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 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

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

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

**setState()** method is used for updating the state of the component. This method will not replace the state but only add changes to original state.

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;

Sometimes you want to update the component manually. You can achieve this by using **forceUpdate()** method.

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;

For DOM manipulation, we can use **ReactDOM.findDOMNode()** method. First we need to import **react-dom**.

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';

ReactDOM.findDOMNode(myDiv).hidden = true;

}

render() {

return (

<div>

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

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

</div>

);

}

}

export default App;

## 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.

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;

## Simple Example

In example below we are setting input form with **value = {this.state.data}**. This allow us to update state whenever input value changes. We are using **onChange** event that will watch input changes and update state accordingly.

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;

This is simple example where we only use one component. We are just adding **onClick** event that will trigger **updateState** function once the button is clicked.

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;

## Using Keys

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.

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;

Main

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.jsx';

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

Router

npm install react-router

In this step we are creating four components. The **App** component will be used as a tab menu. The other three components (**Home**), (**About**) and (**Contact**) are rendered once the route has changed.

### main.js

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>

)

}

}

export default Home;

class About extends React.Component {

render() {

return (

<div>

<h1>About...</h1>

</div>

)

}

}

export default About;

class Contact extends React.Component {

render() {

return (

<div>

<h1>Contact...</h1>

</div>

)

}

}

export default Contact;