ReactJS Framework

Date: 03-21-2017

Chapter 1 What is

What is ReactJS?

A framework developed by facebook, it is fast and component based, and it has the virtual DOM concept to simplify stuff. JSX is a file format that a html is inside js.

1 Installation

NodeJS and NPM required. Atom editor is required for studying and developing code.

1.1 Install global packages

npm install -g babel

npm install -g babel-cli

(????)

1.2 Create project root folder and package.json

mkdir reactApp

npm init

then something will be created like this,

About to write to /Users/xiaofeng/reactjs/reactApp/package.json:

{

"name": "helloworld",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

1.3 Add dependencies and plugins

The **--save** command will add these packages to **package.json** file.

Then we will be needing the following,

C:\Users\username\Desktop\reactApp>npm install webpack --save

C:\Users\username\Desktop\reactApp>npm install webpack-dev-server --save

C:\Users\username\Desktop\reactApp>npm install react --save

C:\Users\username\Desktop\reactApp>npm install react-dom --save

C:\Users\username\Desktop\reactApp>npm install babel-core

C:\Users\username\Desktop\reactApp>npm install babel-loader

C:\Users\username\Desktop\reactApp>npm install babel-preset-react

C:\Users\username\Desktop\reactApp>npm install babel-preset-es2015

Please check my install.sh to install them at once.

Summary:

# Babel is a JavaScript compiler that compiles ES2015.

ECMAScript 2015 is an ECMAScript standard that was ratified in June 2015.

ES2015 is a significant update to the language, and the first major update to the language since ES5 was standardized in 2009. Implementation of these features in major JavaScript engines is [underway now](https://kangax.github.io/es5-compat-table/es6/).

See the [ES2015 standard](http://www.ecma-international.org/ecma-262/6.0/index.html) for full specification of the ECMAScript 2015 language.

For example, future js looks like this,

*let yourTurn = "Type some code in here!";*

and current compatible js is this,

*var yourTurn = "Type some code in here!";*

Ref: https://www.tutorialspoint.com/reactjs/reactjs\_environment\_setup.htm

**First running example**

In this example, we will use webpack-deve-server, that way you can use a builtin http with easy port setting to run a server to test your js. To start the server, you can check it looks like the following,

xiaofengmaclap:reactApp xiaofeng$ npm start

> helloworld@1.0.0 start /Users/xiaofeng/reactjs/reactApp

> webpack-dev-server --hot

Project is running at http://localhost:8081/

webpack output is served from /

loaderUtils.parseQuery() received a non-string value which can be problematic, see https://github.com/webpack/loader-utils/issues/56

parseQuery() will be replaced with getOptions() in the next major version of loader-utils.

Hash: 16ea1e418b5d2aa8fb17

Version: webpack 2.2.1

Time: 1723ms

Asset Size Chunks Chunk Names

index.js 1.06 MB 0 [emitted] [big] main

chunk {0} index.js (main) 1 MB [entry] [rendered]

[98] ./~/react/react.js 56 bytes {0} [built]

[115] ./main.js 470 bytes {0} [built]

[116] ./~/webpack/hot/dev-server.js 1.57 kB {0} [built]

[117] (webpack)-dev-server/client?http://localhost:8081 5.44 kB {0} [built]

[118] ./App.jsx 2.26 kB {0} [built]

[133] ./~/react-dom/index.js 59 bytes {0} [built]

[216] ./~/webpack/hot/emitter.js 77 bytes {0} [built]

[217] ./~/webpack/hot/log-apply-result.js 1.02 kB {0} [built]

[218] (webpack)-dev-server/client/overlay.js 3.6 kB {0} [built]

[219] (webpack)-dev-server/client/socket.js 856 bytes {0} [built]

[223] (webpack)-dev-server/~/html-entities/index.js 231 bytes {0} [built]

[255] (webpack)-dev-server/~/strip-ansi/index.js 161 bytes {0} [built]

[258] (webpack)/hot/emitter.js 77 bytes {0} [built]

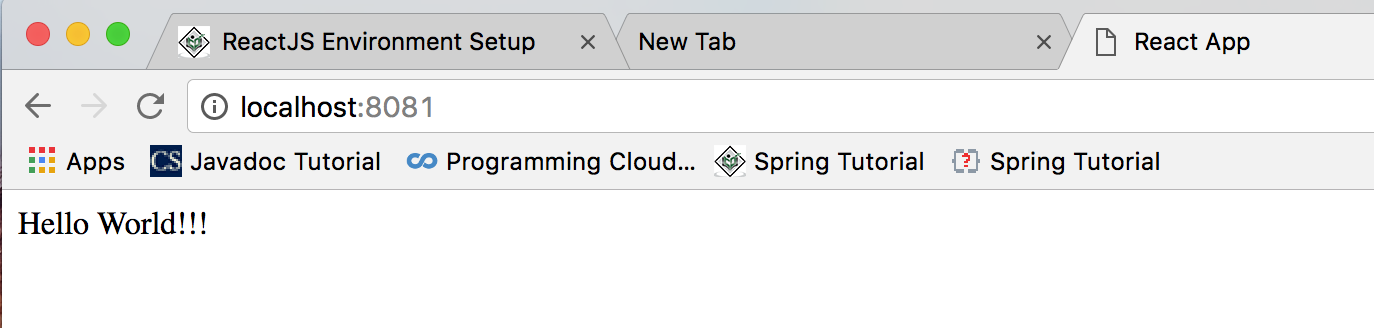
[263] (webpack)/~/url/url.js 23.3 kB {0} [built]

