Stencil a fondo

¡Stencil, no es un framework!

Stencil fue creado por el equipo de Ionic para ayudar a construir componentes más rápidos y capaces de funcionar en los principales frameworks.

No pretende ser un framework. Pretender ofrecer una experiencia parecida a trabajar con uno de ellos.

Al utilizar Stencil un compilador para generar los componentes, Stencil analiza el código de los componentes y genera un código muy optimizado.

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

JSX

Es una extensión de la sintaxis de JavaScript.

JSX puede recordarte a un lenguaje de plantillas, pero viene con todo el poder de JavaScript.

Dado que JSX es más cercano a JavaScript que a HTML, usa la convención de nomenclatura camelCase en vez de nombres de atributos HTML. Ej: tabindex se vuelve tabIndex.

Los componentes del Stencil se renderizan utilizando JSX, una popular sintaxis de plantilla declarativa.

Cada componente tiene una función de renderización que devuelve un árbol de componentes que se renderizan en el DOM en tiempo de ejecución.

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

JSX

- **render()**
  La función de "render" se utiliza para dar salida a un árbol de componentes que se dibujará en la pantalla.

- **<Host>**
  Si desea modificar el elemento anfitrión en sí, como añadir una clase o un atributo al propio componente, utilice el componente funcional <Host>

- **Data Binding**
  Los componentes a menudo necesitan renderizar datos dinámicos. Para hacer esto en JSX, usa { } alrededor de una variable

- **Condicionales**
  Si queremos mostrar un contenido diferente de forma condicional, podemos utilizar las sentencias if/else de JavaScript. Además, se pueden crear condicionales en línea utilizando el operador ternario

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

JSX

Stencil reutiliza los elementos del DOM para mejorar el rendimiento.

```
{someCondition
  ? <my-counter initialValue={2} />
  : <my-counter initialValue={5} />
}
```

```
<my-counter initialValue={someCondition ? 2 : 5} />
```

Por lo tanto, si someCondition cambia, el estado interno de <my-counter> no se restablecerá y sus métodos del ciclo de vida como componentWillLoad() no se dispararán. Si desea destruir y volver a crear un componente en un condicional, puede asignar el atributo key. Esto indica a Stencil que los componentes son en realidad hermanos diferentes:

```
{someCondition
  ? <my-counter key="a" initialValue={2} />
  : <my-counter key="b" initialValue={5} />
}
```

ESCUELA IT

2

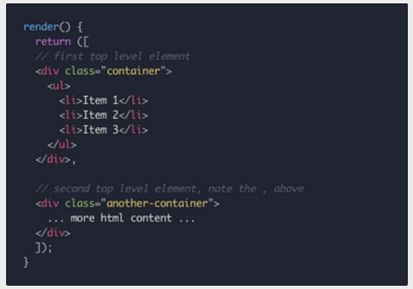
Dominio de Stencil con JSX y Typescript a través de la creación de web components

JSX

- **Slots**
  (Siguiente clase...)

- **Loops (bucles)**
  Los bucles pueden ser creados en JSX usando bucles tradicionales o usando operadores de array como map cuando se inlinean en JSX existentes.
  Cada paso a través de la función map crea un nuevo subárbol JSX y lo añade al array devuelto por map, que se dibuja en el árbol JSX de arriba.

- **Contenido de plantillas complejas**
  En el caso de que un componente tenga múltiples elementos de "nivel superior", la función de renderización puede devolver un array.



ESCUELA IT

2 Dominio de Stencil con JSX y Typescript a través de la creación de web components

JSX

- **Obtener una referencia a un elemento del DOM**
  En los casos en los que necesites obtener una referencia directa a un elemento, como harías normalmente con **document.querySelector,** es posible que quieras utilizar una ref en JSX. Veamos un ejemplo de uso de una ref en un formulario:

  <input type="text" ref={(el) => this.textInput = el as HTMLInputElement} />

- **Styles**

```
render(){
    return (
        <div style={{
          backgroundColor: `#f6f6f6`,
          padding: `20px`
        }}>
            <p>Hello</p>
        </div>
    )
}
```

Estamos asignando una expresión a style que contiene un objeto que representa nuestras propiedades de estilo. Estamos usando camelCase - así que en lugar de usar las propiedades habituales con guiones como background-color o list-style-type usaríamos backgroundColor y listStyleType.

