Svelte

Svelte è un framework di sviluppo web moderno che si differenzia da altri framework come Angular, React e Vue per il modo in cui gestisce la creazione di interfacce utente dinamiche. Invece di essere un framework basato su componenti come gli altri, Svelte è un compilatore che converte il codice scritto in un linguaggio simile a JavaScript in codice JavaScript efficiente che manipola direttamente il DOM.

Ecco alcuni punti chiave su Svelte:

Sintassi dichiarativa:
Svelte utilizza una sintassi dichiarativa simile a JavaScript per definire componenti, gestire lo stato e manipolare il DOM. Tuttavia, il codice scritto in Svelte si traduce in codice JavaScript ottimizzato che viene eseguito direttamente dal browser.

```
<!-- App.svelte -->
<script>
  let name = 'world';
</script>

<h1>Hello, {name}!</h1>
```

Reattività integrata:
Svelte offre un'esperienza di sviluppo reattiva senza l'uso esplicito di concetti come "stati" o "proprietà". Le variabili all'interno dei componenti possono essere reattive di default: quando vengono aggiornate, Svelte si occupa automaticamente di aggiornare il DOM per riflettere i cambiamenti.

```
<!-- ReactiveVariable.svelte -->
<script>
  let count = 0;
```

```
  function increment() {
    count += 1;
  }
</script>

<button on:click={increment}>
  Clicked {count} times
</button>
```

Zero bundle-runtime:
A differenza di altri framework, Svelte non richiede alcun codice di runtime per funzionare. Il codice generato è estremamente ottimizzato e compatto, il che significa che le applicazioni Svelte tendono ad avere dimensioni più piccole rispetto ad applicazioni simili sviluppate con altri framework.

```
// Codice JavaScript generato da Svelte
let count = 0;

function increment() {
  count += 1;
}
```

Animazioni native:
Svelte supporta animazioni native tramite la dichiarazione di transizioni direttamente nel markup, senza bisogno di librerie esterne o di complessi hook di lifecycle.

```
<!-- Fade.svelte -->
<script>
  let visible = false;

  function toggle() {
    visible = !visible;
```

```
  }
</script>

<button on:click={toggle}>
  Toggle Visibility
</button>

{#if visible}
  <div transition:fade>
    This content fades in and out
  </div>
{/if}

<style>
  div {
    transition: opacity 0.5s;
  }
</style>
```

CSS encapsulation:
Svelte offre un sistema di stile che fornisce l'incapsulamento CSS senza l'uso di shadow DOM, rendendo più semplice e intuitivo lo styling dei componenti.

```
<!-- StyledComponent.svelte -->
<style>
  .container {
    background-color: lightblue;
    padding: 20px;
    border-radius: 5px;
  }
</style>

<div class="container">
```

```
  This content is styled with CSS encapsulation
</div>
```

Componenti "single file":
I componenti in Svelte possono essere scritti all'interno di un unico file che include template, script e stili, rendendo la struttura dei file più organizzata e la sintassi più concisa.

```
<!-- MyComponent.svelte -->
<script>
  let name = 'world';
</script>

<style>
  h1 {
    color: green;
  }
</style>

<h1>Hello, {name}!</h1>
```

Ampia compatibilità:
Svelte può essere utilizzato per sviluppare sia applicazioni web che applicazioni native attraverso framework come SvelteKit per applicazioni web e Svelte Native per applicazioni mobile.

```
npx degit sveltejs/template my-svelte-app
cd my-svelte-app
npm install
npm run dev
```

Componenti

In Svelte, i componenti sono costruiti per essere riutilizzabili e modulari. Ogni componente Svelte è definito all'interno di un file `.svelte`, che include il markup HTML, il codice JavaScript e gli stili CSS correlati al componente stesso. Ad esempio:

```
<!-- Button.svelte -->
<script>
  export let text = 'Click me';
</script>

<button>{text}</button>
```

Composizione di componenti

La composizione di componenti in Svelte è semplice e intuitiva. Puoi includere un componente all'interno di un altro componente utilizzando la sua sintassi di tag. Ad esempio:

```
<!-- App.svelte -->
<script>
  import Button from './Button.svelte';
</script>

<Button text="Click me"></Button>
```

Reattività

In Svelte, la reattività è intrinseca. Quando le variabili di stato cambiano, Svelte si occupa automaticamente di aggiornare il DOM per riflettere questi cambiamenti. Ad esempio:

```svelte
<!-- ReactiveVariable.svelte -->
<script>
  let count = 0;

  function increment() {
    count += 1;
  }
</script>

<button on:click={increment}>
  Clicked {count} times
</button>
```

In questo esempio, ogni volta che il pulsante viene cliccato, la variabile `count` viene incrementata e il valore aggiornato viene riflesso immediatamente nel DOM.

Dispatch di eventi dal componente

Puoi dispatchare eventi personalizzati da un componente Svelte utilizzando il metodo `dispatch` e catturarli in un componente genitore. Ad esempio:

```svelte
<!-- Child.svelte -->
<script>
  import { createEventDispatcher } from 'svelte';

  const dispatch = createEventDispatcher();

  function handleClick() {
    dispatch('customEvent', { detail: 'Hello from Child' });
  }
</script>

<button on:click={handleClick}>
```

```
  Click me
</button>
```

```
<!-- Parent.svelte -->
<script>
  function handleCustomEvent(event) {
    console.log(event.detail); // Output: "Hello from Child"
  }
</script>

<Child on:customEvent={handleCustomEvent}></Child>
```

Sincronizzare le variabili

Puoi sincronizzare le variabili tra componenti genitore e figlio utilizzando le proprietà. Quando una variabile viene aggiornata nel componente genitore, l'aggiornamento viene automaticamente riflesso nel componente figlio. Ad esempio:

```
<!-- Parent.svelte -->
<script>
  let name = 'Alice';
</script>

<Child bind:name={name}></Child>
```

```
<!-- Child.svelte -->
<script>
  export let name;
</script>

<p>Hello, {name}</p>
```

In questo caso, la variabile `name` nel componente `Child` è sincronizzata con la variabile `name` nel componente `Parent`. Quando `name` nel componente `Parent` cambia, l'aggiornamento viene automaticamente riflesso nel componente `Child`.

