alguma verificação no código, antes de transcompilar para .js.

Utilizando o atributo files para identificar os arquivos raiz, que podem estar no mesmo diretório ou em subdiretórios.

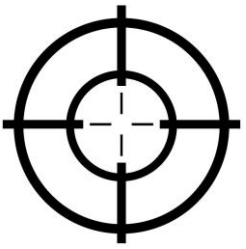
# Como é o tsconfig.json por dentro?



Também é possível usar um modelo de configuração padronizado, como base.

Include e exclude informam de onde devem ser lidos os arquivos raiz e quais arquivos devem ser ignorados no momento da transcompilação.

# Como é o tsconfig.json por dentro?

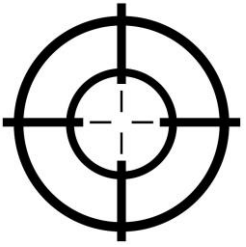


```
{
  "extends": "@tsconfig/node12/tsconfig.json",
  "compilerOptions": {
    "preserveConstEnums": true
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "node_modules",
    "**/*.spec.ts"
  ]
}
```

> node\_modules  
✓ src\calculos  
  TS somador.ts  
  TS subtrador.ts  
  {} package-lock.json  
  {} package.json  
  TS tsconfig.json

include e exclude suportam caracteres curinga como: \* corresponde a zero ou mais caracteres (excluindo separadores de diretório), ? corresponde a qualquer caractere (excluindo separadores de diretório) e \*\*/ corresponde a qualquer diretório aninhado a qualquer nível.

# Configurações padrões...



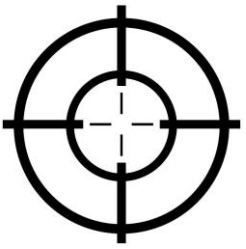
Pode-se escolher uma configuração para o tsconfig.json, que se concentre nas escolhas particulares do projeto. Já existem algumas configurações básicas e espera-se que a comunidade possa adicionar mais, para diferentes ambientes.

<https://www.npmjs.com/package/@tsconfig/recommended>

<https://www.npmjs.com/package/@tsconfig/react-native>

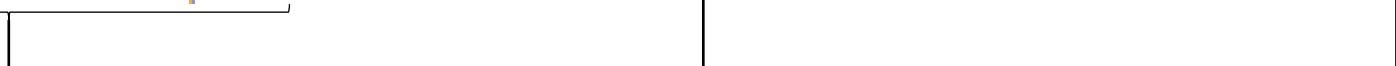
<https://www.typescriptlang.org/pt/tsconfig>

# Especificando o local de saída da compilação...



Geralmente, os arquivos compilados são criados no mesmo diretório dos seus respectivos arquivos raiz. Mas, isto pode ser modificado na configuração.

```
{  
  "extends": "@tsconfig/node12/tsconfig.json",  
  "compilerOptions": {  
    "preserveConstEnums": true,  
    "outDir": "src/output"  
  },  
  "include": [  
    "src/**/*"  
  ],  
}
```



```
src  
├── calculos  
│   ├── TS somador.ts  
│   └── TS subtrador.ts  
└── output  
    ├── JS somador.js  
    └── JS subtrador.js
```

# Tudo que pode ter dentro do tsconfig.json...



Pode-se colocar, retirar ou modificar várias configurações no arquivo tsconfig.json. As modificações a serem colocadas dependem do tipo de projeto que será criado. Uma forma de descobrir as configurações possíveis é acessando o esquema com todas disponíveis.

