rocketseat BLOG

JAVASCRIPT ROCKETSEAT

TypeScript: Vantagens, mitos, dicas e conceitos fundamentais

por **Diego Fernandes** há 2 anos 6 MIN DE LEITURA

Nesse post vamos entender as **vantagens** da utilização de **tipagem estática** em nosso código **JavaScript** utilizando **TypeScript**, além disso iremos abordar alguns mitos associados com a utilização dessa ferramenta.

Antes de mais nada, para introdução ao assunto, o TypeScript é uma ferramenta que adiciona tipagem estática ao JavaScript que por padrão é uma linguagem que possui **tipagem dinâmica**, ou seja, as variáveis e funções podem assumir tipos distintos durante o tempo de execução.

Vale lembrar o código TypeScript é utilizando somente em ambiente de desenvolvimento e é **totalmente convertido para JavaScript** no processo de build de produção, ou seja, o navegador ou o Node lerão somente código JS no fim das contas.

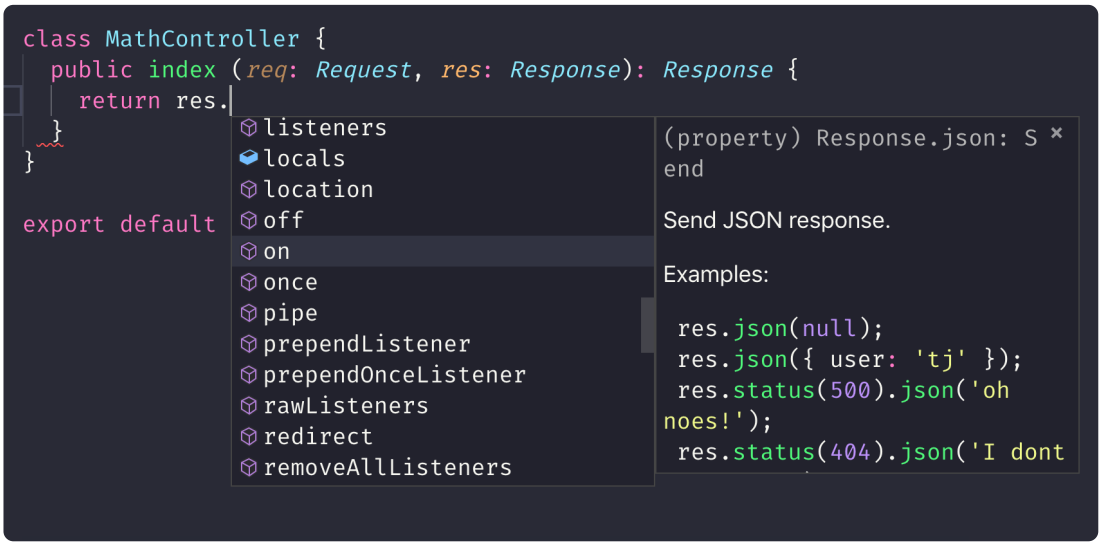
Vantagens do TypeScript

A grande empolgação por trás de usar uma ferramenta como essa é a possibilidade de **descobrir erros** durante o desenvolvimento e incrementar a **inteligência (*IntelliSense*) da IDE** que estamos utilizando.

No exemplo abaixo criamos um controller para uma aplicação back-end utilizando Node.js e ExpressJS. Esse

arquivo exporta métodos que funcionam como rotas dentro do nosso app e sempre recebem uma requisição e resposta.

Sem o TypeScript, **nossa IDE jamais saberia o formato** desses parâmetros e precisaríamos consultar a documentação do ExpressJS para entender os métodos e valores que poderíamos requisitar.



Exemplo da IntelliSense do VSCode com TypeScript

Além de ajudar no ambiente de desenvolvimento, o TypeScript ainda permite que utilizemos de funcionalidades da linguagem que ainda não estão disponíveis de forma nativa, por exemplo, no Node.js podemos utilizar os ES Modules (import/export) normalmente.

Outro ponto bem legal é que podemos transpilar nosso código para que o mesmo seja lido por todas versões de browsers, assim como fazemos com o Babel em aplicações totalmente JavaScript.

De onde vem os tipos?

