Props & State

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Props
- State
- setState()
- Event handler
- Passing data to another component via props
- Passing method references between components

Props

- React component is just a function that has only one parameter “prop”. Prop is an object that can have many properties. Developers directly destructure it.
- React takes advantage of the JS language that you can include anything in the prop such as another component (function), or event handler.
- Props (properties) are arguments passed to components.
- Can be used to pass data from the parent component to child components. One way, parent to child.
- They are passed via HTML properties.
- Props are read-only!
- “Props are parameters from the parent.”

Props in Functional Component

advancedHello.js

```
function AdvancedHello(props) {
   return (<h1>Hello {props.name} </h1>);
}

export default AdvancedHello;
```

App.js

```
function App() {
 return (
   <div className="App">
    <AdvancedHello name='Umur'></AdvancedHello>
    <AdvancedHello name='Unubold'></AdvancedHello>
   </div>
 );
}
```

Props in Class-Based Component

advancedGreeting.js

```
import React from 'react';

class AdvancedGreeting extends React.Component
{
   render() {
       return <h1>Hello {this.props.name}</h1>
   };
}

export default AdvancedGreeting;
```

App.js

```
function App() {
 return (
   <div className="App">
     <AdvancedGreeting name='Erol'></AdvancedGreeting>
     <AdvancedGreeting name='Ayse'></AdvancedGreeting>
   </div>
 );
}
```

Children Prop

Children is a special property of React components which contains any child elements defined within the component. You can achieve composition (advanced technique to reuse code) using the children prop.

```
class AdvancedGreeting extends React.Component {
    render() {
        return <h1>Hello {this.props.name} {this.props.children}</h1>
    };
}

export default AdvancedGreeting;
```

Children Prop

```
function App() {
 return (
   <div className="App">
     <AdvancedGreeting name='Erol'>How are you?</AdvancedGreeting>
     <AdvancedGreeting name='Ayse'>We all miss you</AdvancedGreeting>
   </div>
 );
}
```

Advantages of one-way data flow

- **Debugging** – One-way data flow makes debugging much easier. The developer knows where the data is coming from and where it is going.
- Less prone to errors
- More efficient

Is it possible to change the data from the parent in the child? Yes, pass down the method in the parent that changes the data.

State

State is used to change the component. Changes to state trigger an UI update.

Only **class-based** components can define and use state. State is a property of the class component. We can still use state in functional components using something called **hooks**.

The state object is where you store property values that belongs to the component. You can store as many properties as needed.

“State is just a variable that triggers UI to update”.

Changing State

- To change a value in the state object, always use the **this.setState()** method! We don’t directly mutate state.
- this.setState will ensure that the component knows it’s been updated and then calls the render() method.
- With setState component's render function will be called, **also all its subsequent child components will re-render, regardless of whether their props have changed or not.**

props vs state

- props get passed to the component whereas state is managed within the component
- props is similar to function parameters whereas state is similar to variables declared within a function.

Increment Counter Example

```
class App extends React.Component {
 state={counter:1};
 incrementCounter = () => {
   this.setState({counter:this.state.counter + 1});
 }
 render() {
   return (
     <div className="App">
       <input type='label' value={this.state.counter} />
       <button onClick={this.incrementCounter}>Increment
Counter</button>
…
```

setState

- Do not change the state directly.
- We **have to use setState** to manipulate the state.
- Recap incrementCounter:

```
class App extends React.Component {
 state={counter:1};
 incrementCounter = ()=>{
   this.setState({counter: this.state.counter + 1});
 }
}
```

Handling Events

Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:

- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

```
<button onClick={this.activateLasers}>
  Activate Lasers
</button>
```

Adding event handler - public class fields syntax

```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
…
```

Adding event handler - arrow function in the callback

```
class LoggingButton extends React.Component {
  handleClick(e) {
    console.log('this is:', this);
  }

  render() {
    // You can pass parameters with this syntax.
    return (
      <button onClick={(e) => this.handleClick(e)}>
        Click me
      </button>
…
```

Merging State – Make older example

- Put another attribute to the state.
- Let’s call this attribute ‘dummy’
- We manipulate the state with useState method.
- ‘dummy’ attribute will be in the new state. When you call setState, it only merges the new data into the existing state. Other states that have not been changed will not be lost.

Merging State and Make Older

```
state = {
   students: [
     { name: 'Alice', age: 20 },
     { name: 'Bob', age: 19 }
   ],
   dummy : 'dummy value'
 };
 makeOlder = () => {
   this.setState({
     students: [
       { name: 'Alice', age: 25 },
       { name: 'Bob', age: 23 }
     ]
   });
   console.log(this.state); // Dummy is
still there
 }
```

```
 render() {
   return (
     <div className="App">
       <Student name={this.state.students[0].name}
         age={this.state.students[0].age}>
       </Student>
       <Student name={this.state.students[1].name}
         age={this.state.students[1].age}>
       </Student>
       <button onClick={this.makeOlder}>Make
Older</button>
     </div>
   );
 }
```

useState hook

- Returns an array with 2 elements.
  - The state as the first element via which you can access the state value.
  - Sets the state and triggers the UI update.
- Pass the default value for the state as a param to the useState hook.

```
const [count, setCount] = useState(0);
```

Learn more: using the state hook

Passing Data to Another Component via Props

```
state={students:[
    {name:'Alice', age:20},
    {name:'Bob', age:19}
]};
render() {
  return (
    <div className="App">
      <Student name={this.state.students[0].name}
               age={this.state.students[0].age}>
      </Student>
      <Student name={this.state.students[1].name}
               age={this.state.students[1].age}>
      </Student>
    </div>);
}
```

Passing Method References Between Components

- No difference between passing other properties like name and passing method reference.
- In makeOlder example, we change the age when clicking on the button.
- But, what if we want to change the age when clicking on the name or age?

Passing Method References Between Components

```
render() { // App.js
   return (
     <div className="App">
       <Student
name={this.state.students[0].name}
         age={this.state.students[0].age}
         myClickHandler={this.makeOlder}>
       </Student>
       <Student
name={this.state.students[1].name}
         age={this.state.students[1].age}
         myClickHandler={this.makeOlder}>
       </Student>
       <button
onClick={this.makeOlder}>Make
Older</button>
     </div>
   );
 }
```

```
//student.js
class Student extends React.Component {
   render() {
       return (
           <div>
               <p onClick={this.props.myClickHandler}>I
am {this.props.name}.</p>
               <p>I am {this.props.age} years old.</p>
           </div>
       )
   };
}
```