Bindings in vue

Bindings are a way to link data from the component's state to the DOM. This allows you to dynamically update the UI based on the state of the component. Vue provides several ways to bind data to elements, such as v-bind, v-model, and event handling.

Most commonly used bindings in Vue 3

1. v-bind:

Binding HTML Attributes and Props. v-bind is used to dynamically bind an attribute or a prop to an element. It can bind to any HTML attribute or component prop.

Think of it like a "link" between your data and HTML attributes. Use it whenever you want to change something like class, title, or other attributes based on data.

Example (Binding to HTML attributes):

```
<template>
  <button v-bind:title="buttonTitle">Hover over me</button>
</template>

<script setup>
const buttonTitle = 'This is a button'
</script>
```

In this example, the title attribute of the button will be dynamically updated based on the buttonTitle data.

v-bind:title becomes :title, which is the shorthand syntax and more commonly used

```
<button :title="buttonTitle">Hover over me</button>
```

Example (Binding to Component Props):

```
<template>
  <ChildComponent :message="parentMessage" />
</template>

<script setup>
import ChildComponent from './ChildComponent.vue'

const parentMessage = 'Hello from parent!'
</script>
```

2. v-model

Two-Way Binding (Input, Select, Textarea, etc.). v-model creates a two-way binding between the value of an input element and the data. It allows the component state to be updated automatically when the user interacts with the input element.

"v-model" is like talking to your input elements. Changes are reflected both ways.

Example (Text Input Binding):

```
<template>
  <input v-model="textInput" placeholder="Type something" />
  <p>You typed: {{ textInput }}</p>
</template>

<script>
export default {
  data() {
    return {
      textInput: ""
    }
  }
}
</script>
```

In this example, v-model binds the textInput data to the value of the input field, so when the user types in the input, the textInput data will be updated automatically.

Example (Binding with Custom Components):

```
<template>
  <ChildComponent v-model="isActive" />
</template>

<script setup>
import { ref } from 'vue'
import ChildComponent from './ChildComponent.vue'

const isActive = ref(false)
</script>
```

In this case, v-model binds the isActive variable to the child component's state.

3. v-for: Looping Over Data (Dynamic Rendering) v-for is used to loop over an array or object and render a list of elements dynamically.

"v-for" is like a "repeater". It repeats the elements as many times as needed based on the data.

Example (Looping Over an Array):

```
<template>
  <ul>
    <li v-for="(item, index) in items" :key="index">{{ item }}</li>
  </ul>
</template>

<script setup>
const items = ['Apple', 'Banana', 'Cherry']
</script>
```

This renders an unordered list with each item from the items array.

4. v-if, v-else-if, v-else: Conditional Rendering

v-if is used to conditionally render elements or components in the DOM based on the value of a data property.

"v-if" is like "choosing" based on conditions, showing one thing if true, and something else if false.

Example (Conditional Rendering):

```
<template>
  <div>
    <p v-if="isLoggedIn">Welcome, User!</p>
    <p v-else>Please log in</p>
  </div>
</template>

<script setup>
import { ref } from 'vue'

const isLoggedIn = ref(false)
</script>
```

If isLoggedIn is true, the "Welcome, User!" message is displayed. If it's false, the "Please log in" message appears.

5. v-on: Event Handling

v-on is used to listen for DOM events and call methods or run code in response.

"v-on" is like a "listener". It listens for actions, such as clicks or keypresses, and responds.

Example (Event Binding):

```
<template>
  <button @click="incrementCount">Click me</button>
  <p>Count: {{ count }}</p>
</template>
```

```
<script setup>
import { ref } from 'vue'

const count = ref(0)

function incrementCount() {
  count.value++
}
</script>
```

In this example, clicking the button will trigger the incrementCount method, which updates the count data.

Short Syntax for v-on: You can also use the shorthand @ for v-on:

```
<button @click="incrementCount">Click me</button>
```

6. :class: Dynamically Bind CSS Classes

:class is used to dynamically bind CSS classes to an element based on a condition or object.

":class" is like a "toggle switch". It switches classes on/off based on conditions.

Example (Class Binding):

```
<template>
  <button :class="{ active: isActive, 'btn-primary': isPrimary }">Click me</button>
</template>

<script setup>
import { ref } from 'vue'

const isActive = ref(true)
const isPrimary = ref(false)
</script>
```

In this example, the button will have the active class if isActive is true and the btn-primary class if isPrimary is true.

7. :style: Dynamically Bind Inline Styles

:style is used to dynamically apply inline styles to an element.

":style" is like "painting". It applies styles directly on the element based on data.

Example (Style Binding):

```
<template>
  <div :style="{ color: textColor, fontSize: fontSize + 'px' }">
    This is some text with dynamic styles.
  </div>
</template>

<script setup>
import { ref } from 'vue'

const textColor = ref('blue')
const fontSize = ref(20)
</script>
```

In this example, the text will be styled with the color blue and font size 20px dynamically.

NOTE :

```
| v-bind   | "Link"          |
| v-model  | "Talk"          |
| v-for    | "Repeater"      |
| v-if     | "Choose"        |
| v-on     | "Listener"      |
| :class   | "Toggle Switch"|
| :style   | "Painting"      |
```