



前端網站開發人員證書課程
(二) 進階網絡程式設計--專業React.js應用

4. State and Props

Presented by Krystal Institute



Lesson Outline

- Introduce the Properties and their purpose
- How to use props in react app in different ways
- The state and how it can be used with functional and class components in react

4.1 Props

4.1.1 Props

- Props stand for "Properties." They are read-only components.
- Props are object which stores the value of attributes of a tag and works similarly to the HTML attributes.
- Props give a way to pass data from one component to other components.
- Props are immutable so we cannot modify the props from inside the component.

4.1.2 Props in Class Components

- Inside the components, we can add attributes called props. These props can be accessed in the component using `this.props` and it can be used in the child component.
- Let's look into the Expense Tracker app we created and enhanced in our last session.
- Open the app folder and open the app in Visual Studio code.

```
C:\workspace\react tutorial\create react app\class-component>code .
```


4.1.2 Props in Class Components

- Run the app

```
C:\workspace\react tutorial\create react app\class-component>npm start
```

```
> class-component@0.1.0 start
```

```
> react-scripts start
```

```
(node:10272) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
```

```
(Use `node --trace-deprecation ...` to show where the warning was created)
```

```
(node:10272) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
```

```
Starting the development server...
```

4.1.2 Props in Class Components

- Let's modify the hardcoded values in the class component into the props.

```
import React from 'react';
class ExpenseEntryItem extends React.Component {
  render() {
    return (<div className=" expense-item-container">
      <div className="card">
        <div class="card-header">Mango Juice</div>
        
        <div className="card-body">
          <div><b>Item:</b> <em>Mango Juice</em></div>
          <div><b>Amount:</b> <em>30.00</em></div>
          <div><b>Spend Date:</b> <em>2020-10-10</em></div>
          <div><b>Category:</b> <em>Food</em></div>
        </div>
      </div>
    </div>
  );
}
```

4.1.2 Props in Class Components

- Switch to the App.js file and pass the entry items as parameters/arguments while rendering the class component.

```
<div className='col-md-6 col-xs-12' >  
  <ExpenseEntryItem item="Mango Juice" amount="30.00" date="2020-10-10" category="Food" />  
</div>
```

- Receive Props: Switch to the component file "ExpenseEntryItem.js" and replace the hardcoded values with values passed as props.
- Verify the output it should remain the same.

```
render() {  
  return [

{this.props.item}





<b>Item:</b> <em>{this.props.item}</em></div>

<b>Amount:</b> <em>{this.props.amount}</em></div>

<b>Spend Date:</b> <em>{this.props.date}</em></div>

<b>Category:</b> <em>{this.props.category}</em></div>

  
];  
}


```


4.1.2 Props in Class Components

- The props variables can also be destructured from `this.props` and can be directly used inside the component

```
render() {  
  const { item, amount, date, category } = this.props;  
  
  return (

<div className="card">  
      <div className="card-header">{item}</div>  
        
        <div><b>Item:</b> <em>{item}</em></div>  
        <div><b>Amount:</b> <em>{amount}</em></div>  
        <div><b>Spend Date:</b> <em>{date}</em></div>  
        <div><b>Category:</b> <em>{category}</em></div>  
      </div>  
    </div>  
  </div>);  
}


```

- Verify the output it should remain the same.

4.1.2 Props in Class Components

- The props variables can also be destructured from `this.props` and can be directly used inside the component

```
render() {  
  const { item, amount, date, category } = this.props;  
  
  return (

<div className="card">  
      <div className="card-header">{item}</div>  
        
        <div><b>Item:</b> <em>{item}</em></div>  
        <div><b>Amount:</b> <em>{amount}</em></div>  
        <div><b>Spend Date:</b> <em>{date}</em></div>  
        <div><b>Category:</b> <em>{category}</em></div>  
      </div>  
    </div>  
  </div>);  
}


```

- Verify the output it should remain the same.

4.1.3 Props in Function Components

- In a function component, components receive props exactly like an ordinary function receive parameters.
- A function component will receive the props object with properties you described in the component call.
- Let's modify the hardcoded values in the function component into the props.

```
function ExpenseEntryItem1() {  
  return (<div className="expense-item-container">  
    <div className="card">  
      <div class="card-header">Apple Juice</div>  
        
        <div><b>Item:</b> <em>Apple Juice</em></div>  
        <div><b>Amount:</b> <em>50.00</em></div>  
        <div><b>Spend Date:</b> <em>2020-10-10</em></div>  
        <div><b>Category:</b> <em>Food</em></div>  
      </div>  
    </div>  
  </div>  
  );  
}
```

4.1.3 Props in Function Components

- Switch to the App.js file and pass the entry items as props while rendering the function component.

```
<div className='col-md-6 col-xs-12' >  
  <ExpenseEntryItem1 item="Apple Juice" amount="50.00" date="2020-10-10" category="Food" />  
</div>
```

- Receive Props: Receive the props as an argument as we do for JavaScript functions.

```
function ExpenseEntryItem1(props) {
```


4.1.3 Props in Function Components

- Accessing Props: Switch to the component file “ExpenseEntryItem1.js” and replace the hardcoded values with {props.property_name}

```
return (<div className="expense-item-container">
  <div className="card">
    <div className="card-header">{props.item}</div>
    
      <div><b>Item:</b> <em>{props.item}</em></div>
      <div><b>Amount:</b> <em>{props.amount}</em></div>
      <div><b>Spend Date:</b> <em>{props.date}</em></div>
      <div><b>Category:</b> <em>{props.category}</em></div>
    </div>
  </div>
</div>
);
```

- Verify the output it should remain the same.