ESCUELA IT

2 Dominio de Stencil con JSX y Typescript a través de la creación de web components

JSX

- **Vinculación de eventos**
  Podemos manejar los eventos del DOM vinculándolos a propiedades como onClick. Por ejemplo, si quisiéramos ejecutar un método que hemos creado llamado handleClick() cuando el usuario hace clic en un botón, podríamos hacer algo así:

```
<ion-button onClick={this.handleClick(event)}>Click me</ion-button>
```

Esto está bien, e invocaría nuestro método handleClick, pero la desventaja de este enfoque es que no mantendrá el alcance de this. Esto significa que si tratas de referenciar una variable miembro como this.loggedIn dentro de tu método handleClick no funcionará.

JSX

Puede resolver este problema de cualquiera de las siguientes maneras. Puede vincular manualmente la función al ámbito correcto de la siguiente manera:

```
<ion-button onClick={this.handleClick(event).bind(this)}>Click me</ion-button>
```

```
<ion-button onClick={(event) => this.handleClick(event)}>Click me</ion-button>
```

recomendaciones:
https://reactjs.org/docs/introducing-jsx.html
https://stenciljs.com/docs/templating-jsx#using-jsx

referencias textos eventos: **https://www.joshmorony.com/understanding-jsx-for-stencil-js-applications/**

ESCUELA IT

2 Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Component()

Cada componente Stencil debe ser decorado con un decorador @Component() del paquete @stencil/core. En el caso más sencillo, los desarrolladores deben proporcionar un nombre de etiqueta HTML para el componente. A menudo, también se utiliza un styleUrl, o incluso styleUrls, donde se pueden proporcionar múltiples hojas de estilo diferentes para diferentes modos/temas de aplicación.

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

@Component({
  tag: 'todo-list',
  styleUrl: 'todo-list.css'
})
export class TodoList {

}
```

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Prop()

Los Props son atributos/propiedades personalizados expuestos públicamente en el elemento para los que los desarrolladores pueden proporcionar valores.
Los componentes hijos no deben conocer o hacer referencia a los componentes padres, por lo que los Props deben ser utilizados para pasar datos del padre al hijo.
Los componentes deben declarar explícitamente las Props que esperan recibir utilizando el decorador @Prop().
Los Props pueden ser un número, una cadena, un booleano, o incluso un objeto o una matriz.
Por defecto, cuando un miembro decorado con un decorador @Prop() se establece, el componente se rerenderizará eficientemente.

```
import { Prop } from '@stencil/core';

...
export class TodoList {
  @Prop() color: string;
  @Prop() favoriteNumber: number;
  @Prop() isSelected: boolean;
  @Prop() myHttpService: MyHttpService;
}
```

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

@State()

El decorador @State() puede utilizarse para gestionar los datos internos de un componente. Esto significa que un usuario no puede modificar estos datos desde fuera del componente, pero el componente puede modificarlos como crea conveniente. Cualquier cambio en una propiedad @State() hará que se llame de nuevo a la función de renderización del componente.

```
import { Component, State, Listen, h } from '@stencil/core';

@Component({
  tag: 'my-toggle-button'
})

export class MyToggleButton {
  @State() open: boolean;

  @Listen('click', { capture: true })
  handleClick() {
    this.open = !this.open;
  }

  render() {
    return <button>
      {this.open ? "On" : "Off"}
    </button>;
  }
}
```

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Event()

NO existen los eventos Stencil, sino que Stencil fomenta el uso de eventos DOM. Sin embargo, Stencil proporciona una API para especificar los eventos que un componente puede emitir, y los eventos que un componente escucha. Lo hace con los decoradores Event() y Listen().

Para enviar eventos DOM personalizados para que los manejen otros componentes, utilice el decorador @Event()

```
import { Event, EventEmitter } from '@stencil/core';

