Hyperiondev

TASK

React - Managing States

Visit our website

Introduction

WELCOME TO THE REACT - MANAGING STATES TASK!

We have already briefly discussed using state in React. However, there is a lot more to learn about this topic. In this task, we will answer the following questions about states: How do we keep the state of two or more components synced? When should you use an external state store (like Redux) to handle state?

A FEW REMINDERS ABOUT STATE MANAGEMENT WITH REACT

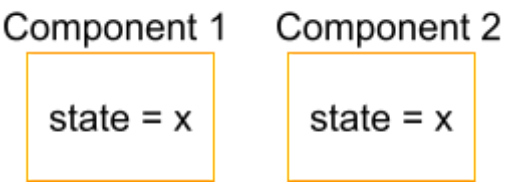
Remember that with React, you *use state when some data associated with a component changes over time*. For example, you would need to store information about whether a checkbox is checked or not in its state. In other words, if a component needs to change one of its attributes over time, that attribute should be part of its state.

Here are a few more important things you need to know about state:

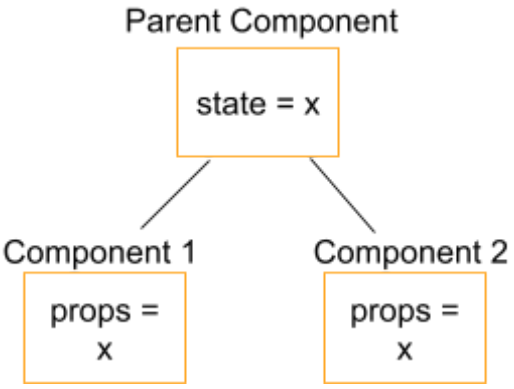
- Only components defined as classes can have state, i.e. components implemented using functions instead of a class cannot have state.
- A component can change its state (unlike props which cannot change).
- State is private to its component. State is not accessible to any component other than the one that owns and sets it. However, a component can choose to pass info stored in its state down to children using props.
- Have as few components that use state (stateful components) as possible in your application. Using too much state will make a program more complex and difficult to debug and maintain.
- The state is user-defined, and it should be a plain JavaScript object.

KEEPING THE STATE OF TWO COMPONENTS SYNCED

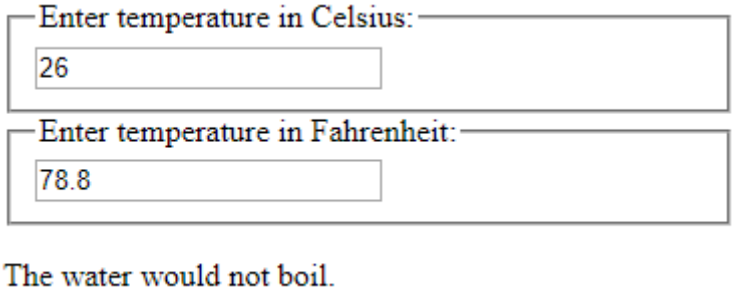
As you create more UIs, you are bound to want the state of two or more components to be synced at some stage. In other words, if the value in component 1 changes, you want the value in component 2 to change too.



In React, we accomplish this by *lifting state* up. This means that if two or more components need to have the same state at any given time, we store the state in the nearest shared parent component instead of in the separate children components. We then pass the value we want to keep synced down as props to the child components.



Let's look at an example of how this is done. In this example (provided by React), we create two components: a Calculator component and a TemperatureInput component. The Calculator component contains two TemperatureInput components (for inputting temperature in degrees Celsius or Fahrenheit) as shown below:



When a user enters a value in either of the two `TemperatureInput` components, the value in the other `TemperatureInput` component should automatically change. To implement this, follow these steps:

Step 1: Make the parent component a stateful component.

In this example, the Calculator component is the parent container since it contains the two TemperatureInput components. The Calculator component must, therefore, be a stateful component. You have already learnt how to create a stateful component.

