O decorator `@Input` é usado em Angular para permitir que um componente receba dados de seu componente pai (ou de outro componente que o utiliza). Ele serve para criar propriedades que podem ser configuradas externamente no momento em que o componente é instanciado em um template.

Como Funciona o `@Input`

Quando você usa o `@Input` em uma propriedade de um componente, você está essencialmente dizendo ao Angular que aquela propriedade pode ser preenchida dinamicamente por valores que vêm de fora do componente.

Exemplo Básico

Componente Filho (`child.component.ts`)

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

@Component({
  selector: 'app-child',
  template: `<p>Olá, meu nome é {{name}} e eu tenho {{age}} anos.</p>`
})
export class ChildComponent {
  @Input() name: string = ''; // Propriedade configurável externamente
  @Input() age: number = 0;  // Outra propriedade configurável
}
```

Componente Pai (`parent.component.html`)

```
<app-child [name]="'Jessica'" [age]="30"></app-child>
<app-child [name]="'Carlos'" [age]="25"></app-child>
```

Saída no Navegador

Olá, meu nome é Jessica e eu tenho 30 anos.
Olá, meu nome é Carlos e eu tenho 25 anos.

Usando Alias no `@Input`

Você pode alterar o nome usado no template para configurar a propriedade, diferente do nome da variável no componente.

Componente Filho (`child.component.ts`)

@Input('userName') name: string = '';

Componente Pai (`parent.component.html`)

<app-child [userName]="'Maria'"></app-child>

Aqui, no template, usamos `[userName]`, mas no componente filho, a propriedade ainda é `name`.

Validação com `@Input` Obrigatório

Para garantir que uma propriedade decorada com `@Input` seja obrigatória, podemos usar o modificador `required`.

Componente Filho (`child.component.ts`)

@Input({ required: true }) name!: string;

Se a propriedade não for passada pelo componente pai, o Angular emitirá um erro.

Propriedades Complexas

O `@Input` também suporta objetos e arrays. Você pode passar estruturas complexas para os componentes.

Componente Filho

@Input() user: { name: string; age: number } = { name: '', age: 0 };

Componente Pai

<app-child [user]="{ name: 'João', age: 45 }"></app-child>

Template do Filho

```
<p>Nome: {{user.name}}</p>
<p>Idade: {{user.age}}</p>
```

Exemplo Avançado: Comunicação entre Componentes

Estrutura

- `app.component.html`: Componente Pai
- `child.component.ts`: Componente Filho

Código do Pai

```
<app-child [data]="message"></app-child>
<button (click)="updateMessage()">Atualizar Mensagem</button>

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  message: string = 'Mensagem inicial';

  updateMessage() {
    this.message = 'Mensagem atualizada!';
  }
}
```

Código do Filho

```
@Component({
  selector: 'app-child',
  template: `<p>Mensagem recebida: {{data}}</p>`,
})
export class ChildComponent {
  @Input() data: string = ''; // Propriedade recebe o valor do Pai
}
```

Saída no Navegador

- Antes de clicar no botão: `Mensagem recebida: Mensagem inicial`
- Depois de clicar no botão: `Mensagem recebida: Mensagem atualizada!`

Resumo do que Pode Ser Feito com `@Input`

1. **Tipos Primitivos**: Passar strings, números ou booleanos.
2. **Estruturas Complexas**: Passar objetos, arrays ou funções.
3. **Aliases**: Usar nomes diferentes para a propriedade no componente pai e no filho.
4. **Two-Way Binding (com `@Output`)**: Junto com `@Output`, permite comunicação bidirecional entre componentes.
5. **Validação Obrigatória**: Garantir que o dado será sempre fornecido.
6. **Acesso Dinâmico**: Permite atualizar as propriedades dinamicamente em resposta a eventos.

O `@Input` é essencial em Angular para criar componentes modulares e reutilizáveis, permitindo passar dados de forma clara e bem estruturada.