TypeScript Utility Types: Kapsamlı Kılavuz

Giriş

TypeScript utility type'ları, var olan tipleri dönüştürmek, genişletmek ve manipüle etmek için kullanılan yerleşik tip araçlarıdır. Bu kılavuz, tüm önemli utility type'ları detaylı bir şekilde açıklayacaktır.

1. `Partial<Type>`

Verilen bir tipin tüm özelliklerini isteğe bağlı (optional) hale getirir.

```
interface User {
    ad: string;
    soyad: string;
    yas: number;
}

// Tüm alanlar isteğe bağlı olur
type PartialUser = Partial<User>;
// Çıktı: { ad?: string; soyad?: string; yas?: number; }
```

2. `Readonly<Type>`

Tipin tüm özelliklerini salt okunur (immutable) yapar.

```
interface Settings {
    tema: string;
    dil: string;
}

// Değiştirilemeyen bir tip
type FixedSettings = Readonly<Settings>;
// Çıktı: { readonly tema: string; readonly dil: string; }
```

3. `Pick<Type, Keys>`

Belirli özellikleri seçerek yeni bir tip oluşturur.

```
interface Personal {
    ad: string;
    soyad: string;
    departman: string;
    maas: number;
}

// Sadece ad ve soyad özellikleri olan yeni bir tip
```

```ts
type PersonalBasic = Pick<Personal, 'ad' | 'soyad'>;
// Çıktı: { ad: string; soyad: string; }
```

4. Omit<Type, Keys>

Belirli özellikleri çıkararak yeni bir tip oluşturur.

```ts
interface Product {
    id: number;
    ad: string;
    fiyat: number;
    stok: number;
}

// id ve stok çıkarılmış yeni tip
type ShowProduct = Omit<Product, 'id' | 'stok'>;
// Çıktı: { ad: string; fiyat: number; }
```

5. Record<Keys, Type>

Belirli anahtarlar ve değer tipi ile nesne tipi oluşturur.

```ts
// Dil kodları ve çevirileri için tip
type Language = Record<'tr' | 'en' | 'fr', string>;
// Çıktı: { tr: string; en: string; fr: string; }
```

6. Required<Type>

Tüm özellikleri zorunlu (non-optional) hale getirir.

```ts
interface Settings {
    tema?: string;
    dil?: string;
}

// Tema ve dil artık zorunlu
type MandatorySettings = Required<Settings>;
// Çıktı: { tema: string; dil: string; }
```

7. Exclude<Type, ExcludedUnion>

Birleşim tipinden belirli tipleri çıkarır.

```ts
type Color = 'kirmizi' | 'mavi' | 'yesil';
type RemovedColors = Exclude<Color, 'mavi'>;
// Çıktı: 'kirmizi' | 'yesil'
```

8. Extract<Type, Union>

Birleşim tipinden belirli tipleri seçer.

```
type Number = number | string | boolean;
type OnlyNumber = Extract<Number, number>;
// Çıktı: number
```

9. NonNullable<Type>

Null ve undefined tiplerini çıkarır.

```
type CurrentType = string | number | null | undefined;
type NullSiz = NonNullable<CurrentType>;
// Çıktı: string | number
```

10. ReturnType<Type>

Fonksiyon tipinin dönüş tipini alır.

```
function createUser() {
    return { id: 1, ad: 'John' };
}

type UserType = ReturnType<typeof createUser>;
// Çıktı: { id: number; ad: string; }
```

11. Parameters<Type>

Fonksiyon tipinin parametre tiplerini bir tuple olarak alır.

```
function addition(x: number, y: number) {
    return x + y;
}

type AdditionParameters = Parameters<typeof addition>;
// Çıktı: [number, number]
```

12. ConstructorParameters<Type>

Sınıf constructor'ının parametre tiplerini alır.

```
class User {
    constructor(ad: string, yas: number) {}
}

type UserConstructorParameters =
    ConstructorParameters<typeof User>;
// Çıktı: [string, number]
```

13. InstanceType<Type>

Sınıf constructor'ının örnek tipini alır.

```
class Account {
    bakiye: number = 0;
    yatir(miktar: number) {}
}

type AccountExample = InstanceType<typeof Account>;
// Çıktı: Account sınıfının örnek tipi
```

Gelişmiş Kullanım Örneği

```
interface ApiResponse<T> {
    veri: T;
    hata?: string;
}

type ResponseOptimization<T> =
    Readonly<Partial<ApiResponse<T>>>;

// Esnek ve güvenli bir tip tanımı
type UserResponse = ResponseOptimization<User>;
```

Sonuç

TypeScript utility type'ları, statik tip güvenliği sağlayarak kodunuzu daha net, bakımı kolay ve hata ayıklaması daha kolay hale getirir. Her bir utility type, farklı senaryolarda tipleri dönüştürmek ve manipüle etmek için güçlü araçlar sunar.

Öneriler

- Utility type'ları birlikte kullanarak daha karmaşık tip dönüşümleri yapabilirsiniz.
- Kod tekrarını önlemek ve tip güvenliğini artırmak için utility type'lardan yararlanın.
- Projelerinizde tiplerin esnek ve net olması için bu araçları kullanın.

Performans Notu

Utility type'lar derleme zamanında çalışır ve çalışma zamanında herhangi bir performans yükü getirmez.