[265] multi (webpack)-dev-server/client?http://localhost:8081 webpack/hot/dev-server ./main.js 52 bytes {0} [built]

+ 251 hidden modules

webpack: Compiled successfully.

The output looks like this.



Now let us see the code and how react works.

webpack.config.js

var config = {

entry: '/Users/xiaofeng/reactjs/reactApp/main.js',

output: {

path:'/Users/xiaofeng/reactjs/reactApp',

filename: 'index.js',

},

devServer: {

inline: true,

port: 8081

},

module: {

loaders: [

{

test: /\.jsx?$/,

exclude: /node\_modules/,

loader: 'babel-loader',

query: {

presets: ['es2015', 'react']

}

}

]

}

}

module.exports = config;

**main.js**

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));

**index.html**

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "UTF-8">

<title>React App</title>

</head>

<body>

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

<script src = "index.js"></script>

</body>

</html>

**App.jsx**

import React from 'react';

class App extends React.Component {

render() {

var myStyle = {

fontSize: 100,

color: '#FF0000'

};

//ternary

var i = 1;

return (

<div>

{/\*hello\*/}

<h1 style={myStyle}> Header</h1>

<h2>Content</h2>

{/\*hello\*/}

<p data-myattribute = "somevalue">Hello World!!! This is the content</p>

{1+1}

<h1>{i == 1?'True!':'False'}</h1>

</div>

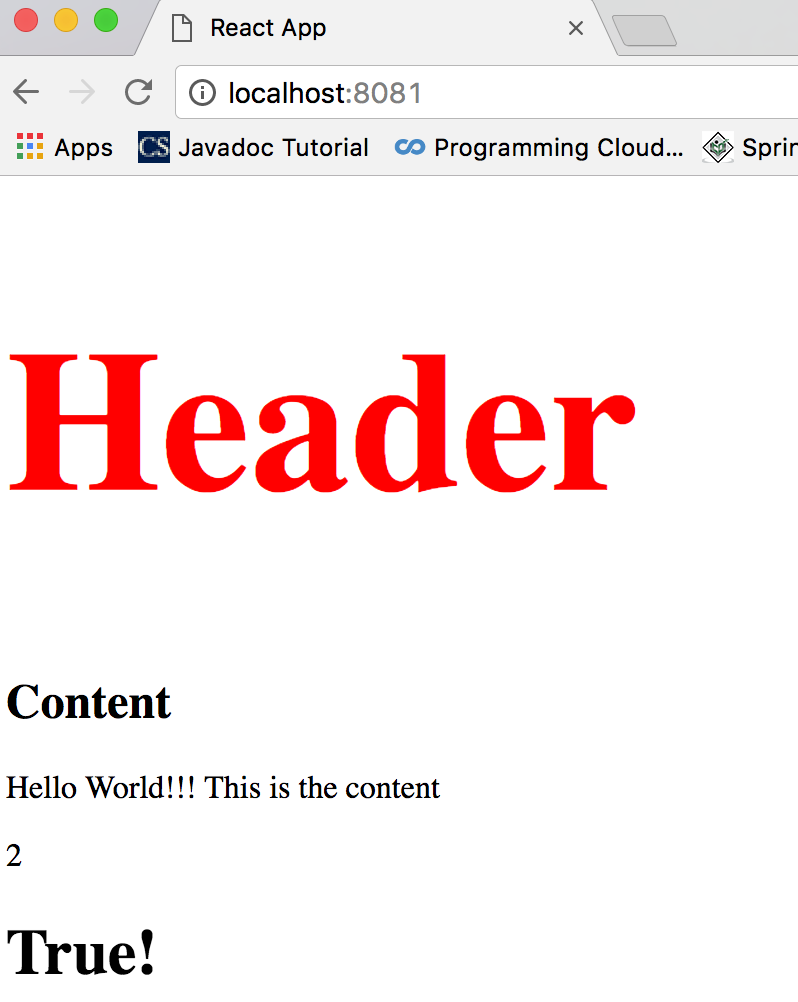
);

}

}

export default App;

**Output:**

****

**Troubleshooting**

To trouble shooting the webpack-dev-server issue, please pay close attention to the error message.