4.1.3 Props in Function Components

- The props can also be destructured inside the component and can be directly used inside the component.

```
function ExpenseEntryItem1(props) {  
  const { item, amount, date, category } = props;  
  return (<div className="expense-item-container">  
    <div className="card">  
      <div className="card-header">{item}</div>  
        
        <div><b>Item:</b> <em>{item}</em></div>  
        <div><b>Amount:</b> <em>{amount}</em></div>  
        <div><b>Spend Date:</b> <em>{date}</em></div>  
        <div><b>Category:</b> <em>{category}</em></div>  
      </div>  
    </div>  
  </div>  
);
```

- Verify the output it should remain the same.

4.1.4 Default Props

- Switch to the App.js file and remove the props passed while rendering the component.

```
<div className='col-md-6 col-xs-12' >  
  <ExpenseEntryItem1/>  
</div>
```

- Verify the output it should remain the same.

4.2 State

4.2.1 State

- The state is an updatable structure that is used to contain data or information about the component.
- The state in a component can change over time.
- The change in state over time can happen as a response to user action or system events.
- A component with a state is known as a stateful component. It is the heart of the react component which determines the behavior of the component and how it will render.
- They are also responsible for making a component dynamic and interactive.
- A state must be kept as simple as possible.
- Creating and defining a state is different for function and class components.

4.2.2 Difference between Props and State

Props	State
Props are immutable i.e. once set the props cannot be changed	State is an observable object that is to be used to hold data that may change over time and to control the behavior after each change.
Props are set by the parent component	State is generally updated by event handlers.

4.2.3 State in Function Component

- React V16.8 and later versions allow us to use hooks to create and update *state* in functional components.
- The *useState()* hook sets up individual state property. It returns an array containing two elements: the **current state value**, and a **function** you can call with a new value to **update** the state.
- The state update function is just a regular function. It's used in the *event* handlers or any other user interaction to update the current state value. React's internal handling of state values ensures your component will then be re-rendered when the *state* is updated. *useState()* will supply the new value, causing the state change to be effected.

4.2.4 State in Class Component

- The state can be created inside the class component with the keyword “state” as an object
`state = {name: "World" };`
- The state can be accessed as `{this.state.name}`
- `this.setState()` is used to change the value of the state object

4.2.4 State in Class Component

The `setState()` method:

- The state can be updated in response to event handlers, server responses, or prop changes. This is done using the `setState()` method.
- The `setState()` method enqueues all of the updates made to the component state and instructs React to re-render the component and its children with the updated state.
- Always use the `setState()` method to change the state object, since it will ensure that the component knows it's been updated and calls the `render()` method.

4.2.5 Create State in a Component

- As we are clear with the class and function components now. Let's remove any one component in our App as they both do the same work.
- Switch to the App.js file and remove the second rendered component.

```
function App() {  
  return (  
    <div className="row text-center">  
      <div className='col-md-6 col-xs-12' >  
        <ExpenseEntryItem item="Mango Juice" amount="30.00" date="2020-10-10" category="Food" />  
      </div>  
    </div>  
  );  
}
```


4.2.5 Create State in a Component

- View the output, It should have only one card:



4.2.5 Create State in a Component

- Let's create a state in the App component which is a function component.
- Import useState from React.

```
✓ import { useState } from 'react';
```

- Create a state for the item name, where the initial value is "Mango Juice".

```
function App() {  
  const [itemName, setItemName] = useState("Mango Juice");  
  return (  
    <div className="row text-center">  
      <div className='col-md-6 col-xs-12' >  
        <ExpenseEntryItem item="Mango Juice" amount="30.00" date="2022-01-01">  
        </div>  
      </div>  
    );  
  }  
}
```

4.2.5 Create State in a Component

- Replace the props value with the state variable "itemName"

```
<div className='col-md-6 col-xs-12' >  
  <ExpenseEntryItem item={itemName} amount="30.00" date="2020-10-10" category="Food" />  
</div>
```

- Verify the output it should remain the same.

```
import { useState } from 'react';  
import logo from './logo.svg';  
import './App.css';  
import ExpenseEntryItem from './components/ExpenseEntryItem';  
import ExpenseEntryItem1 from './components/ExpenseEntryItem1';  
  
function App() {  
  const [itemName, setItemName] = useState("Mango Juice");  
  return (  
    <div className="row text-center">  
      <div className='col-md-6 col-xs-12' >  
        <ExpenseEntryItem item={itemName} amount="30.00" date="2020-10-10" category="Food" />  
      </div>  
    </div>  
  );  
}  
  
export default App;
```

4.2.6 Update State

- Let's add some additional features to our app. Add buttons to increase and decrease the count of the item i.e. Mango Juice.
- Switch to the component(ExpenseEntryItem.js), and create buttons in a header to increase and decrease the count.

```
<div className="card-header">  
  <button className="btn" >-</button>  
  {item}  
  <button className="btn" >+</button>  
</div>
```

4.2.6 Update State

- View the output, the header should have buttons.



