Rendering Static Pages in React

Ustun Ozgur

2015-11-03 Tue

Rendering a Static Page in React

render method

- In render, we return a JSX expression
- Composed of ordinary HTML elements or composite React components

JSX in Detail

- Not a version of HTML or XML
- Transpiled to ordinary JavaScript
- `<div name="foo" surname="bar">content</div>`
- `React.createElement('div', {name: "foo", surname: "bar"}, "content")`
- Attributes collected in an object called *props*
- Children passed as consecutive arguments
- `functionForTheElement(props, children)`
- Can nest: Composite components

A more complex Example

```
<div name="foo"><span>Bar</span><span>Baz</span></div>

React.createElement('div', {name: 'foo'},
   React.createElement('span', {}, "Bar"),
   React.createElement('span', {}, "Baz"))
```

JSX: Give it 5 Minutes

- HTML in JS?
- JS in HTML?
- Actually no mixing of JS and HTML, it is all JS

Main Features

- Always return a single element
  - Just like you return a single value from a function
- All elements should be closed or self-closing
  - `<img/>`, `<input/>`
- Use curly braces to embed JSX expressions
  - `<div>{2 + 3}</div>` valid JSX

Main Features

```
var inner = <div>Hello</div>

var outer = <div>{inner} {inner}</div>
```

- Not string concetenation. `inner` is just another variable.

```
var contents = [];

contents.push(inner);
contents.push(inner);

var outer = <div>{contents}</div>
```

Main Features

```
var names = ["John", "Mary", "Jane"];

var contents = names.map(function (name) {
    return <div>Hello {name}</div>;
});

var outer = <div>{contents}</div>
```

Main Features

```
var content;

if (this.props.color == "red") {
   content = <div>The color is red.</div>
}


outer = <div>{content}</div>
```

Main Features

- If statements are not expressions
- But ternaries are expressions: ~a == 3 ? "a is 3" : "a is not 3"~

```
outer = <div>{this.props.color == "red" ?
                  <div>The color is red</div>:  null}</div>

outer = <div>{this.props.color == "red" &&
                 <div>The color is red</div>}</div>
```

- May or may not make the reading harder

CSS in JSX

- JSX is JS behind the scenes
- `class` is reserved in JS
- Use `className` instead
- `<div class="foo bar">` becomes `<div className="foo bar">`
- Styles in HTML are strings. In CSS, a syntax similar to JS objects
- In JSX, styles are real JS objects

CSS in JSX (cont'd)

`<div style="color: red; background-color: yellow">`

- String converted to JS object

`{color: 'red', backgroundColor: 'yellow'}` `<div style={{color: 'red', backgroundColor: 'yellow'}}`

- Note the double {'s: The first is for JSX, the second is for JS object literal
- No dashes: instead capitalize the next letter:

`backgroundColor` instead of `background-color`

- We could have quoted background-color since it is in object key position, but this is easier

CSS in JSX (cont'd)

- Numbers in JSX: Most units are px in CSS: so automatically appende

```
fontSize:'12px'
```

can be written as:

```
fontSize: 12
```

CSS in JSX (cont'd)

- Doesn't always work as expected. For lineHeight, the default unit is 'em'

`lineHeight: 18 means lineHeight: "18em"`

Write

`lineHeight: '18px'`

Converting Existing HTML to React

1. Create a React component called App. Take the existing HTML and return it from the render method of the component.
2. Make the required CSS changes, so that classes are converted classNames, styles are converted to objects.
3. Ensure that all the tags are closed properly.
4. Reference the App.js file, add a container HTML element where your React component will live, mount the React component using ‘ReactDOM.render‘. In fact, you can directly mount to body, but this is discouraged since sometimes using external plugins such as Facebook or Twitter embeds add some nodes into the body and this might cause some bugs.

For Steps 2-3, run the transpiler in watch mode. Make changes until it no longer complains.

Tools for Converting JSX to JavaScript

- Transpilation
- Install babel-cli using `npm install -g babel-cli`
- Install React preset for babel using `npm install` (preset specified in package.json)
- Two modes: single file input, or directory input

A Simple Example

```
var HelloWorld = React.createClass({
    render: function () {
            return <div>Hello World</div>
    }
})
```

In HTML, we have

```
<div id="app"></div>
```

Mount using:

```
ReactDOM.render(<HelloWorld/>, document.getElementById('app
```

Exercise 1: Getting up and running with React: 5-10 minutes

Hint: Look at Makefile for instructions.
a - Clone the repository
b - Go to `examples/01_hello_world`
c - Download react.js, install babel, compile the application using babel, and see "Hello World" in browser.
d - Tweak the application so that you see "Hello Sweden!" in browser.
e - Repeat for the folder `examples/01_hello_world_watch` but this time, make sure to run babel in watch mode.
f - Bonus: How would you copy the ReactDOM.render call to an inline script in HTML? You need to transform the JSX manually to a React.createElement call.
g - Question: Why do we put the script at the end of body?

Exercise 2:

a - Convert the HTML given in `02_convert_mockup/mockup.html` to React. Note that you will have two root nodes here, the main part and the footer.

- Run webpack-dev-server from the examples folder and go to `http://localhost:8080` for the remaining exercises.

Exercise 3: A More Complex Example: ToDo App

- Given `examples/03_todo/mockup.html`, convert to React.
- Online HTML to JSX transformer: `https://facebook.github.io/react/html-jsx.html`
- Command line version: `https://www.npmjs.com/package/htmltojsx`
- Paste output in `src/todo.js`
- Dissect the ToDo app in smaller components: You should have components

like SearchBar, Todos, TodoItem, Footer. This will prepare us for the next section, in which we discuss props.