* It basically means any path in the webpack.config.js should be absolute path, not something like './'.
* Port is not being used.
* The following loader is 'babel-loader' instead of 'babel'

loader: 'babel-loader'

* Install everything in node\_modules in the same directory with webpack.config.js. There might be a way to setup the path if node\_modules is not in the same directory with webpack.config.js, but I will need to check it out.

**How react works?**

* \*.jsx is the js with html inside to define the template and component.

.jsx and other js can be called in another js, for example, main.js.

* main.js will be compiled to something virtually exists, called index.js.
* index.js is used in html file.

Index.js is configured to be created in webpack.config.js. Because webpack is a bundling software to make the project management.

* Naming the jsx file starting with camel case and it is a 'export default class Name extends component.\*'. It is a class.
* Jsx can only override render() once.
* Inside of render(), it can only be html tags.
* Inside of render(), it can be only one <div>.

2. JSX

JSX is JS. And it is used for templating than JS. It is then compiling to JS and it is faster.

**2.1 Custom Attributes of html tags**

You can use your own custom attributes in addition to regular HTML properties and attributes. When you want to add custom attribute, you need to use **data-** prefix.

**2.2 JS expression**

If you want to evaluate something, put them in {}.

**2.3 JS conditional**

You can not use if/else in {} but you can use **conditional (ternary)** expressions instead.

Eg:

{i == 1 ? 'True!' : 'False'}

**2.4 Styling**

React recommends using inline styles. When you want to set inline styles, you need to use **camelCase**syntax. React will also automatically append **px**after the number value on specific elements.

**2.4 Comments**

you can use // or /\*\*/ but they have to be inside of {}.

## **2.5 Naming Convention**

HTML tags are always using **lowercase** tag names, while React components starts with **Uppercase**.

You should use **className** and **htmlFor** as XML attribute names instead of **class** and **for**.

**2.6 Export**

[Export](https://developer.mozilla.org/en/docs/web/javascript/reference/statements/export) like export default HelloWorld; and [import](https://developer.mozilla.org/en/docs/web/javascript/reference/statements/import), such as import React from 'react' are part of the [ES6](http://www.2ality.com/2014/09/es6-modules-final.html)

[modules system](http://www.2ality.com/2014/09/es6-modules-final.html).

There are two kinds of exports that can reside in a module:

**Named exports** - for example export function func() {} is a named export with the name of func. Named modules can be imported using import { exportName } from 'module';. In this case the name of the import should be the same, as the name of the export, so to import the func in the example, you'll have to use import { func } from 'module';. There can be multiple exports in one module.

**Default export** - is the value that will be imported from the module, if you use the simple import statement import X from 'module'. X is the name that will be given locally to the variable assigned to contain the value, and it doesn't have to be named like the origin export. There can be only one default export.

A module can contain both named exports and a default export, and they can be imported together using import defaultExport, { namedExport1, namedExport3, etc... } from 'module';.

**2.7 Import**

2.7.1

/\*\*

\* This is how you import stuff. In this case you're actually

\* importing two things: React itself and just the "Component"

\* part from React. Importing the "Component" part by itself makes it

\* so that you can do something like:

\*

\* class MyComponent extends Component ...

\*

\* instead of...

\*

\* class MyComponent extends React.Component

\*

\* Also note the comma below

\*/

import react, {Component} from 'react';

2.7.2

/\*\*

\* This is a "default" export. That means when you import

\* this module you can do so without needing a specific module

\* name or brackets, e.g.

\*

\* import Header from './header';

\*

\* instead of...

\*

\* import { Header } from './header';

\*/

export default class Header extends Component {

}

2.7.3 ()

/\*\*

\* This is a named export. That means you must explicitly

\* import "Header" when importing this module, e.g.

\*

\* import { Header } from './header';

\*

\* instead of...

\*

\* import Header from './header';

\*/

export const Header = React.createClass({

})

2.7.4

/\*\*

\* This is another "default" export, only just with a

\* little more shorthand syntax. It'd be functionally

\* equivalent to doing:

\*

\* const MyClass = React.createClass({ ... });

\* export default MyClass;

\*/

export default React.createClass({

})

When creating React class components, for most intents and purposes you can think of React.createClass as functionally equivalent to class MyComponent extends React.Component. One is just using the new ES2015 class syntax whereas the other is using the pre-ES2015 syntax.

For example,

Exporting without default means it's a "named export". You can have multiple named exports in a single file. So if you do this,

export class Template {}

export class AnotherTemplate {}

Then you have to import these exports using their exact names. So to use these components in another file you'd have to do,

import {Template, AnotherTemplate} from './components/templates'

Alternatively if you export as the default export like this,

export default class Template {}

Then in another file you import the default export without using the {}, like this,

import Template from './components/templates'

There can only be one default export per file. In React it's a convention to export one component from a file, and to export it is as the default export.

