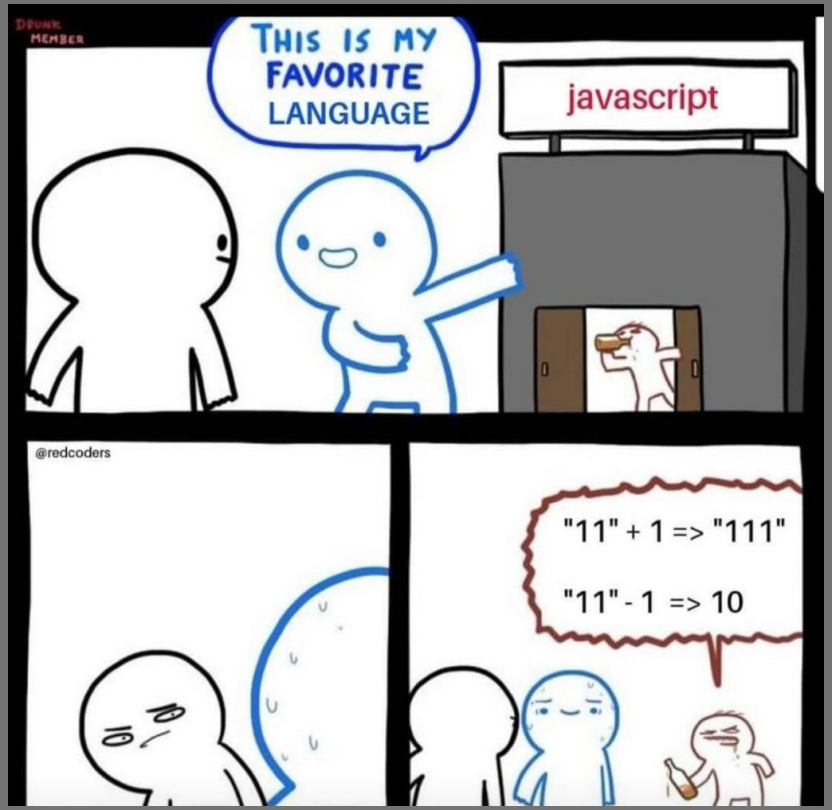
TypeScript

Precavendo erros, não resolvendo bugs.

ConFLOSS

Aqui vamos começar a entender sobre a linguagem TypeScript, seus valores e quais são os ganhos do uso.



Quais são os erros mais comuns em JavaScript que não são detectáveis em tempo de desenvolvimento e dependendo dos testes aplicados podem também não serem detectados?

- Operações matemáticas entre tipos diferentes de dados. Ex: string e número.
- Comparações entre informações de tipos diferentes usando o comparador idêntico. Ex: “10” === 10 (esperando retornar verdadeiro)
- Incremento ou decremento do valor de uma variável cujo valor não é numérico. Ex: var count = ‘1O’; count++

Qual é o propósito?

A linguagem TypeScript, na minha humilde opinião, veio com o propósito de precaver os erros mais comuns em desenvolvimento e que ocorrem com grande facilidade no JavaScript, geralmente ligados ao uso de qualquer informação sem antes verificar/validar se esta atende a requisitos básicos (tipo de dado, formato e etc.). Garantindo dessa forma que tais “erros” básicos não passem do ambiente de desenvolvimento/build.

Como funciona?

A linguagem funciona no contexto de desenvolvimento, verificando/validando as informações que transitam dentro do fluxo e garantindo assim que seus valores atendam às necessidades que garantem a execução sem “erros”.

Tudo que é escrito em TypeScript parte da sintaxe do próprio JavaScript adicionando a isso tais verificações/validações. E na compilação (build) tudo é transformado em JavaScript puro garantindo assim seu funcionamento nas inúmeras plataformas que o JavaScript roda.

Exemplo básico:

Um simples cadastro de pessoas onde os campos são: Name (nome), Age (idade) e Occupation (cargo).

