16. Щёлкаем тест-кейсы с Enzyme

Содержание урока

- Обзор;
- Знакомство с Enzyme;
- Система адаптеров;
- Shallow Rendering;
- Full DOM Rendering;
- Testing Business logic;
- Подведём итоги.

Обзор

Привет! 👋🏼👨🏽‍🎓 В этой части конспекта мы рассмотрим `Enzyme` — утилиту-драйвер для тестирования компонентов React быстро, удобно, надёжно и многофункционально. 🧞

К делу! 🏀🦹🏻‍♀️👊🏼

Знакомство с Enzyme

Enzyme — это утилита для тестирования React-компонентов, улучшающая процесс манипуляции и исследования компонентов в тестовом окружении, а также предоставляющая широкий спектр специализированных ассертов. 🌀

Enzyme обладает интуитивным и лаконичным API, слегка напоминающим синтаксис jQuery (в хорошем смысле, ауч). 🤦🏼‍♀️

Система адаптеров

Enzyme реализует специальную систему `адаптеров`. Данная система предусматривает использование Enzyme с `адаптером` для соответствующей версии React. 🚧

Например, для тестирования компонента, написанного с помощью `React v16.x`, необходимо установить:

- Утилиту `enzyme`;
- Адаптер `enzyme-adapter-react-16`.

Так выглядит начало файла с тестом компонента:

```
import Enzyme from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

Enzyme.configure({ adapter: new Adapter() });
```

Такая система помогает `enzyme` лучше работать с разными версиями React с учетом частого его обновления, тем самым облегчив непосредственно `core`-архитектуру утилиты.

Например, чтобы тестировать компоненты React версий `v0.13.x`, `v0.14.x`, `v15.x`, `v16.x`, необходимо установить адаптер, соответствующий каждой версии `enzyme-adapter-react-{{version}}`. 🐬

Enzyme предоставляет несколько режимов рендеринга React-компонентов в тестовом окружении:

- Shallow rendering;
- Static rendering;
- Full DOM rendering.

Каждый режим имеет свои достоинства и недостатки. Сперва мы рассмотрим `shallow rendering`.

Shallow rendering

Данный режим используется чаще всего: с его помощью можно тестировать компоненты как юниты, в изоляции. Это становится возможным за счёт механики shallow rendering — этот режим рендерит компоненты `на один уровень в глубь`.

🖥️ Пример кода 16.1:

```
// ShoppingList.js
import React, { Component } from 'react';
import ShoppingListItem from './ShoppingListItem';
import './styles.css';

export default class ShoppingList extends Component {
    render () {

        return (
            <ul className = 'shoppingList'>
                <li className = 'shoppingListItem'>Cookies</li>
                <ShoppingListItem
                    item = 'Banana'
                />
```

```
                <ShoppingListItem
                    item = 'Milk'
                />
                <ShoppingListItem
                    item = 'Honey'
                />
            </ul>
        );
    }
}
```

```
// ShoppingListItem.js
import React, { Component } from 'react';
import './styles.css';

export default class ShoppingListItem extends Component {
    render () {
        const { item } = this.props;

        return <li className = 'shoppingListItem'>{ item }</li>;
    }
}
```

```
// ShoppingList.test.js
import React from 'react';
import Adapter from 'enzyme-adapter-react-16';
import { shallow, configure } from 'enzyme';
import ShoppingList from './ShoppingList';

configure({ adapter: new Adapter() });

const result = shallow(<ShoppingList />);

test('<ShoppingList /> should have 1 root element', () => {
    expect(result.find('ul').length).toBe(1);
});

test('<ShoppingList /> should have one element with ‘shoppingListItem‘
class', () => {
    expect(result.find('.shoppingListItem').length).toBe(1);
    expect(result.find('li').hasClass('shoppingListItem'));
});

test('<ShoppingList /> ‘li’ children should have ‘Cookies‘ as text
content', () => {
    expect(result.find('li').text()).toBe('Cookies');
});
```

```
23
24  test('<ShoppingList /> should have 3 ShoppingListItem components as a
    children', () => {
25      expect(result.find('ShoppingListItem').length).toBe(3);
26  });
```

В примере кода `16.1` в файле `ShoppingList.test.js` описан тест компонента `ShoppingList`.

Сперва вызывается метод `configure` для настройки адаптера, а затем с помощью метода `shallow` происходит рендер компонента `ShoppingList`, в итоге возвращающий срендеренный компонент. Результат рендера и используется в тестировании.

Например, можно вызывать метод `find` для поиска различных элементов в отрендеренном компоненте и проверять их различными ассертами и условиями. Метод `find` умеет искать элементы по имени тэга, CSS-класса, имени компонента, имени атрибутов элементов и по многим другим критериям. Метод `find` гибкий и лаконичный.

На `строке кода 21` в тесте вызывается метод `text`, возвращающий текстовый контент элемента и подвергая его ассерту.

А на `строке кода 25` метод `find` использован для поиска компонента по имени `ShoppingListItem` и проверке количества найденных элементов, посредством обращения к свойству `length` и ассерту относительно числа. Данный тест показывает, что компонент `ShoppingList` рендерится на один уровень в глубь. ⛏️

🔎 Хозяйке на заметку:

> Режим `shallow rendring` предоставляет ещё много полезных методов и ассертов, полный список которых доступен на официальном сайте с документацией.

Full DOM rendering

Режим `full DOM rendering` хорошо подходит для сценариев, когда нужно протестировать взаимодействие компонентов с DOM API или методы "жизненного цикла" компонента.

🔎 Хозяйке на заметку:

> В отличие от `shallow rendering`, `full DOM rendering` — компоненты до самого низкого уровня вложенности.

Для тестирования DOM API можно использовать `JSDOM` — JavaScript-имплементацию DOM.

🖥️ Пример кода 16.2:

```
// SpellBook.js
import React, { Component } from 'react';

export default class SpellBook extends Component {
    componentDidMount () {
        console.log('Dong!');
    }

    render () {

        return <h1>I am mastering the sorcery of React!</h1>;
    }
}
```

```
// SpellBook.test.js
import React from 'react';
import Adapter from 'enzyme-adapter-react-16';
import { mount, configure } from 'enzyme';
import { spy } from 'sinon';
import SpellBook from './SpellBook';

configure({ adapter: new Adapter() });

test('<SpellBook /> componentDidMount gets called once', () => {
    spy(SpellBook.prototype, 'componentDidMount');

    mount(<SpellBook />);

 expect(SpellBook.prototype.componentWillMount.calledOnce).toBe(true);
});
```

`Пример кода 16.2` описывает пример тестирования метода "жизненного цикла" компонента посредством активации `шпиона` на прототипе компонента. Концепция `шпионов` распространена в тестировании программного кода.

🖐🏽 Совет бывалых:

> Режим `full DOM rendering` даёт максимум потенциала для тестирования компонентов любой сложности, однако этот режим заметно медленней режима `shallow rendering` в контексте производительности. Поэтому учитывать эту деталь при необходимости использования метода `mount` в тестах весьма целесообразно.

Подведём итоги

Enzyme — мощная и шустрая утилита для тестирования React-компонентов с красивым и удобным API. 💂

Для использования Enzyme следует установить `адаптер`, соответствующий используемой версии React.

Спасибо, что остаёшься с нами! 👋🏼 Надеюсь, данный конспект был тебе полезен. Успешного тебе программирования. До встречи! 🦋

> Мы будем очень признательны, если ты оставишь свой фидбек в отношении этой части конспекта на нашу электропочту `hello@lectrum.io`.