WebComponents

Stencil进阶专题

主题

Data-binding

Unit Testing

E2E Testing

Use Ionic Libs

Router & Apps

实际项目注意事项

Data binding

[演示]


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

@Component({
    styleUrl: 'skin.css',
    tag: 'demo-data-binding',
    // shadow: true,
})
export class DemoDataBinding {
    @State()
    value: string = 'aaa';

    setValue(e) {
        this.value = e.target.value
    }

    render() {
        return (
            <Host>
                <input type='text' value={this.value} onChange={e => {
                    this.setValue(e)
                }} />

                <button onClick={e => {
                    this.value = Math.random() + ''
                }}>{this.value}</button>
            </Host>
        );
    }
}
```

Unit Testing

[演示]

Unit Testing
https://stenciljs.com/docs/unit-testing

```
import { newSpecPage } from '@stencil/core/testing';
import { MyCmp } from '../my-cmp';

it('should render my component', async () => {
  const page = await newSpecPage({
    components: [MyCmp],
    html: `<my-cmp></my-cmp>`,
  });
  expect(page.root).toEqualHtml(`
    <my-cmp>Success!</my-cmp>
  `);
});
```

The example below uses the template option to test the component

```
// mycmp.spec.tsx
// Since the 'template' argument to `newSpecPage` is using jsx s
import { h } from '@stencil/core';
import { newSpecPage } from '@stencil/core/testing';
import { MyCmp } from '../my-cmp';

it('should render my component', async () => {
  const greeting = 'Hello World';
  const page = await newSpecPage({
    components: [MyCmp],
    template: () => (<my-cmp greeting={greeting}></my-cmp>),
  });
  expect(page.root).toEqualHtml(`
    <my-cmp>Hello World</my-cmp>
  `);
});
```

E2E Testing

E2E Testing

End-to-end Testing

E2E tests verify your components in a real browser. For example, when `my-component` has the X attribute, the child component then renders the text Y, and expects to receive the event Z. By using Puppeteer for rendering tests (rather than a Node environment simulating how a browser works), your end-to-end tests are able to run within an actual browser in order to give better results.

Stencil provides many utility functions to help test Jest and Puppeteer. For example, a component's shadow dom can be queried and tested with the Stencil utility functions built on top of Puppeteer. Tests can not only be provided mock HTML content, but they can also go to URLs of your app which Puppeteer is able to open up and test on Stencil's dev server.

End-to-end tests require a fresh build, dev-server, and puppeteer browser instance created before the tests can actually run. With the added build complexities, the `stencil test` command is able to organize the build requirements beforehand.

To run E2E tests, run `stencil test --e2e`. By default, files ending in `.e2e.ts` will be executed.

E2E Testing

https://stenciljs.com/docs/end-to-end-testing

An example E2E test might have the following boilerplate:

```
import { newE2EPage } from '@stencil/core/testing';

describe('example', () => {
  it('should render a foo-component', async () => {
    const page = await newE2EPage();
    await page.setContent(`<foo-component></foo-component>`);
    const el = await page.find('foo-component');
    expect(el).not.toBeNull();
  });
});
```

Example End-to-end Test

```
import { newE2EPage } from '@stencil/core/testing';

it('should create toggle, unchecked by default', async () => {
  const page = await newE2EPage();

  await page.setContent(`
    <ion-toggle class="pretty-toggle"></ion-toggle>
  `);

  const ionChange = await page.spyOnEvent('ionChange');

  const toggle = await page.find('ion-toggle');

  expect(toggle).toHaveClasses(['pretty-toggle', 'hydrated']);

  expect(toggle).not.toHaveClass('toggle-checked');

  toggle.setProperty('checked', true);

  await page.waitForChanges();

  expect(toggle).toHaveClass('toggle-checked');

  expect(ionChange).toHaveReceivedEventDetail({
    checked: true,
    value: 'on'
  });
});
```

Use Ionic Libs

http://johnborg.es/2020/04/ionic-overlays-stencil.html

Add @ionic/core to Project

In order to use Ionic with Stencil, we need to use the core version of the library. This is meant to be used with standard JavaScript projects without frameworks.

```
npm i @ionic/core@latest --save-dev
```

```
/* src/global/app.ts */

import '@ionic/core'
```

Make sure to load `app.ts` as your global startup script.

```
/* stencil.config.ts */

import { Config } from '@stencil/core';

export const config: Config = {
  globalScript: 'src/global/app.ts',
  // ...
}
```

Router & Apps

Router & Apps


```
<stencil-router>
  <stencil-route-switch scrollTopOffset={0}>
    <stencil-route url="/" component="app-home" exact={true} />
    <stencil-route url="/profile/:name" component="app-profile" />
  </stencil-route-switch>
</stencil-router>
```

实际项目注意事项

内容回顾