As bibliotecas escritas com TypeScript geralmente incluem um **arquivo de definição** de tipos (extensão `.d.ts`) gerado automaticamente. Esse arquivo guarda a informação de tipagem de todas variáveis, funções, classes, etc, exportados pela biblioteca.

Na grande maioria das vezes, esses tipos são instalados separadamente da lib principal, por exemplo, quando instalamos o React em nosso app é legal instalarmos as definições de tipo da ferramenta também dessa forma:

```
yarn add react
yarn add -D @types/react
```

Mitos do TypeScript

Antes de entendermos na prática o que muda quando começamos a tipar nosso código, vamos desmistificar algumas coisas sobre o TypeScript:

❓ O TypeScript diminui a produtividade

Não tenho como negar, meu medo maior de entrar no mundo do TypeScript sempre foi **produtividade**, mas eu acho que o TypeScript simplesmente não é uma tecnologia pra todos.

Na minha opinião na grande maioria dos casos o TypeScript **faz você perder produtividade** no início do aprendizado

sim, são novos conceitos, novas regras e acaba que você se vê acessando a documentação inúmeras vezes pra entender alguns funcionamentos.

Mas isso é um processo, e na minha visão quando uma aplicação cresce, passa a ter mais desenvolvedores trabalhando junto ou até é aberta de forma open-source a tipagem estática ajuda muito. Isso porque o código fica muito mais simples de ser entendido.

❌ O TypeScript vai substituir o JavaScript

Apesar de muita gente tratar o TypeScript como uma linguagem, na minha opinião, o TS (assim como a própria definição do site) é um conjunto de funcionalidades adicionadas ao JavaScript.

Isso quer dizer que o TypeScript gira em torno dos avanços da **ECMAScript** que tem como foco principal o JavaScript e com base nisso monta suas funcionalidades.

❌ Preciso adicionar tipos à todas variáveis e funções

Durante o desenvolvimento do seu projeto o TypeScript "executa" o código e encontra todas entradas e saídas possíveis tentando determinar **automaticamente** (inferência de tipos) o tipo das variáveis, parâmetros e retornos de funções.

Isso quer dizer que para declarar uma variável numérica, por exemplo:

```
const a = 5;
```

O TypeScript já entende que a variável `a` é numérica e se tentarmos alterar seu valor para uma *string* posteriormente receberemos um erro.

❌ TypeScript é a única forma de tipagem

Mesmo com o TypeScript ainda temos outras formas de tipagem de código, por exemplo, no **React** podemos fazer a tipagem de propriedades dos componentes utilizando o PropTypes, que é um tipo de checagem de tipagem dinâmica que funciona em **tempo de execução**.

Tipagem estática vs dinâmica

O grande pulo do gato entre escolher entre cada uma dessas tipagens é que, a tipagem dinâmica acontece em **tempo de execução**, ou seja, você irá descobrir os erros apenas executando a aplicação enquanto que a tipagem estática pode garantir que o "*build*" do código não seja gerado caso algum erro de tipagem exista.

Flow

Além disso, outra ferramenta que faz o mesmo trabalho que o TypeScript é o **Flow**. Apesar de apresentar praticamente as mesmas funcionalidades, o TypeScript possui um **ecossistema** mais **maduro** e possui um suporte maior da **comunidade**, por isso, recomendo seguir com o TS por enquanto.

JSDocs

Um conceito mais novo, mas que também é bem promissor é adicionar tipagem via JSDocs, ou seja, através de comentários. Alguns exemplos:

```
/**
 * @type {{name: string}, {age: number}}
 */
const person = {
  name: 'Joe',
  age: 32
}
```

Com isso, se você estiver utilizando o Visual Studio Code, pode adicionar a configuração:

```
"javascript.implicitProjectConfig.checkJs":
true
```

Você pode conferir todas opções aceitas do JSDoc no VSCode na documentação do TypeScript.

Apesar desse modelo de tipagem via comentários ser extremamente performático por não precisarmos transpilar o código JS, na minha opinião, acaba sujando muito o código e o TypeScript tem uma sintaxe mais bonita.

❌ Preciso aprender muita coisa pra iniciar com TypeScript

Último mito que também está incorreto, com a quantidade de ferramentas que temos disponíveis por volta do ecossistema do TypeScript e com a integração excepcional do VSCode com essa ferramenta, fica muito simples iniciarmos com a utilização dessa ferramenta.