You're free to rename the default export as you import it,

import TheTemplate from './components/templates'

And you can import default and named exports at the same time,

import Template,{AnotherTemplate} from './components/templates'

ref: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export>

**2.8 build your project (Frontend stuff)**

In the directory where you put your webpack.config.js, you can run: `webpack`, that will get the “bundled” js generated.

In reality, this step is a must-do step to produce all the final js your project would need. In our webpack-dev-server, it just generate it and hotplug it, so even when you modify the code, you will not to restart the server, and you do not see the intermediate js files.

xiaofengmaclap:reactApp xiaofeng$ webpack

loaderUtils.parseQuery() received a non-string value which can be problematic, see https://github.com/webpack/loader-utils/issues/56

parseQuery() will be replaced with getOptions() in the next major version of loader-utils.

Hash: **0e8c9c8129b46178c754**

Version: webpack **2.2.1**

Time: **1174**ms

**Asset** **Size** **Chunks** **Chunk Names**

**index.js** **729 kB** **0** **[emitted]** **[big]** main

[6] **./~/fbjs/lib/ExecutionEnvironment.js** 1.06 kB {**0**} **[built]**

[8] **./~/react-dom/lib/ReactInstrumentation.js** 601 bytes {**0**} **[built]**

[10] **./~/react-dom/lib/ReactUpdates.js** 9.53 kB {**0**} **[built]**

[19] **./~/react/lib/React.js** 2.69 kB {**0**} **[built]**

[52] **./~/react/react.js** 56 bytes {**0**} **[built]**

[80] **./App.jsx** 2.97 kB {**0**} **[built]**

[81] **./~/react-dom/index.js** 59 bytes {**0**} **[built]**

[109] **./~/react-dom/lib/ReactDOM.js** 5.14 kB {**0**} **[built]**

[169] **./~/react/lib/ReactChildren.js** 6.19 kB {**0**} **[built]**

[170] **./~/react/lib/ReactClass.js** 26.5 kB {**0**} **[built]**

[171] **./~/react/lib/ReactDOMFactories.js** 5.53 kB {**0**} **[built]**

[172] **./~/react/lib/ReactPropTypes.js** 15.8 kB {**0**} **[built]**

[173] **./~/react/lib/ReactPureComponent.js** 1.32 kB {**0**} **[built]**

[174] **./~/react/lib/ReactVersion.js** 350 bytes {**0**} **[built]**

[178] **./main.js** 470 bytes {**0**} **[built]**

+ 164 hidden modules

xiaofengmaclap:reactApp xiaofeng$ ls

App.jsx main.js package.json

index.html node\_modules webpack.config.js

index.js npm-debug.log

In this case, our generated index.js has the following content, and it is very large,

……

SyntheticEvent.Interface = EventInterface;

if (process.env.NODE\_ENV !== 'production') {

if (isProxySupported) {

/\*eslint-disable no-func-assign \*/

SyntheticEvent = new Proxy(SyntheticEvent, {

construct: function (target, args) {

return this.apply(target, Object.create(target.prototype), args);

},

apply: function (constructor, that, args) {

return new Proxy(constructor.apply(that, args), {

set: function (target, prop, value) {

if (prop !== 'isPersistent' && !target.constructor.Interface.hasOwnProperty(prop) && shouldBeReleasedProperties.indexOf(prop) === -1) {

process.env.NODE\_ENV !== 'production' ? warning(didWarnForAddedNewProperty || target.isPersistent(), 'This synthetic event is reused for performance reasons. If you\'re ' + 'seeing this, you\'re adding a new property in the synthetic event object. ' + 'The property is never released. See ' + 'https://fb.me/react-event-pooling for more information.') : void 0;

didWarnForAddedNewProperty = true;

}

target[prop] = value;

return true;

}

});

}

});

/\*eslint-enable no-func-assign \*/

}

}

**2.9 How webpack works**

**content.js**

module.exports = "It works from content.js.";

**entry.js**

document.write(require("./content.js"));

**webpack.config.js**

module.exports = {

entry: "./entry.js",

output: {

path: \_\_dirname,

filename: "bundle.js"

},

module: {

loaders: [

{ test: /\.css$/, loader: "style!css" }

]

}

};

to run it,

webpack entry.js bundle.js

and bundle.js will be generated in the output location.

**ref:**

[https://blog.madewithenvy.com/getting-started-with-webpack-2-ed2b86c68783#.18thf8edr](https://blog.madewithenvy.com/getting-started-with-webpack-2-ed2b86c68783" \l ".18thf8edr)

<http://krasimirtsonev.com/blog/article/javascript-library-starter-using-webpack-es6>

[https://medium.com/@rajaraodv/webpack-the-confusing-parts-58712f8fcad9#.os7punqdq](https://medium.com/@rajaraodv/webpack-the-confusing-parts-58712f8fcad9" \l ".os7punqdq)

[https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460#.1t8qwpafq](https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460" \l ".1t8qwpafq)

<https://webpack.js.org/configuration/output/>

https://webpack.github.io/docs/configuration.html

Chapter 2 Components

2.1 Stateless example

In this example, we will create a component App, and which is the owner of component Header and Content.

