A propriedade `transform` do decorador `@Input` no Angular é uma funcionalidade útil introduzida para modificar os valores recebidos de um componente pai antes que sejam armazenados na propriedade do componente filho. Isso permite transformar ou validar os dados recebidos diretamente no nível da entrada.

Aqui estão exemplos detalhados para mostrar como utilizar a propriedade `transform` do `@Input` e suas várias possibilidades:

---

1. Exemplo básico com transformação de string

Transformar uma string recebida para letras maiúsculas.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-greeting',
  template: `<p>Mensagem: {{ greeting }}</p>`,
})
export class GreetingComponent {
  @Input({ transform: (value: string) => value.toUpperCase() })
  greeting: string = ''; // A string será automaticamente transformada para letras maiúsculas
}
```

Uso no componente pai:

```
<app-greeting [greeting]="'Olá, mundo!'"></app-greeting>
```

Saída renderizada:

```
Mensagem: OLÁ, MUNDO!
```

---

2. Validação de entrada

Validar e garantir que um número seja positivo. Caso contrário, usar um valor padrão.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-price',
  template: `<p>Preço: R$ {{ price }}</p>`,
})
export class PriceComponent {
  @Input({ transform: (value: number) => (value > 0 ? value : 10) })
  price: number = 0; // Se o valor for inválido ou negativo, será ajustado para
10
}
```

Uso no componente pai:

```
<app-price [price]="-50"></app-price>
```

Saída renderizada:

```
Preço: R$ 10
```

---

3. Aplicando múltiplas transformações

Modificar uma string para remover espaços extras e adicionar um sufixo.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-title',
  template: `<h1>{{ title }}</h1>`,
})
export class TitleComponent {
  @Input({
    transform: (value: string) =>
      `${value.trim()} (Edição Especial)`,
  })
  title: string = '';
}
```

Uso no componente pai:

<app-title [title]="' Promoção Incrível '"></app-title>

Saída renderizada:

```
Promoção Incrível (Edição Especial)
```

---

4. Manipulação de objetos

Transformar um objeto recebido no formato esperado.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-user-card',
  template: `<p>Nome: {{ user.name }}</p>
             <p>Idade: {{ user.age }}</p>`,
})
export class UserCardComponent {
  @Input({
    transform: (value: any) => ({
      name: value?.name || 'Desconhecido',
      age: value?.age || 'Não informado',
    }),
  })
  user: { name: string; age: string } = { name: '', age: '' };
}
```

Uso no componente pai:

<app-user-card [user]="{ name: 'Jessica' }"></app-user-card>

Saída renderizada:

```
Nome: Jessica Idade: Não informado
```

---

5. Conversão de valores primitivos

Converter um valor numérico em uma string formatada como moeda.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-currency-display',
  template: `<p>{{ amount }}</p>`,
})
export class CurrencyDisplayComponent {
  @Input({
    transform: (value: number) =>
      `R$ ${value.toFixed(2).replace('.', ',')}`,
  })
  amount: string = '';
}
```

Uso no componente pai:

```
<app-currency-display [amount]="1500.5"></app-currency-display>
```

Saída renderizada:

```
R$ 1500,50
```

---

6. Combinação com `booleanAttribute`

Usar a transformação para converter strings em booleanos automaticamente.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-availability',
  template: `<p>Status: {{ available ? 'Disponível' : 'Indisponível' }}</p>`,
})
export class AvailabilityComponent {
  @Input({ transform: (value: string) => value === 'true' })
  available: boolean = false;
}
```

Uso no componente pai:

<app-availability [available]="'true'"></app-availability>

Saída renderizada:

```
Status: Disponível
```

---

7. Combinação com lógica de negócios

Transformar e calcular valores complexos, como aplicar um desconto.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-discount',
  template: `<p>Preço com desconto: R$ {{ discountedPrice }}</p>`,
})
export class DiscountComponent {
  @Input({
    transform: (value: number) =>
      value - value * 0.1, // Aplica 10% de desconto
  })
  discountedPrice: number = 0;
}
```

Uso no componente pai:

<app-discount [discountedPrice]="200"></app-discount>

Saída renderizada:

```
Preço com desconto: R$ 180
```

---

8. Transformação com manipulação assíncrona

Embora o `transform` de `@Input` não suporte operações assíncronas diretamente, você pode utilizar essa funcionalidade para disparar funções que retornam valores processados de forma síncrona.

Pontos importantes

- **Validação e limpeza de dados:** Use transformações para garantir que os dados recebidos estejam no formato correto.
- **Aplicação de regras de negócio:** Realize cálculos ou ajustes com base em critérios pré-definidos.
- **Simplificação de componentes filhos:** A lógica de transformação no nível de entrada reduz a necessidade de lidar com dados inconsistentes dentro do componente.

Essa flexibilidade ajuda a manter os componentes simples, reutilizáveis e robustos.