Conceitos iniciais

Agora que já entendemos as vantagens e desmistificamos alguns pontos do TypeScript vamos entender na prática alguns conceitos e aplicações dessa ferramenta.

Tipagem simples

Para tipar variáveis adicionarmos o `:tipo` após sua declaração, lembrando que nem todas variáveis precisam tipagem devido a **inferência de tipos** do TypeScript.

```
const x: number = 3;

const vetor: number[] = [1, 2, 3];
```

Para tipar os parâmetros de uma função informamos o tipo logo após a declaração do argumento e para o retorno de função adicionamos a tipagem logo após fechar os parênteses dos parâmetros.

```
function compare (x: number, y: number): string {
  return x > y ? 'X maior que Y' : 'Y maior que X';
}
```

Interfaces

Muitas vezes precisamos reaproveitar tipagens entre vários arquivos e funções da aplicação, nisso entram as interfaces, por exemplo:

```
interface Pessoa {
  nome: string
  idade: number
}

function bomDia (pessoa: Pessoa): string {
  return `Bom-dia ${pessoa.nome}`;
}

function maiorDeIdade (pessoa: Pessoa): boolean {
  return pessoa.idade >= 18;
}
```

Types

Quando uma variável pode assumir formatos distintos mesmo que pertencendo a uma mesma entidade podemos

utilizar os Types. Esses se diferem das interfaces em alguns pontos como:

- Interfaces podem herdar outras interfaces, Types não;
- Types pode assumir formatos distintos;

```
type Polygon =
  { type: 'square', x: number } |
  { type: 'circle', radius: number } |
  { type: 'rectangle', x: number, y: number };

export function area (polygon: Polygon): number {}
```

Veja que no exemplo acima, o parâmetro da função pode assumir três formatos diferentes baseado no tipo de figura geométrica.

Enums

Os Enums são formas de definirmos constantes na tipagem a fim de reaproveitarmos código entre funções e/ou arquivos. Por exemplo, na função acima poderíamos usar os Enums da seguinte forma:

```
enum PolygonTypes { Square, Circle, Rectangle }

type Polygon =
  { type: PolygonTypes.Square, x: number } |
  { type: PolygonTypes.Circle, radius: number } |
  { type: PolygonTypes.Rectangle, x: number, y: number };

export function area (polygon: Polygon): number {
```

```
  switch (polygon.type) {
    case PolygonTypes.Square: return polygon.x ** 2
    case PolygonTypes.Rectangle: return polygon.x * polygon.y
    case PolygonTypes.Circle: return Math.PI * polygon.radius **
  }
}
```

Veja que definimos via Enum as possibilidades de tipos de polígonos possíveis e reaproveitamos essa tipagem dentro do corpo da função para fazer o switch/case.

Dicas

Durante meus primeiros passos com TypeScript eu sofri com alguns pontos e por isso gostaria de deixar algumas dicas:

1. Use Visual Studio Code;
2. Configure a parte de linting com ESLint (não TSLint);
3. Nem tsc, nem babel, para desenvolvimento utilize o sucrase (performance excepcional);
4. Para rodar testes com Jest adicione em seu `jest.config.js` :

```
{
  moduleFileExtensions: ['ts', 'tsx', 'js'],
  testMatch: ['**/__tests__/*.spec.+(ts|tsx|js)', '**/*.test.+(ts
  transform: {
    '^.+\\.(ts|tsx)$': 'ts-jest',
  }
}
```

5. Crie arquivos de configuração do TypeScript distintos para ambiente de desenvolvimento e produção, para otimizar o processo de build;

Fim: true

Além desses conceitos ainda existem outros vários que suprem as mais diversas necessidades de tipagem, mas como esse post tem como objetivo introduzir e conscientizar sobre o uso dessa ferramenta vou parando por aqui.

Se você quiser is mais a fundo, a documentação dessa ferramenta é extremamente completa e simples de entender.

Lembrando que a forma mais adequada de aprender algo é **colocando em prática**, então não fique com medo e comece construindo pequenas aplicações para sentir suas dificuldades e dúvidas.

PRÓXIMO POST
Paginação com React Router Dom

POST ANTERIOR
Utilizando Notification do browser

HOME BACK END FRONT END MOBILE