Passare stato ad un componente

Puoi passare lo stato a un componente Svelte utilizzando le proprietà (props). Ad esempio:

```
<!-- Parent.svelte -->
<script>
  let message = 'Hello from Parent';
</script>

<Child message={message}></Child>
```

```
<!-- Child.svelte -->
<script>
  export let message;
</script>

<p>{message}</p>
```

In questo esempio, `message` viene passato al componente `Child` come proprietà.

Spread Operator

Puoi utilizzare lo spread operator (`...`) per passare un oggetto di proprietà a un componente Svelte. Ad esempio:

```
<!-- App.svelte -->
<script>
  let user = { name: 'Alice', age: 30 };
</script>
```

```
<UserInfo {...user}></UserInfo>
```

In questo modo, tutte le proprietà dell'oggetto `user` vengono passate al componente `UserInfo` come singole proprietà.

Componenti e CSS

Puoi stili i componenti Svelte direttamente all'interno del file `.svelte` utilizzando blocchi `<style>`. Ad esempio:

```
<!-- Button.svelte -->
<style>
  button {
    background-color: blue;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
  }
</style>

<button>{text}</button>
```

Rendering condizionale

Puoi eseguire il rendering condizionale di elementi in base a una condizione utilizzando le direttive `if` e `else`. Ad esempio:

```
{#if isLoggedIn}
  <p>Welcome, {username}!</p>
{:else}
```

```
  <p>Please log in</p>
{/if}
```

In questo esempio, il paragrafo di benvenuto viene visualizzato solo se la variabile `isLoggedIn` è true.

Iterare su una collezione

Puoi iterare su una collezione di elementi utilizzando la direttiva `each`. Ad esempio:

```
<ul>
  {#each items as item}
    <li>{item}</li>
  {/each}
</ul>
```

In questo esempio, ogni elemento nell'array `items` viene visualizzato come un elemento della lista.

Blocchi Await

Puoi utilizzare il blocco `await` per attendere l'esecuzione di una promessa all'interno di un blocco `{#await}` e visualizzare un messaggio di caricamento durante l'attesa. Ad esempio:

```
{#await promise}
  <p>Loading...</p>
{:then data}
  <p>Data loaded: {data}</p>
{:catch error}
  <p>Error: {error.message}</p>
{/await}
```

Two-way data binding

In Svelte, puoi utilizzare la sintassi `bind:` per creare un legame bidirezionale tra una variabile nello stato del componente e un elemento del DOM. Ad esempio:

```
<input type="text" bind:value={name}>
<p>Your name is: {name}</p>
```

In questo esempio, il valore dell'input viene legato alla variabile `name` e viene riflessa nel paragrafo sottostante.

Riferimenti ad elementi HTML

Puoi ottenere un riferimento a un elemento HTML utilizzando la direttiva `bind:this`. Ad esempio:

```
<script>
  let inputElement;

  function focusInput() {
    inputElement.focus();
  }
</script>

<input type="text" bind:this={inputElement}>
<button on:click={focusInput}>Focus Input</button>
```

In questo modo, `inputElement` contiene un riferimento all'elemento HTML `input`, che può essere utilizzato per eseguire operazioni come il focus.

Lifecycle dei componenti

Svelte offre diversi metodi di ciclo di vita per gestire il comportamento dei componenti in diverse fasi del loro ciclo di vita. Alcuni di questi metodi includono `onMount`, `onDestroy`, `beforeUpdate`, `afterUpdate`, ecc.

```
<script>
  import { onMount, onDestroy } from 'svelte';

  onMount(() => {
    console.log('Component mounted');
  });

  onDestroy(() => {
    console.log('Component destroyed');
  });
</script>
```

Store

Uno store in Svelte è un contenitore di stato reattivo che può essere condiviso tra più componenti all'interno di un'applicazione. Svelte offre vari tipi di store, come `Writable`, `Readable`, `Writable`, ecc.

Unsubscribe

Quando si utilizzano store Svelte, è importante annullare la sottoscrizione ai cambiamenti quando un componente non è più necessario per evitare memory leaks. Puoi farlo utilizzando il metodo `unsubscribe`.

Readable Store

Uno store `Readable` in Svelte è un contenitore di stato che può essere letto da più componenti, ma solo il componente che lo ha creato può modificarlo. Ad esempio:

```
<script>
  import { readable } from 'svelte/store';

  const count = readable(0, (set) => {
    const interval = setInterval(() => {
      set(prevCount => prevCount + 1);
```

```
    }, 1000);

    return () => clearInterval(interval);
  });
</script>
```

Derived Store

Uno store `Derived` in Svelte è uno store che dipende da uno o più altri store e viene aggiornato automaticamente quando i valori dei suoi store dipendenti cambiano. Ad esempio:

```
<script>
  import { writable, derived } from 'svelte/store';

  const count = writable(0);
  const squared = derived(count, $count => $count * $count);
</script>
```

Custom Store

Puoi creare un custom store in Svelte per gestire lo stato della tua applicazione in base alle tue esigenze specifiche. Puoi farlo utilizzando le funzioni `writable`, `readable`, ecc., per creare uno store personalizzato.