App.jsx

import React from 'react';

import Header from './Header.jsx';

import Content from './Content.jsx';

class App extends React.Component {

render() {

return (

<div>

<Header/>

<Content/>

</div>

);

}

}

export default App;

Content.jsx

import React from 'react';

class Content extends React.Component {

render() {

return (

<div>

this is content

</div>

);

}

}

export default Content;

Header.jsx

import React from 'react';

class Header extends React.Component {

render() {

return (

<div>

This is header

</div>

);

}

}

export default Header;

main.js

To render the component on page, we will need to import react-dom.render().

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));

webpack.config.js is the same as the above examples.

Check the result by hitting: localhost:8081

How this works?

1/ From js to reference jsx or from jsx to reference jsx, we will need to include jsx extension in the import statement and its relative path.

Eg: import ComponentName from '…/Afile.jsx'”

2/ For js to render stuff, you will need to import react-dom component so that you can call render method.

2.2 Stateful Example

In this example we will set the state for owner component (**App**). The **Header** component is just added like in the last example since it doesn't need any state. Instead of content tag, we are creating **table** and **tbody** elements where we will dynamically insert **TableRow** for every object from the **data** array. You can see that we are using EcmaScript 2015 arrow syntax (⇒) which looks much cleaner then the old JavaScript syntax.

In this example, we put all the classes in one jsx file.

App.jsx

import React from 'react';

class App extends React.Component {

constructor() {

super();

this.state = {

data:

[

{

"id":1,

"name":"Foo",

"age":"20"

},

{

"id":2,

"name":"Bar",

"age":"30"

},

{

"id":3,

"name":"Baz",

"age":"40"

}

]

}

}

render() {

return (

<div>

<Header/>

<table>

<tbody>

{this.state.data.map((person, i) => <TableRow key = {i} data = {person} />)}

</tbody>

</table>

</div>

);

}

}

class Header extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

</div>

);

}

}

class TableRow extends React.Component {

render() {

return (

<tr>

<td>{this.props.data.id}</td>

<td>{this.props.data.name}</td>

<td>{this.props.data.age}</td>

</tr>

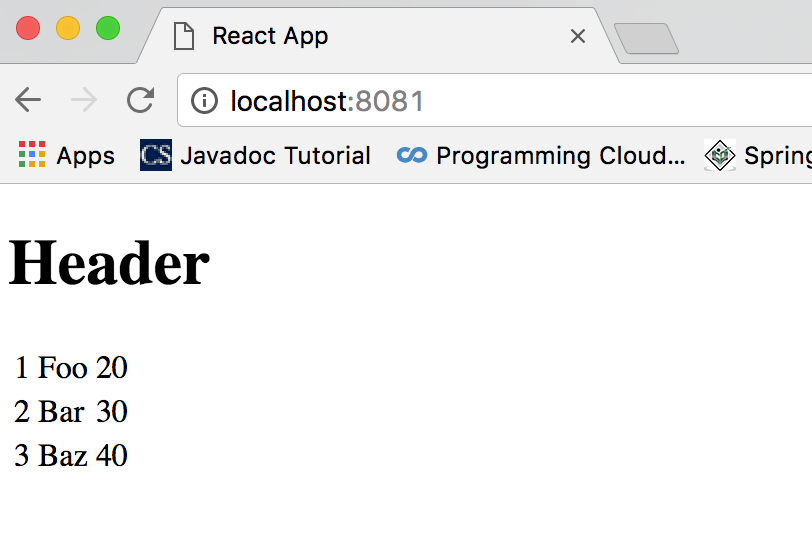
);

}

}

export default App;

main.js and webpack.config.js are skipped as they are the same as the above examples. Output looks like the following,



…..

<table>

<tbody>

{this.state.data.map((person, i) => <TableRow key = {i} data = {person} />)}

</tbody>

</table>

….

and

class TableRow extends React.Component {

render() {

return (

<tr>

<td>{this.props.data.id}</td>

<td>{this.props.data.name}</td>

<td>{this.props.data.age}</td>

</tr>

);

}

}

how to understand the above part?

Because TableRow has a property called 'data', so this.props.data will get the value of {person} which is mapped from

'this.state.data' from App's constructor.

The trick to understand here is the map(a => b) function feature,

[1,2,3].map(function(item){

return item+1;

})

The above reactJS example uses generic js array map like above,

More info can be found in Array.prototype.\*() reference.

