ReactJS – Components

## **Stateless Example**

Our first component in the following example is **App**. This component is owner of **Header** and **Content**. We are creating **Header** and **Content** separately and just adding it inside JSX tree in our **App** component. Only **App** component needs to be exported.

### **App.jsx**

import React from 'react';

class App extends React.Component {

render() {

return (

<div>

<Header/>

<Content/>

</div>

);

}

}

class Header extends React.Component {

render() {

return (

<div>

<h1>Header</h1>

</div>

);

}

}

class Content extends React.Component {

render() {

return (

<div>

<h2>Content</h2>

<p>The content text!!!</p>

</div>

);

}

}

export default App;

To be able to render this on the page, we need to import it in **main.js** file and call **reactDOM.render()**. We already did this while setting the environment.

### **main.js**

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

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

The above code will generate the following result.



## **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.

It can be seen that we are using EcmaScript 2015 arrow syntax (**⇒**) which looks much cleaner than the old JavaScript syntax. This will help us create our elements with fewer lines of code. It is especially useful when we need to create a list with a lot of items.

### **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**

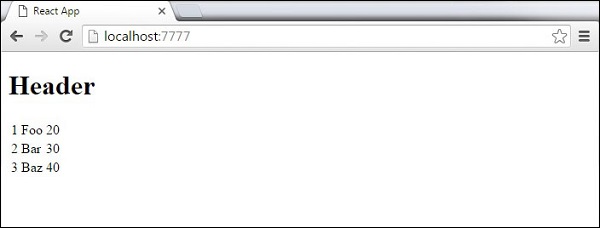
import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

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

**Note** − Notice that we are using **key = {i} inside map()** function. This will help React to update only the necessary elements instead of re-rendering the entire list when something changes. It is a huge performance boost for larger number of dynamically created elements.



# ReactJS – State

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

## **Using Props**

The following sample code shows how to create a stateful component using EcmaScript2016 syntax.

### **App.jsx**

import React from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

header: "Header from state...",

content: "Content from state..."

}

}

render() {

return (

<div>

<h1>{this.state.header}</h1>

<h2>{this.state.content}</h2>

</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'));

# ReactJS - Props Overview

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

## **Using Props**

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

This will produce the following result.



## **Default Props**

You can also set default property values directly on the component constructor instead of adding it to the **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'));

Output is the same as before.



## **State and Props**

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

### **App.jsx**

import React from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

header: "Header from props...",

content: "Content from props..."

}

}

render() {

return (

<div>

<Header headerProp = {this.state.header}/>

<Content contentProp = {this.state.content}/>

</div>

);

}

}

class Header extends React.Component {

render() {

return (

<div>

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

</div>

);

}

}

class Content extends React.Component {

render() {

return (

<div>

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

The result will again be the same as in the previous two examples, the only thing that is different is the source of our data, which is now originally coming from the **state**. When we want to update it, we just need to update the state, and all child components will be updated. More on this in the Events chapter.

# React Props Example

# ReactJS - Props Validation

## **Validating Props**

In this example, we are creating **App** component with all the **props** that we need. **App.propTypes** is used for props validation. If some of the props aren't using the correct type that we assigned, we will get a console warning. After we specify validation patterns, we will set **App.defaultProps**.

### **App.jsx**

import PropTypes from 'prop-types';

import React from 'react';

import ReactDOM from 'react-dom';

class App extends React.Component {

render() {

return (

<div>

<h1> Hello, {this.props.name} </h1>

<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>

</div>

);

}

}

App.propTypes = {

name: PropTypes.string,

propArray: PropTypes.array.isRequired,

propBool: PropTypes.bool.isRequired,

propFunc: PropTypes.func,

propNumber: PropTypes.number,

propString: PropTypes.string,

};

App.defaultProps = {

name: 'Tutorialspoint.com',

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

propBool: true,

propFunc: function(e) {

return e

},

propNumber: 1,

propString: "String value..."

}

export default App;

### **main.js**

import React from 'react';

import PropTypes from 'prop-types';

import ReactDOM from 'react-dom';

import App from './App.jsx';

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

### **webpack.config.js**

var config = {

entry: './main.js',

output: {

path:'/',

filename: 'index.js',

},

devServer: {

inline: true,

port: 8080

},

externals: {

'react': 'React'

},

module: {

loaders: [

{

test: /\.jsx?$/,

exclude: /node\_modules/,

loader: 'babel-loader',

query: {

presets: ['es2015', 'react']

}

}

]

}

}

module.exports = config;

Since all **props** are valid, we will get the following result.

As can be noticed, we have use **isRequired** when validating **propArray** and **propBool**. This will give us an error, if one of those two don't exist. If we delete **propArray:** **[1,2,3,4,5]** from the **App.defaultProps** object, the console will log a warning.

React Props Validation Error

If we set the value of **propArray: 1**, React will warn us that the propType validation has failed, since we need an array and we got a number.

# React Props Validation Error 2