TypeScript

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Vantagens e Desvantagens
- Tipagem Estática
- Escalabilidade
- Integração com JavaScript
- Codificação

Typescript

TypeScript é uma linguagem de programação de código aberto que é uma extensão do JavaScript. Ela adiciona recursos ao JavaScript, como a capacidade de definir tipos de dados para variáveis, funções e outros elementos do código.

Ao usar o TypeScript, os desenvolvedores podem escrever código mais robusto, com menos erros, já que o compilador do TypeScript pode detectar erros de digitação e outros problemas comuns antes da execução do código. Além disso, o TypeScript permite que os desenvolvedores escrevam código mais legível e escalável, o que é especialmente útil para projetos maiores e mais complexos.

Vantagens

- Tipagem estática
- Melhor escalabilidade
- Integração com Javascript
- Suporte para recursos modernos do ECMAScript

Tipagem Estática

O TypeScript permite que os desenvolvedores definam tipos de dados para variáveis, funções e outros elementos do código. Com a tipagem estática, é possível detectar erros de digitação e outros problemas comuns antes da execução do código. Isso resulta em código mais robusto e menos propenso a erros.

Escalabilidade

Com a tipagem estática, o TypeScript torna o código mais legível e escalável, especialmente para projetos maiores e mais complexos. Isso ajuda os desenvolvedores a trabalharem em equipe de maneira mais eficiente, já que cada um pode entender melhor o código escrito pelos outros.

Integração com Javascript

Como o TypeScript é uma extensão do JavaScript, é possível utilizar bibliotecas e frameworks JavaScript existentes em projetos TypeScript. Isso permite que os desenvolvedores aproveitem a vasta quantidade de recursos disponíveis na comunidade JavaScript.

Recursos modernos do ECMAScript

O TypeScript é compatível com as versões mais recentes do ECMAScript, o padrão que define a linguagem JavaScript. Isso significa que os desenvolvedores podem utilizar recursos modernos do JavaScript, como classes, arrow functions e async/await, mesmo em navegadores mais antigos que não suportam esses recursos nativamente.

Desvantagens

- **Necessidade de compilação:** como o TS é uma linguagem compilada, é preciso compilar o código para JS antes de executá-lo.

- **Tamanho dos arquivos:** Como o TS inclui informações de tipagem em seus arquivos, eles podem ser maiores do que os arquivos JavaScript equivalentes.

- **Necessidade de atualização:** À medida que novas versões do TS são lançadas, pode ser necessário atualizar o código existente para compatibilidade com a nova versão.

JS x TS

A maior diferença entre o código JS e TS está na tipagem das variáveis, onde no TS é possível definir tipos estáticos para variáveis, argumentos e retornos de função, enquanto no JS a tipagem é dinâmica.

Além disso, **o TS é uma linguagem orientada a objetos e suporta recursos como interfaces e herança de classes, que não são nativos do JS**. No entanto, é importante lembrar que o JS também suporta programação orientada a objetos, mas com uma abordagem diferente da adotada pelo TS.

Código Javascript

Abaixo está um exemplo de uma função em Javascript que multiplica dois números enviados por parâmetro nas funções.

```
function divideNumbers(num1, num2) {
    if (num2 !== 0) {
      return num1 / num2;
    } else {
      return "Erro: o denominador não pode ser zero. ";
    }
}
```

Código Typescript

Abaixo está um exemplo da mesma função declarada no slide anterior mas seguindo as boas práticas do Typescript.

```typescript
function divideNumbers(num1: number, num2: number): number | string {
    if (num2 !== 0) {
      return num1 / num2;
    } else {
      return "Erro: o denominador não pode ser zero. ";
    }
}
```

Tipagem

O TypeScript oferece suporte a vários tipos, incluindo tipos primitivos (como number, string e boolean), tipos complexos (como arrays e objetos), tipos de função (incluindo parâmetros e valores de retorno) e muito mais. Além disso, o TypeScript permite a criação de interfaces e classes para definir a estrutura e o comportamento dos objetos em seu código.

A tipagem no TS é opcional, o que significa que você pode optar por usar ou não tipos em seu código. No entanto, é recomendável usar tipos em projetos grandes e complexos para melhorar a qualidade e a manutenção do código.