To use map generically,

var map = Array.prototype.map;

var a = map.call('Hello World', function(x) {

return x.charCodeAt(0);

});

### Normal use of map, Using map to reverse a string

var str = '12345';

Array.prototype.map.call(str, function(x) {

return x;

}).reverse().join('');

Chapter 3 State and Props

State is the place where the data comes from. You should always try to make your state as simple as possible and minimize number of stateful components. If you have, for example, ten components that need data from the state, you should create one container component that will keep the state for all of them.

The main difference between state and props is that **props** are immutable. This is why container component should define state that can be updated and changed, while the child components should only pass data from the state using props.

Props is an object containing properties passed in a tag component.

Props.items is the property that named 'items'.

Props.item.map() maps the items array to something.

Try to understand map with ES6.

<tr> { this.arr.map((obj, i) => <td key={i}> {obj.name} </td>) } </tr>

## **3.1 Using state**

Code sample below shows how to create stateful component using EcmaScript2016 syntax.

### App.jsx

import React from 'react';

import $ from 'jquery';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

"header": "this is header",

content: "this is content"

}

}

render() {

var output = "";

$.each(this.state, function(value) {

output= output+ value + ' ';

});

return (

<div>

<h1></h1>

<h1>{output}</h1>

</div>

);

}

}

export default App;

### main.js, webpack.config.js and html files are the same as the above examples.

### In this examples, we integrate with Jquery, just like other packages. When webpack builds, it will combine and generate a bundled js.

## **3.2 Using Props**

## Props is short term for property objects of a tag.

When you need immutable data in your component you can just add props to **reactDOM.render()** function in **main.js** and use it inside your component.

### App.jsx

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>{this.props.headerProp}</h1>

<h2>{this.props.contentProp}</h2>

</div>

);

}

}

export default App;

### main.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App headerProp = "Header from props..." contentProp = "Content

from props..."/>, document.getElementById('app'));

export default App;

## html and webpack.config.js are the same with above examples.

## **3.3 Default Props**

You can also set default property values directly on the component constructor instead of adding it to **reactDom.render()** element.

### App.jsx

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h1>{this.props.headerProp}</h1>

<h2>{this.props.contentProp}</h2>

</div>

);

}

}

App.defaultProps = {

headerProp: "Header from props...",

contentProp:"Content from props..."

}

export default App;

### main.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));

**Note that:**

The App.defaultProps is not inside of the class. It is outside of the class but part of the

same file.

**3.4 Using State and Props together**

The example below shows how to combine state and props in your app. We are setting **state** in our parent component and passing it down the component tree using **props**. Inside **render** function, we are setting **headerProp** and **contentProp** that are used in child components.

**https://www.tutorialspoint.com/reactjs/reactjs\_props\_overview.htm**

**(can you understand this?)**

**3.5 Validating Props**

Properties validation is useful way to force correct usage of your components. This will help you during development to avoid future bugs and problems once your app become larger. It also makes code more readable since you can see how each component should be used.

**App.propTypes** is used for props validation. If some of the props aren't using correct type that we assigned, we will get console warning. After we specified validation patterns, we are setting **App.defaultProps**.

App.jsx,

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<h3>Array: {this.props.propArray}</h3>

<h3>Bool: {this.props.propBool ? "True..." : "False..."}</h3>

<h3>Func: {this.props.propFunc(3)}</h3>

<h3>Number: {this.props.propNumber}</h3>

<h3>String: {this.props.propString}</h3>

<h3>Object: {this.props.propObject.objectName1}</h3>

<h3>Object: {this.props.propObject.objectName2}</h3>

<h3>Object: {this.props.propObject.objectName3}</h3>

</div>

);

}

}

App.propTypes = {

propArray: React.PropTypes.array.isRequired,

propBool: React.PropTypes.bool.isRequired,

propFunc: React.PropTypes.func,

propNumber: React.PropTypes.number,

propString: React.PropTypes.string,

propObject: React.PropTypes.object

}

App.defaultProps = {

propArray: [1,2,3,4,5],

propBool: true,

propFunc: function(e){return e},

propNumber: 1,

propString: "String value...",

propObject: {

objectName1:"objectValue1",

objectName2: "objectValue2",

objectName3: "objectValue3"

}

}

export default App;

### main.js, index.html, webpack.config.js are the same as other projects.

**https://www.tutorialspoint.com/reactjs/reactjs\_jsx.htm**

**Chapter 4 Component API and Life Cycle**

4.1 Common API

We will show you three methods: **setState()**, **forceUpdate**and **ReactDOM.findDOMNode()**. In new **ES6**classes we have to manually bind **this**. You will see in examples below that we are using **this.method.bind(this)**.