<http://json.schemastore.org/tsconfig>

# Criando o tsconfig.json com tudo...



Decorar todos os atributos possíveis da configuração do tsc é complicado, talvez inviável. Mas, pode-se criar um arquivo de configuração com tudo, com os atributos comentados.

```
{
  "compilerOptions": {
    /* Visit https://aka.ms/tsconfig.json to read more about this file */

    /* Basic Options */
    // "incremental": true,
    "target": "es5",
    "module": "commonjs",
    // "lib": [],
    // "allowJs": true,
    // "checkJs": true,
    // "jsx": "preserve",
    // "declaration": true,
    // "declarationMap": true,

    /* Enable incremental compilation */
    /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES6', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019', 'ES2020', 'ES2021', 'ES2022', 'ESNEXT'. */
    /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'es2020', 'esnext'. */
    /* Specify library files to be included in the compilation. */
    /* Allow javascript files to be compiled. */
    /* Report errors in .js files. */
    /* Specify JSX code generation: 'preserve', 'react-native', 'react'. */
    /* Generates corresponding '.d.ts' file. */
    /* Generates a sourcemap for each corresponding '.d.ts' file. */
  }
}
```

`tsc --init`

# Alguns detalhes...



O atributo “module” indica qual será a forma de resolução na transcompilação.

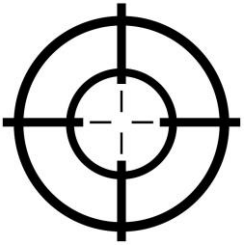
```
"target": "es5",  
/* Specify module code generation:  
'none', 'commonjs', 'amd', 'system', 'umd',  
'es2015', 'es2020', or 'ESNext'. */  
"module": "es2015",
```

Por exemplo, usar “es2015”, informaria para o tsc que seu interpretador JavaScript, que será usado na transcompilação, é capaz de analisar instruções de importação do ES6.

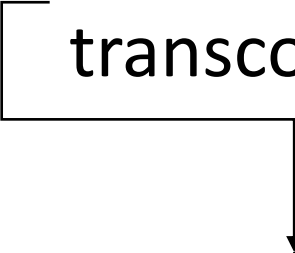
Algumas versões antigas do node.js não suportam os recursos do ES6.



# Alguns detalhes...



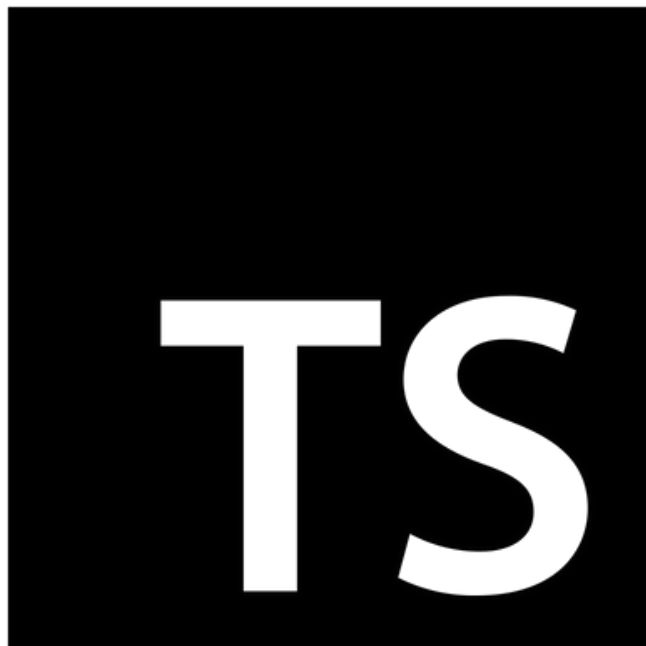
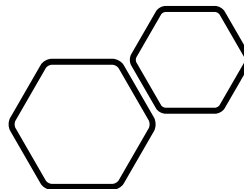
O atributo “target” informa como deverá ser o código de saída, após a transcompilação.



```
"target": "es5",  
/* Specify module code generation:  
'none', 'commonjs', 'amd', 'system', 'umd',  
'es2015', 'es2020', or 'ESNext'. */  
"module": "es2015",
```

A transcompilação do código ts visa uma variante específica da linguagem JavaScript.

Se o valor para target for ES5, por exemplo, o código compilado poderá ser executado por navegadores e interpretadores compatíveis com ES5.



TypeScript