O que é o Two-Way Data Binding no Angular?

O **Two-Way Data Binding** no Angular conecta dados entre o modelo (component) e a visualização (view) de forma sincronizada. Isso significa que:

1. **Se o modelo for atualizado**, a visualização é atualizada automaticamente.
2. **Se o usuário interagir com a visualização** (como digitar em um campo de entrada), o modelo será atualizado automaticamente.

Isso é conseguido usando a diretiva `[(ngModel)]`.

Como funciona?

O Two-Way Data Binding combina:

- **Property Binding:** `[value]="property"` → Liga uma propriedade do componente ao valor de um elemento HTML.
- **Event Binding:** `(input)="eventHandler($event)"` → Liga um evento ao componente para que ele reaja às mudanças no elemento HTML.

No Angular, `[(ngModel)]` combina as duas ligações acima para simplificar a sincronização entre a visualização e o modelo.

Exemplo Básico

Código do Template (HTML):

```
<h1>Formulário de Cadastro</h1>
<!-- Input com Two-Way Data Binding -->
<label for="name">Digite seu nome:</label>
<input id="name" type="text" [(ngModel)]="name">

<!-- Exibe o valor atualizado da variável 'name' -->
<p>Nome digitado: {{ name }}</p>
```

Código do Componente (TypeScript):

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  name: string = ''; // Variável do modelo que será vinculada ao input
}
```

O que pode ser feito com Two-Way Data Binding?

1. **Inputs de Texto:**
    - Sincronizar entradas de texto com variáveis do componente.

```
<input type="text" [(ngModel)]="username">
<p>Usuário: {{ username }}</p>
```

2. **Checkbox:**
    - Sincronizar o estado de seleção de um checkbox.

```
<input type="checkbox" [(ngModel)]="isChecked">
<p>Checkbox está {{ isChecked ? 'marcado' : 'desmarcado' }}</p>
```

3. **Radio Buttons:**
    - Sincronizar o valor selecionado.

```
<label>
  <input type="radio" name="gender" [(ngModel)]="gender"
value="Masculino"> Masculino
</label>
<label>
  <input type="radio" name="gender" [(ngModel)]="gender" value="Feminino">
Feminino
</label>
<p>Gênero selecionado: {{ gender }}</p>
```

4. **Select (Dropdown):**

    - Sincronizar opções selecionadas de um dropdown.

```
<select [(ngModel)]="selectedOption">
  <option *ngFor="let option of options" [value]="option">{{ option }}</option>
</select>
<p>Opção selecionada: {{ selectedOption }}</p>
 options = ['Opção 1', 'Opção 2', 'Opção 3'];
selectedOption = 'Opção 1';
```

5. **Forms Dinâmicos:**

    - Sincronizar valores de múltiplos inputs em tempo real.

```
<form>
  <label for="email">Email:</label>
  <input id="email" type="email" [(ngModel)]="formData.email" name="email">

  <label for="password">Senha:</label>
  <input id="password" type="password" [(ngModel)]="formData.password"
name="password">
</form>
<p>Form Data: {{ formData | json }}</p>
 formData = { email: '', password: '' };
```

6.

Vantagens do Two-Way Data Binding

1. **Simplicidade:**
    - Reduz código repetitivo para atualizar a visualização e o modelo.
2. **Sincronização Automática:**
    - Qualquer mudança no componente reflete na visualização e vice-versa.
3. **Ideal para Formulários:**
    - Simplifica o trabalho com formulários dinâmicos e campos de entrada.

Limitações do Two-Way Data Binding

1. **Requer FormsModule:**
   - O módulo `FormsModule` precisa ser importado para habilitar `[(ngModel)]`.
2. **Performance:**
   - Em aplicações muito grandes, pode afetar o desempenho devido à constante verificação de mudanças.

Alternativa: Reactive Forms

Se o Two-Way Data Binding parecer limitado, você pode usar Reactive Forms no Angular para maior controle e validação de formulários.

```
import { FormControl } from '@angular/forms';

@Component({
  //...
})
export class AppComponent {
  name = new FormControl(''); // Cria um controle reativo para o input
}
```

Resumo

O Two-Way Data Binding simplifica a interação entre o modelo e a visualização, especialmente em formulários. Ele é fácil de implementar e ideal para campos de entrada simples. No entanto, para cenários mais complexos, como validações avançadas, considere usar Reactive Forms.