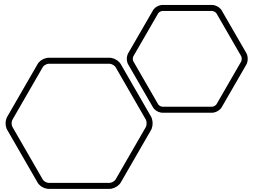


Módulos

Exportando....

Assim como com JS, também pode-se exportar módulos TS.

```ts
export default class Somador {
    public somar = (numero1: number, numero2: number) => {
        return numero1 + numero2
    }
}
```

Lembre-se, esta é a forma quando existe apenas um elemento para exportar.

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

export default Somador
```

Detalhe sobe a exportação....

Lembre-se, código JS também é código TS. O node.js, por padrão, não executa código TS. Na exportação, o arquivo transcompilado .js deve ser apontado como saída do módulo, no arquivo package.json.

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

export default Somador
```

```
"description": "",
"main": "somador.js",
```

Detalhe sobe a exportação....

Lembre-se! A palavra reservada “import” só pode ser usada em módulos definidos pelo ES6. Para que um módulo seja entendido como módulo ES6, o atributo “type”, do arquivo package.json deve ser configurado.

```
"main": "somador.js",
"type": "module",
```

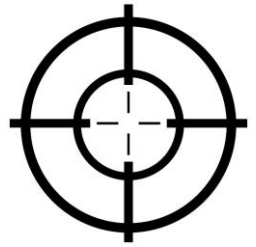
Uma outra configuração importante é o tipo de resolução do módulo, que impacta na forma como o tsc transcompila o código .ts.

```
{
  "compilerOptions": {
    "target": "ES2015",
    "module": "es2015",
    "strict": true,
    "noImplicitAny": false,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "moduleResolution": "node"
  },
}
```

Boa prática

- Uma boa prática para criação de módulos é nomear o diretório do módulo com mesmo nome descrito no arquivo package.json

Exportando mais de um elemento...



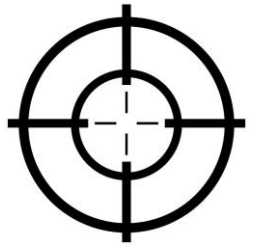
Também pode-se exportar mais de um elemento no mesmo módulo.

```
class Somador {
    public calcular = (numero1: number, numero2: number) => {
        return numero1 + numero2
    }
}

class Subtrador {
    public calcular = (numero1: number, numero2: number) => {
        return numero1 - numero2
    }
}

export {Somador, Subtrador}
```

```
{
  "name": "calculos",
  "version": "1.0.0",
  "description": "",
  "main": "calculos.js",
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Perceba que é um forma diferente, de quando se tem apenas um elemento para exportar.

Importando mais de um elemento...



- Também pode-se importar mais de um elemento do mesmo módulo.

```
import { Somador, Subtrador } from 'calculos'

let somador = new Somador()
let subtrador = new Subtrador()

console.log(somador.calcular(2, 2))
console.log(subtrador.calcular(10, 5))
```

```
{
  "name": "calculos",
  "version": "1.0.0",
  "description": "",
  "main": "calculos.js",
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Todo módulo tem um arquivo de configuração?

A resposta é não! JavaScript tem uma longa história, com diferentes maneiras de lidar com a modularização de código.

Ao longo do tempo, a comunidade e a especificação JS convergiram para um formato denominado módulos ES (ou módulos ES6), que é conhecido pelo uso das palavras-chave import e export.

O padrão ES foi adicionado à especificação JS em 2015 e, em 2020, atingiu um amplo suporte na maioria dos navegadores e interpretadores JS.

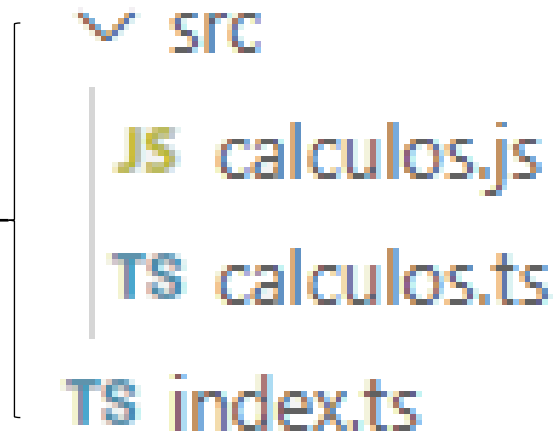
Quando um módulo não precisa de um arquivo de configuração?

As definições do ECMAScript 2015 informam que qualquer arquivo que contenha as palavras-chave import ou export no seu topo é considerado um módulo.

Portanto, qualquer arquivo que não contenha as palavras-chave import ou export não é considerado um módulo. Estes arquivos são considerados scripts, como os scripts comuns usados para manipular páginas html.

Pode-se criar módulos, dentro do mesmo projeto.

```
import {Somador, Subtrador} from "./src/calculos.js"

let somador  = new Somador()
let subtrador = new Subtrador()

console.log(somador.calcular(2,2))
console.log(subtrador.calcular(10,5))
```



Neste código acontece uma importação a um módulo, já transcompilado, que está dentro do diretório src. Este módulo não tem um arquivo de configuração.

Por que criar módulos com arquivo de configuração?

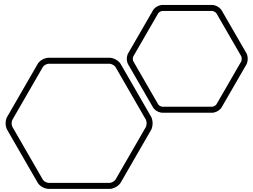
Provavelmente, esta pergunta já foi respondida antes, mas não custa relembrar.

O avanço da tecnologia provocou uma grande mudança, tornou-se necessário fornecer mecanismos para dividir programas JavaScript em partes separadas, que podem ser importadas quando necessário, como bibliotecas. Estas partes são chamadas de módulos.

Além disso, módulos podem ser ofertados a partir de repositórios, como no GitHub. Para isto precisam de um arquivo de configuração.

Documentação

- Para saber mais sobre o TS, seu compilador e detalhes sobre módulos, deve-se buscar informações na documentação: https://www.typescriptlang.org/docs/handbook.



TypeScript