Aqui queremos garantir que o campo Name não aceite qualquer coisa diferente de uma string, que o campo Age não aceite qualquer coisa diferente de um valor numérico e o campo Occupation não aceite qualquer coisa diferente das strings ‘Analista’, ‘Desenvolvedor(a) Júnior’, ‘Desenvolvedor(a) Pleno’ e ‘Desenvolvedor(a) Sênior’.

```
let Name: string;
let Age: number;
let Occupation: 'Analista'|'Desenvolvedor(a) Júnior'|'Desenvolvedor(a) Pleno'|'Desenvolvedor(a) Sênior';

Name = 'Aristóteles';
Age = 22;
Occupation = 'Desenvolvedor(a) Júnior';

console.log(`${Name} - ${typeof Name}`);
console.log(`${Age} - ${typeof Age}`)
console.log(`${Occupation} - ${typeof Occupation}`)
```

Compilando:

```
npm run-script build src/basico.ts

> typescript@1.0.0 build
> tsc src/basico.ts
```

Executando após compilado:

```
node dist/basico.js
Aristóteles - string
22 - number
Desenvolvedor(a) Júnior - string
```

Type e Interface:

Além dos tipos básicos do JavaScript, em TypeScript podemos também definir types e interfaces que nos permitem expandir o leque de informações a serem validadas.

```typescript
export type tOccupation = 'Analista' | 'Desenvolvedor(a) Júnior' | 'Desenvolvedor(a) Pleno' | 'Desenvolvedor(a) Sênior';

1 usage
export interface iEmployee {
    Name: string;
    Age: number;
    Occupation: tOccupation;
}

const Employee: iEmployee = {
    Name: 'Aristóteles',
    Age: 25,
    Occupation: 'Desenvolvedor(a) Pleno'
};

console.log(Employee)
```

Compilando:

```
npm run-script build src/type_interface.ts

> typescript@1.0.0 build
> tsc src/type_interface.ts
```

Executando após compilado:

```
node dist/type_interface.js
{ Name: 'Aristóteles', Age: 25, Occupation: 'Desenvolvedor(a) Pleno' }
```

Classe:

Podemos também definir as verificações/validações nas classes.

```
import {tOccupation} from "./type_interface";

class cEmployee {
    private Name: string;
    private Age: number;
    private Occupation: tOccupation;
    constructor(Name: string, Age: number, Occupation: tOccupation) {
        this.setName(Name);
        this.setAge(Age);
        this.setOccupation(Occupation);
    }
    setName(Name: string): void {
        this.Name = Name;
    }
    getName(): string {
        return this.Name;
    }
    setAge(Age: number): void {
        this.Age = Age;
    }
    getAge(): number {
        return this.Age;
    }
    setOccupation(Occupation: tOccupation): void {
        this.Occupation = Occupation;
    }
    getOccupation(): tOccupation {
        return this.Occupation;
    }
}


const Employee = new cEmployee('Aristóteles', 30, 'Desenvolvedor(a) Sênior');

console.log(Employee)

Employee.setName(`${Employee.getName()} - Promovido`);
Employee.setAge(40);
Employee.setOccupation('Analista');

console.log(`Name: ${Employee.getName()}`)
console.log(`Age: ${Employee.getAge()}`)
console.log(`Occupation: ${Employee.getOccupation()}`)
```

Compilando:

```
npm run-script build src/classe.ts

> typescript@1.0.0 build
> tsc src/classe.ts
```

Executando após compilado:

```
node dist/classe.js
cEmployee {
  Name: 'Aristóteles',
  Age: 30,
  Occupation: 'Desenvolvedor(a) Sênior'
}
Name: Aristóteles - Promovido
Age: 40
Occupation: Analista
```

Forçando um erro:

Aqui forçamos um erro no código para mostrar que não será possível passar com o erro da fase de build.

```
const multiplier: number = 5;
for (let position: number = 0; position <= 100; position = `${(++position) * multiplier}`) {
    console.log(position);
}
```

```
npm run-script build src/com_erro.ts

> typescript@1.0.0 build
> tsc src/com_erro.ts

src/com_erro.ts:2:49 - error TS2322: Type 'string' is not assignable to type 'number'.

2 for (let position: number = 0; position <= 100; position = `${(++position) * multiplier}`) {
                                                  ~~~~~~~~


Found 1 error in src/com_erro.ts:2
```

TypeScript é:

A base sólida de uma construção.

Obrigado!

João Souza

Linkedin:
https://www.linkedin.com/in/jaoxico

GitHub:
https://github.com/jaoxico

ConFLOSS