...
export class TodoList {

  @Event() todoCompleted: EventEmitter<Todo>;

  todoCompletedHandler(todo: Todo) {
    this.todoCompleted.emit(todo);
  }
}
```

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Event()

NO existen los eventos Stencil, sino que Stencil fomenta el uso de eventos DOM. Sin embargo, Stencil proporciona una API para especificar los eventos que un componente puede emitir, y los eventos que un componente escucha. Lo hace con los decoradores Event() y Listen().

Para enviar eventos DOM personalizados para que los manejen otros componentes, utilice el decorador @Event()

```
import { Event, EventEmitter } from '@stencil/core';

...
export class TodoList {

  // Event called 'todoCompleted' that is "composed", "cancellabl
  @Event({
    eventName: 'todoCompleted',
    composed: true,
    cancelable: true,
    bubbles: true,
  }) todoCompleted: EventEmitter<Todo>;

  todoCompletedHandler(todo: Todo) {
    const event = this.todoCompleted.emit(todo);
    if(!event.defaultPrevented) {
      // if not prevented, do some default handling code
    }
  }
}
```

ESCUELA IT

2 Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Listen()

El decorador Listen() es para escuchar los eventos del DOM, incluyendo los enviados desde @Events.

En el ejemplo siguiente, supongamos que un componente hijo, TodoList, emite un evento TodoCompleted utilizando el EventEmitter.

```
import { Listen } from '@stencil/core';

...
export class TodoApp {

  @Listen('todoCompleted')
  todoCompletedHandler(event: CustomEvent<Todo>) {
    console.log('Received the custom todoCompleted event: ', ever
  }
}
```

ESCUELA IT

2 Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Method()

El decorador @Method() se utiliza para exponer métodos en la API pública. Las funciones decoradas con el decorador @Method() pueden ser llamadas directamente desde el elemento, es decir, están pensadas para ser llamadas desde el exterior.

```
import { Method } from '@stencil/core';

export class TodoList {

  @Method()
  async showPrompt() {
    // show a prompt
  }
}
```

```
(async () => {
  await customElements.whenDefined('todo-list');
  const todoListElement = document.querySelector('todo-list');
  await todoListElement.showPrompt();
})();
```

ESCUELA IT

2 Dominio de Stencil con JSX y Typescript a través de la creación de web components

@Watch()

Cuando un usuario actualiza una propiedad, Watch disparará el método al que está unido, y pasará a ese método el nuevo valor de la propiedad junto con el valor anterior. Watch es útil para validar propiedades o manejar efectos secundarios. El decorador Watch no se activa cuando un componente se carga inicialmente.

```
import { Prop, Watch } from '@stencil/core';

export class LoadingIndicator {
  @Prop() activated: boolean;

  @Watch('activated')
  watchHandler(newValue: boolean, oldValue: boolean) {
    console.log('The new value of activated is: ', newValue);
  }
}
```

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

¡Reactividad!

Cuando un componente se actualiza debido a un cambio de estado (props o cambio de estado), se programa la ejecución del método render().

En el caso de los arrays, operaciones de array mutables estándar, como push() y unshift(), no provocarán una actualización de los componentes. En su lugar, deben utilizarse operadores de matrices no mutables, ya que devuelven una copia de una nueva matriz.
Estos incluyen map()
y filter(), y la sintaxis del operador de propagación ES6.

```
@State() items: string[];

// our original array
this.items = ['ionic', 'stencil', 'webcomponents'];

// update the array
this.items = [
  ...this.items,
  'awesomeness'
]
```

El operador de propagación también debe utilizarse para actualizar objetos. Al igual que con las matrices, la mutación de un objeto no provocará una actualización de la vista en Stencil, pero la devolución de una nueva copia del objeto sí lo hará. A continuación se muestra un ejemplo:

```
@State() myCoolObject;

// our original object
this.myCoolObject = {first: '1', second: '2'}

// update our object
this.myCoolObject = { ...myCoolObject, third: '3' }
```

ESCUELA IT

2

Dominio de Stencil con JSX y Typescript a través de la creación de web components

Rafa Bernal

Programador Frontend

{pixel developer & Stencil Advocate}

@rbernalBer

autentia