Example 1, setState()

import React from 'react';

class App extends React.Component {

constructor() {

super();

this.state = {

data: []

}

this.setStateHandler = this.setStateHandler.bind(this);

};

setStateHandler() {

var item = "setState..."

var myArray = this.state.data;

myArray.push(item)

this.setState({data: myArray})

};

render() {

return (

<div>

<button onClick = {this.setStateHandler}>SET STATE</button>

<h4>State Array: {this.state.data}</h4>

</div>

);

}

}

export default App;

Example 2, ForceUpdate()

import React from 'react';

class App extends React.Component {

constructor() {

super();

this.forceUpdateHandler = this.forceUpdateHandler.bind(this);

};

forceUpdateHandler() {

this.forceUpdate();

};

render() {

return (

<div>

<button onClick = {this.forceUpdateHandler}>FORCE UPDATE</button>

<h4>Random number: {Math.random()}</h4>

</div>

);

}

}

export default App;

Example 3, findDOMNode(node)

import React from 'react';

import ReactDOM from 'react-dom';

class App extends React.Component {

constructor() {

super();

this.findDomNodeHandler = this.findDomNodeHandler.bind(this);

};

findDomNodeHandler() {

var myDiv = document.getElementById('myDiv');

ReactDOM.findDOMNode(myDiv).style.color = 'green';

}

render() {

return (

<div>

<button onClick = {this.findDomNodeHandler}>FIND DOME NODE</button>

<div id = "myDiv">NODE</div>

</div>

);

}

}

export default App;

main.js, webpack.config.js and index.html are the same as before.

4.2 Component Life Cycle

## Lifecycle Methods

* **componentWillMount** is executed before rendering, on both server and client side.
* **componentDidMount** is executed after first render only on the client side. This is where AJAX requests and DOM or state updates should occur. This method is also used for integration with other JavaScript frameworks and any functions with delayed execution like **setTimeout** or **setInterval**. We are using it to update the state so we can trigger the other lifecycle methods.
* **componentWillReceiveProps** is invoked as soon as the props are updated before another render is called. We triggered it from **setNewNumber** when we updated the state.
* **shouldComponentUpdate** should return **true** or **false** value. This will determine if component will be updated or not. This is set to **true** by default. If you are sure that component doesn't need to render after **state** or **props** are updated, you can return **false** value.
* **componentWillUpdate** is called just before rendering.
* **componentDidUpdate** is called just after rendering.
* **componentWillUnmount** is called after the component is unmounted from the dom. We are unmounting our component in **main.js**.

In our example we are setting initial **state** in constructor function. The **setNewnumber** is used to update the **state**. All the lifecycle methods are inside **Content** component.

App.jsx

import React from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

data: 0

}

this.setNewNumber = this.setNewNumber.bind(this)

};

setNewNumber() {

this.setState({data: this.state.data + 1})

}

render() {

return (

<div>

<button onClick = {this.setNewNumber}>INCREMENT</button>

<Content myNumber = {this.state.data}></Content>

</div>

);

}

}

class Content extends React.Component {

componentWillMount() {

console.log('Component WILL MOUNT!')

}

componentDidMount() {

console.log('Component DID MOUNT!')

}

componentWillReceiveProps(newProps) {

console.log('Component WILL RECIEVE PROPS!')

}

shouldComponentUpdate(newProps, newState) {

return true;

}

componentWillUpdate(nextProps, nextState) {

console.log('Component WILL UPDATE!');

}

componentDidUpdate(prevProps, prevState) {

console.log('Component DID UPDATE!')

}

componentWillUnmount() {

console.log('Component WILL UNMOUNT!')

}

render() {

return (

<div>

<h3>{this.props.myNumber}</h3>

</div>

);

}

}

export default App;

main.js

**import** React from 'react';

**import** ReactDOM from 'react-dom';

**import** App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));

setTimeout(() => {

ReactDOM.unmountComponentAtNode(document.getElementById('app'));}, 10000);

**When time allows please read the react.js and react-dom.js.**

**Chapter 5 Form and Events**

1. Button events

import React from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

data: 'Initial data...'

}

this.updateState = this.updateState.bind(this);

};

updateState() {

this.setState({data: 'Data updated...'})

}

render() {

return (

<div>

<button onClick = {this.updateState}>CLICK</button>

<h4>{this.state.data}</h4>

</div>

);

}

}

export default App;

2. Form input events

import React from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

data: 'Initial data...'

}

this.updateState = this.updateState.bind(this);

};

updateState(e) {

this.setState({data: e.target.value});

}

render() {

return (

<div>

<input type = "text" value = {this.state.data}

onChange = {this.updateState} />

<h4>{this.state.data}</h4>

</div>

);

}

}

export default App;

**Chapter 6 Key, Refs and Router**

**6.1 Refs**

The **ref**is used to return a reference to your element. **Refs**should be avoided in most cases but they can be useful when you need DOM measurements or to add methods to your components.

Note, this can be achived by using id and getElementById.