Interface

```
interface IAnimal {
    nome: string;
    come(): void;
}

...

const animal: IAnimal = {
    nome: 'Buzzy',
    come: () => {
        console.log('Animal comeu.')
    }
}
```

Exemplo de uso de interface.

Classe

```
class Animal {
    private nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }

    come() {
        console.log(`${this.nome} comeu.`)
    }
}

const animal = new Animal("Buzzy")
```

Exemplo de uso de classe.

Interface vs Classe

No TypeScript, a interface define a forma que um objeto deve ter e a classe implementa a lógica por trás desses objetos, como suas operações e comportamentos que um objeto pode executar. A interface é usada para especificar o contrato que uma classe deve seguir, enquanto a classe não pode ser usada como uma interface.

E conseguimos combinar o uso da classe com a interface para produzir melhores códigos, com a interface sempre sendo definida como um modelo de dados para a classe e consequentemente para todos os objetos instanciados a partir dela.

Classe com interface

```
interface IAnimal {
    nome: string;
    come(): void;
}
class Animal implements IAnimal {
    nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }
    come(): void {
        console.log(`${this.nome} comeu.`)
    }
}
```

Exemplo de uso de classe com uma implementação de interface.

POO

A **POO** é uma forma de programação que representa objetos no código, permitindo que eles interajam entre si e com o mundo exterior. São utilizados quatro conceitos principais: **encapsulamento** (esconder detalhes), **herança** (criar novos objetos a partir de outros), **polimorfismo** (usar métodos de maneira flexível) e **abstração** (simplificar objetos).

Herança

É a capacidade de criar novas classes a partir de classes existentes, herdando seus atributos e métodos. Na POO, a herança é utilizada para reutilizar código e definir uma hierarquia de classes.

Herança

```
class Animal {
    nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }
}
class Cachorro extends Animal {
    raca: string;
    constructor(nome: string, raca: string) {
        super(nome);
        this.raca = raca;
    }
}
const cachorro = new Cachorro("Buzzy", "Labrador")
```

Exemplo de uso de herança

Polimorfismo

É a capacidade de um objeto se comportar de diferentes formas em diferentes contextos. Na POO, o polimorfismo é utilizado para permitir que um objeto seja tratado de forma genérica, independente de sua classe específica, e que possa responder a mensagens de formas diferentes dependendo do contexto em que é utilizado.

Polimorfismo

```
class Animal {
    emitiuSom() {
        console.log("O animal está fazendo seu barulho específico.")
    }
}
class Cachorro extends Animal {
    emitiuSom() {
        console.log("O animal está latindo")
    }
}
const cachorro = new Cachorro()
cachorro.emitiuSom() // Saída: O animal está latindo
```

Exemplo de uso de polimorfismo

Encapsulamento

É a técnica de ocultar a complexidade do objeto, expondo apenas as informações e operações essenciais. Na POO, o encapsulamento é utilizado para proteger os dados e comportamentos de um objeto de acessos indevidos.

Encapsulamento

```
class Animal {
    private nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }
    public getNome(): string {
        return this.nome;
    }
    public setNome(nome: string): void {
        this.nome = nome;
    }
}
const animal = new Animal("Buzzy")
console.log(animal.getNome()) // Saída: Buzzy
```

Exemplo de uso do encapsulamento

Abstração

É a capacidade de extrair as características essenciais de um objeto e representá-las de forma simplificada. Na POO, a abstração é utilizada para modelar objetos que possuem características comuns.

Abstração

```
abstract class Animal {
    abstract fazerSom(): void;
}

class Cachorro extends Animal {
    fazerSom() {
        console.log("Au au");
    }
}
class Gato implements Animal {
    fazerSom() {
        console.log("Miau");
    }
}
```

Exemplo de uso de abstração

ECMAScript

ECMAScript (também conhecido como ES) é um padrão para linguagens de script baseadas em JavaScript. O **ECMAScript** define a sintaxe, estrutura e recursos da linguagem JavaScript, incluindo variáveis, funções, operadores, objetos e outros elementos. O ECMAScript é mantido pela Ecma International, uma organização que desenvolve e mantém padrões para tecnologias da informação e comunicação.

Referência: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>