TypeScript: Tipos Avançados e Union Types

Por Escola Dnc

Introdução

O TypeScript é uma linguagem que estende o JavaScript, adicionando tipagem estática opcional. Neste ebook, vamos explorar alguns dos recursos mais poderosos e flexíveis do TypeScript: tipos personalizados, union types e como eles podem melhorar significativamente a qualidade e a robustez do seu código.

Tipos Personalizados (Custom Types)

O que são tipos personalizados?

Os tipos personalizados no TypeScript permitem que você defina estruturas de dados complexas e específicas para o seu projeto. Eles vão além dos tipos primitivos, oferecendo maior flexibilidade e precisão na definição de dados.

Como criar um tipo personalizado

Para criar um tipo personalizado, usamos a palavra-chave `type`. Veja um exemplo:

```
type Identificacao = {  id: string,  setor: string,  ala: number}
```

Neste exemplo, criamos um tipo `Identificacao` que possui três propriedades: `id` (string), `setor` (string) e `ala` (number).

Utilizando tipos personalizados

Após definir um tipo personalizado, você pode usá-lo para tipar variáveis, parâmetros de funções ou retornos de funções:

```
const identificacao: Identificacao = {  id: "42",  setor: "norte",
ala: 25}console.log(identificacao); // Saída: { id: "42", setor:
"norte", ala: 25 }
```

Benefícios dos tipos personalizados:

- Melhor organização do código
- Maior clareza na estrutura dos dados

- Facilita a manutenção e evolução do código

Union Types

O que são Union Types?

Union Types permitem que uma variável, parâmetro ou retorno de função aceite mais de um tipo. Isso oferece flexibilidade enquanto mantém a segurança de tipos.

Sintaxe dos Union Types

Para criar um Union Type, usamos o operador `|` (pipe) entre os tipos possíveis:

```
type IdenticacaoFlexivel = string | number
```

Neste exemplo, `IdenticacaoFlexivel` pode ser tanto uma string quanto um número.

Exemplo prático de Union Types

```
const id: IdenticacaoFlexivel = "ABC123" // válidoconst outroId:
IdenticacaoFlexivel = 42 // também válido// const idInvalido:
IdenticacaoFlexivel = true // Isso geraria um erro
```

Vantagens dos Union Types:

- Maior flexibilidade na definição de tipos
- Redução de erros em tempo de execução
- Melhor representação de dados que podem ter múltiplos formatos

Tipos Enumerados com Union Types

Criando tipos enumerados

Union Types também podem ser usados para criar tipos enumerados, onde você lista explicitamente todos os valores possíveis:

```
type TipoSanguineo = "A+" | "A-" | "B+" | "B-" | "AB+" | "AB-" |
"O+" | "O-"
```

Benefícios dos tipos enumerados

- **Autocompletar**: IDEs podem sugerir os valores válidos
- **Prevenção de erros**: O compilador impede o uso de valores não listados
- **Documentação implícita**: O tipo em si documenta os valores possíveis

Exemplo de uso

```
function exibirTipoSanguineo(tipo: TipoSanguineo): void {
console.log(`O tipo sanguíneo é:
${tipo}`);}exibirTipoSanguineo("A+"); // Válido//
exibirTipoSanguineo("C+"); // Erro: Argumento inválido
```

Aplicações Práticas

Em funções

Union Types e tipos personalizados são particularmente úteis em assinaturas de funções:

```
function processarIdentificacao(id: string | number): void {  // Lógica para processar tanto strings quanto números}processarIdentificacao("ABC123");processarIdentificacao(42);
```

Em interfaces e classes

Você pode usar tipos personalizados e Union Types em interfaces e classes:

```
interface Usuario {  id: string | number;  tipo: "admin" | "regular" | "convidado";  perfil: PerfilUsuario; // Tipo personalizado}class SistemaAutenticacao {  autenticar(usuario: Usuario): boolean {    // Lógica de autenticação  }}
```

Melhorando a segurança do código

Ao usar tipos mais específicos, você reduz a chance de erros e melhora a qualidade geral do código:

```
type StatusPedido = "pendente" | "aprovado" | "rejeitado";function processarPedido(status: StatusPedido): void {  switch (status) {  case "pendente":      // Lógica para pedidos pendentes      break;
```

```
case "aprovado":      // Lógica para pedidos aprovados      break;
case "rejeitado":      // Lógica para pedidos rejeitados
break;    default:      // TypeScript garante que todos os casos
foram cobertos      const _exhaustiveCheck: never = status;  }}
```

Conclusão

Tipos personalizados e Union Types são ferramentas poderosas no arsenal do TypeScript. Eles permitem:

- Criar estruturas de dados complexas e específicas
- Oferecer flexibilidade com segurança de tipos
- Melhorar a legibilidade e manutenção do código
- Prevenir erros comuns em tempo de compilação

Ao dominar esses conceitos, você estará bem equipado para escrever código TypeScript mais robusto, flexível e fácil de manter. Lembre-se de que a tipagem forte do TypeScript não é uma limitação, mas sim uma ferramenta para tornar seu código mais confiável e auto-documentado.

```
Dica final: Use tipos personalizados e Union Types sempre que
precisar representar dados complexos ou variáveis que podem ter
múltiplos tipos. Isso tornará seu código mais expressivo e menos
propenso a erros.
```

Com prática e experiência, você descobrirá que esses recursos do TypeScript se tornarão indispensáveis em seu fluxo de trabalho de desenvolvimento, ajudando a criar aplicações mais robustas e fáceis de manter.