**App.jsx,**

import React from 'react';

import ReactDOM from 'react-dom';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

data: ''

}

this.updateState = this.updateState.bind(this);

this.clearInput = this.clearInput.bind(this);

};

updateState(e) {

this.setState({data: e.target.value});

}

clearInput() {

this.setState({data: ''});

ReactDOM.findDOMNode(this.refs.myInput).focus();

}

render() {

return (

<div>

<input value = {this.state.data} onChange = {this.updateState}

ref = "myInput"></input>

<button onClick = {this.clearInput}>CLEAR</button>

<h4>{this.state.data}</h4>

</div>

);

}

}

export default App;

**6.2 Key**

React **keys**are useful when working with dynamically created components or when your lists are altered by users. Setting the **key**value will keep your components uniquely identified after the change.Let's dynamically create **Content**elements with unique index (**i**). The **map**function will create three elements from our **data**array. Since **key**value needs to be unique for every element, we will assign **i**as a key for each created element.

Actually we have seen this example before,

App.jsx

import React from 'react';

class App extends React.Component {

constructor() {

super();

this.state = {

data:

[

{

component: 'First...',

id: 1

},

{

component: 'Second...',

id: 2

},

{

component: 'Third...',

id: 3

}

]

}

}

render() {

return (

<div>

<div>

{this.state.data.map((dynamicComponent, i) => <Content

key = {i} componentData = {dynamicComponent}/>)}

</div>

</div>

);

}

}

class Content extends React.Component {

render() {

return (

<div>

<div>{this.props.componentData.component}</div>

<div>{this.props.componentData.id}</div>

</div>

);

}

}

export default App;

**6.3 Router**

**This is an extra component you have to install,**

**C:\Users\username\Desktop\reactApp>npm install react-router**

**App.jsx**

**import React from 'react';**

import ReactDOM from 'react-dom';

import { Router, Route, Link, browserHistory, IndexRoute } from 'react-router'

class App extends React.Component {

render() {

return (

<div>

<ul>

<li>Home</li>

<li>About</li>

<li>Contact</li>

</ul>

{this.props.children}

</div>

)

}

}

export default App;

class Home extends React.Component {

render() {

return (

<div>

<h1>Home...</h1>

</div>

)

}

}

class About extends React.Component {

render() {

return (

<div>

<h1>About...</h1>

</div>

)

}

}

class Contact extends React.Component {

render() {

return (

<div>

<h1>Contact...</h1>

</div>

)

}

}

export {Home,About,Contact};

**main.js**

**import** React from 'react';

**import** ReactDOM from 'react-dom';

**import** App, {Home,About, Contact} from './App.jsx';

ReactDOM.render((

<Router history = {browserHistory}>

<Route path = "/" component = {App}>

<IndexRoute component = {Home} />

<Route path = "home" component = {Home} />

<Route path = "about" component = {About} />

<Route path = "contact" component = {Contact} />

</Route>

</Router>

), document.getElementById('app'))

**This example does not work yet.**

**Chapter 7 Flux**

Flux is programming concept where the data is **uni-directional**. This data enters the app and flows through it in one direction until it is rendered on screen.

**This is simple explanation of the flux concept. In our next tutorial we will explain how to implement this into your app.**

* **Actions** − Actions are sent to dispatcher to trigger the data flow.
* **Dispatcher** − This is central hub of the app. All the data is dispatched and sent to the **stores**.
* **Store** − Store is the place where the application state and logic are held. Every store is maintaining particular state and it will update it when needed.
* **View** − The **view** will receive data from the **store** and re render the app.

Flux pattern is implemented with Redux framework,to install it, we would need,

npm install –save react-redux

The following example will show how to integrate react with redux.

Example not working yet.

**Chapter 8 Animation and Others**

8.1 installation

This is React add-on used for creating basic CSS transitions and animations. We will install it from command prompt window −

C:\Users\username\Desktop\reactApp>npm install react-addons-css-transition-group

8.2 Example

add the following to the index.html

**<link rel = "stylesheet" type = "text/css" href = "css/style.css">**

**css/style.css**

***.example-appear*** {

opacity: *0.01*;

}

*.example-appear.example-appear-active* {

opacity: *1*;

transition: *opacity* *1500ms* *ease-in*;

}

The **ReactCSSTransitionGroup**element will be used as a wrapper of the component we want to animate. You can see it is using **transitionAppear**and **transitionAppearTimeout**while **transitionEnter**and **transitionLeave**are false.

App.jsx

import React from 'react';

var ReactCSSTransitionGroup = require('react-addons-css-transition-group');

class App extends React.Component {

render() {

return (

<div>

<ReactCSSTransitionGroup transitionName = "example"

transitionAppear = {true} transitionAppearTimeout = {500}

transitionEnter = {false} transitionLeave = {false}>

<h1>My Element...</h1>

</ReactCSSTransitionGroup>

</div>

);

}

}

export default App;

8.3 High Order Compotents

missing <https://www.tutorialspoint.com/reactjs/reactjs_higher_order_components.htm>