Utilizando o `ts-node-dev`

O `ts-node-dev` combina o `ts-node`, que permite a execução de TypeScript diretamente no Node.js sem compilação prévia, com a capacidade de monitorar mudanças nos arquivos do projeto e reiniciar automaticamente. É uma ferramenta perfeita para desenvolvimento rápido, pois você não precisa recompilar seu código manualmente a cada mudança.

Instalação

Para começar a usar o `ts-node-dev`, primeiro você precisa instalá-lo. Como é uma ferramenta de desenvolvimento, é uma boa prática instalá-la como uma dependência de desenvolvimento em seu projeto:

```
npm install --save-dev ts-node-dev
```

Configuração

Com o `ts-node-dev` instalado, você pode configurá-lo para executar seu projeto. Adicione um script no seu `package.json` para facilitar a execução:

```
"scripts": {
    "start": "ts-node-dev --respawn --transpile-only ./index.ts"
}
```

- `-respawn`: garante que o processo seja reiniciado após cada alteração.
- `-transpile-only`: acelera a execução ao pular as verificações de tipo completas do TypeScript (que você pode querer em um ambiente de produção).

Executando o Projeto

Para rodar seu projeto com `ts-node-dev`, basta usar o comando:

```
npm run start
```

Isso inicia seu aplicativo e monitora qualquer alteração nos arquivos TypeScript, reiniciando automaticamente o servidor sempre que você salvar um arquivo. Isso é incrivelmente útil durante o desenvolvimento, pois você vê as alterações em tempo real sem a necessidade de reiniciar manualmente o servidor.

Agora que seu ambiente está configurado, você está pronto para mergulhar mais fundo no TypeScript!

3. Tipos Básicos em TypeScript

TypeScript traz uma camada adicional de segurança ao JavaScript através da tipagem estática. Isso significa que você define tipos para suas variáveis e funções, o que ajuda o compilador a detectar erros antes que seu código seja executado. Vamos explorar os tipos básicos que você usará na maioria dos projetos.

Tipos Primitivos

1. **Boolean**
   - **O tipo mais simples, `boolean`, representa um valor lógico: verdadeiro ou falso.**

```
let estaAtivo: boolean = true;
```

2. **Number**
   - **Assim como no JavaScript, todos os números em TypeScript são valores de ponto flutuante. TypeScript fornece suporte tanto para números inteiros quanto para valores de ponto flutuante.**

```
let total: number = 0;
let pi: number = 3.14159;
```

3. **String**
   - **Para textos e caracteres, use o tipo `string`. Você pode usar aspas simples (' '), aspas duplas (" ") ou crases (``) para strings que precisam de interpolação ou múltiplas linhas.**

```
let nome: string = "João";
let saudacao: string = `Olá, ${nome}!`;
```

4. **Array**
   - **Arrays podem ser escritos de duas formas: usando o tipo dos elementos seguido de `[ ]` ou usando um tipo de array genérico.**

```
let numeros: number[] = [1, 2, 3];
let frutas: Array<string> = ["maçã", "banana", "cereja"];
```

5. **Tuple**
    - **Os tuples permitem expressar um array com um número fixo de elementos cujos tipos são conhecidos, mas não precisam ser iguais.**

```
let endereco: [string, number] = ["Av. Paulista", 1578];
```

6. **Enum**
    - **Um `enum` permite definir um conjunto de constantes nomeadas. O TypeScript suporta numéricos e baseados em string.**

```
enum Cor {Vermelho, Verde, Azul};
let c: Cor = Cor.Verde;
```

7. **Any**
    - **Use `any` para capturar valores cujo tipo não é importante e sobre os quais você não quer realizar checagem de tipo. Ideal para migração de JavaScript para TypeScript.**

```
let variavelIndefinida: any = 4;
variavelIndefinida = "talvez uma string";
```

8. **Void, Null e Undefined**
    - **`void` é usado em funções que não retornam nada. `null` e `undefined` são subtipos de todos os outros tipos.**

```
function alerta(): void {
    alert("Esta é uma mensagem de alerta!");
}
```

Exercícios para Praticar

1. **Trabalhando com Tipos:**
    - **Crie variáveis para cada tipo básico e imprima-as.**
    - **Tente atribuir valores incorretos a essas variáveis para ver o que acontece.**
2. **Função com Tipos:**
    - **Escreva uma função que aceita um array de números e retorna a soma de todos os elementos.**
    - **Assegure-se de tipar tanto a entrada quanto a saída da função.**
3. **Enumerações:**
    - **Crie um enum para representar os dias da semana e use-o em uma função que imprime uma mensagem de acordo com o dia passado.**
4. **Tuplas:**
    - **Crie uma tupla que representa um produto (com nome e preço). Use essa tupla em uma função que imprime o nome e o preço do produto.**
5. **Any:**
    - **Crie uma variável do tipo `any` e atribua diferentes tipos de valores a ela. Note como o TypeScript não emite erros nesse caso.**
6. **Void, Null e Undefined:**
    - **Crie uma função que não retorna nada (`void`) e outra que retorna `undefined`. Compare as duas.**