The code for this example is shown below. This example is explained in detail **here**.

```
class Calculator extends React.Component {
   constructor(props) {
       super(props);
       this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
       this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
       this.state = {temperature: '', scale: 'c'};
   }

   handleCelsiusChange(temperature) {
       this.setState({scale: 'c', temperature});
   }

   handleFahrenheitChange(temperature) {
       this.setState({scale: 'f', temperature});
   }

   render() {
       const scale = this.state.scale;
       const temperature = this.state.temperature;
       const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) :
temperature;
       const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit)
: temperature;

       return (
           <div>
               <TemperatureInput
                   scale="c"
                   temperature={celsius}
                   onTemperatureChange={this.handleCelsiusChange} />

               <TemperatureInput
                   scale="f"
                   temperature={fahrenheit}
                   onTemperatureChange={this.handleFahrenheitChange} />

               <BoilingVerdict
                   celsius={parseFloat(celsius)} />

           </div>
       );
   }
}
```

Step 2: In the parent component, create and register event handlers that will respond to events triggered in the child components - i.e. the `handle_Change()` functions. These event handlers are used to change the state using `this.setState()`.

Step 3: In the children components (the `TemperatureInput` components), let the event handler that is triggered by the event that changes the state of the parent container respond by calling the event handler in the parent component.

```
class TemperatureInput extends React.Component {
   constructor(props) {
       super(props);
       this.handleChange = this.handleChange.bind(this);
   }

   handleChange(e) {
       this.props.onTemperatureChange(e.target.value);
   }

   render() {
       const temperature = this.props.temperature;
       const scale = this.props.scale;
       return (
           <fieldset>
               <legend>Enter temperature in {scaleNames[scale]}:</legend>
               <input value={temperature}
                      onChange={this.handleChange} />
           </fieldset>
       );
   }
}
```

Here, `onTemperatureChange` is both a function and a property (prop). It is a function from the parent passed to the child as a property.

This is an important concept and we recommend that you take the time to work through **this official React guide** or see this example explained in more detail **here**.

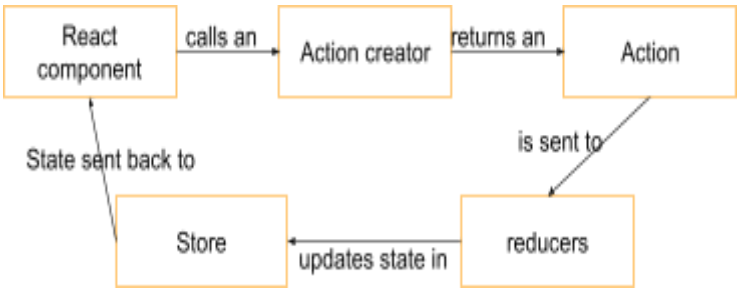
REDUX

Redux is a state container that can be used with React (and other view libraries) to assist with state management. The maker of Redux, Dan Abramov, recommends that you create React apps without Redux where possible and only use Redux when absolutely necessary. You can solve most frontend problems without Redux, and therefore you are not required to use it. If you are interested, read on to see how it works.

How does Redux work?

As stated earlier, Redux is a state container. It stores all state information for an app in one central place. The state information is stored in the store and the entire state of the application is stored in a single object known as the state tree.

We will now explain how state information is managed with Redux. A React component can call an *action creator* which creates an *action*. An action is an object that describes the change that is required. The action is sent to a *reducer*, which is a function that takes the previous state and the change (described in the action) and returns the next state. This is updated in the store and sent to the React component. When the React component's state is changed, it re-renders.



To learn more about Redux, you can do this **free tutorial** by the creator of Redux or visit the official Redux page **here** to see how to get started.

Compulsory Task 1

Create a React app that simulates the **Monty Hall** problem.

- The user should be shown three doors that they can click. One of the three doors is a winning door, and the other two are losing doors. I.e., the user wishes to pick the winning door, but which one is unknown (random). You can put a pot of gold behind the winning door, or whatever you choose as the prize.
- When the user clicks on a door it should be marked selected. It should not be opened yet.
- Following this, one of the other two doors that is a losing door should be opened. I.e. at this point one door is selected, one losing door is open, and one door is closed.
- The user must now be allowed to open either one of the two closed doors. This is their final decision.
- Their chosen door should open to reveal whether it is the winning door or not.
- The user must be able to reset the game at any time and start over.

(Fun fact) If you program the probabilities correctly (winning door is random each play) the winning door will be the opposite of the initially selected door about two thirds of the time. Many statisticians are still dumbfounded by this. Test it out to see if it feels true.

As per usual, submit everything except the `node_modules` dir.

Things to look out for:

1. Create a components directory inside your src directory to contain all the components you create.

2. Remember, each component should be stored as a separate .js file inside the component directory and each of these files should start with a capital letter.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCE:

React.js. (2020). Getting Started – React. Retrieved 6 August 2020, from **https://reactjs.org/docs/getting-started.html**