4.2.6 Update State

- Switch to the App.js file and add a new state that can be updated on the click of buttons i.e. itemCount and set the initial value as 1.

```
function App() {  
  const [itemName, setItemName] = useState("Mango Juice");  
  const [itemCount, setItemCount] = useState(1);
```

- Pass the state as a prop to the component.

```
<div className='col-md-6 col-xs-12' >  
  <ExpenseEntryItem item={itemName} amount="30.00" date="2020-10-10" category="Food" itemCount={itemCount} setItemCount={setItemCount}/>  
</div>  
div>
```

- Switch to the component(ExpenseEntryItem.js).

4.2.6 Update State

- Receive the new props in the destructure statement.

```
const { item, amount, date, category, itemCount, setItemCount } = this.props;
```

- Display the count in the UI.

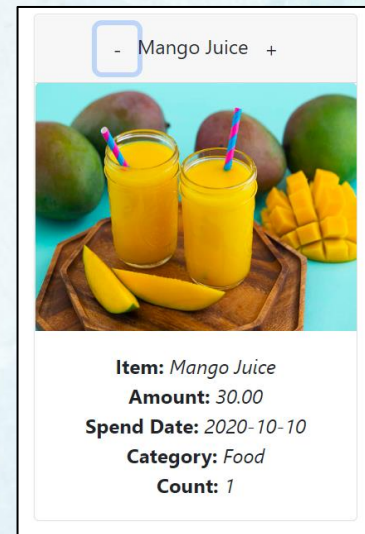
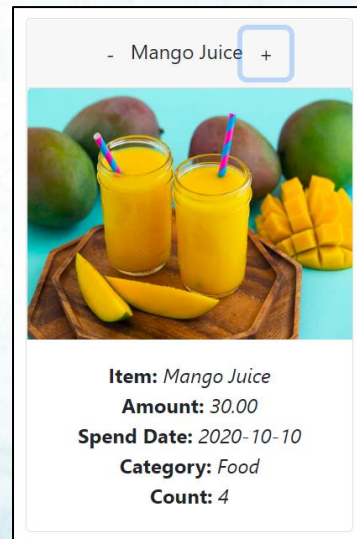
```
<div><b>Item:</b> <em>{item}</em></div>  
<div><b>Amount:</b> <em>{amount}</em></div>  
<div><b>Spend Date:</b> <em>{date}</em></div>  
<div><b>Category:</b> <em>{category}</em></div>  
<div><b>Count:</b> <em>{itemCount}</em></div>
```

4.2.6 Update State

- Update the count on the click of the buttons.

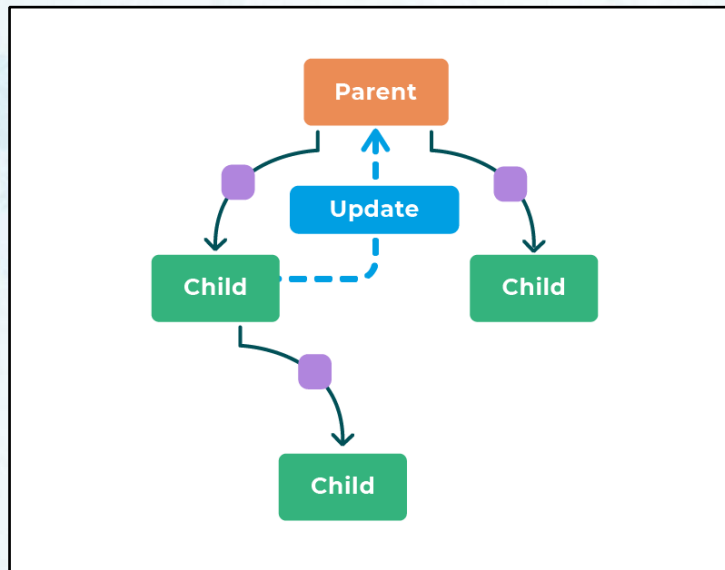
```
<div className="card-header">  
  <button onClick={() => setItemCount(itemCount - 1)} className="btn" >-</button>  
  {item}  
  <button onClick={() => setItemCount(itemCount + 1)} className="btn" >+</button>  
</div>
```

- View the output, and verify “updateState” by clicking the buttons. The count value in the body should be updated with respect to button clicks.



4.2.7 Share State between Different Components

- Components may have parent-child relations and there might be a hierarchy of children also.



4.2.7 Share State between Different Components

Parent-child:

- Pass information down in props to the component that needs it.

```
<div><b>Item:</b> <em>{item}</em></div>
<div><b>Amount:</b> <em>{amount}</em></div>
<div><b>Spend Date:</b> <em>{date}</em></div>
<div><b>Category:</b> <em>{category}</em></div>
<div><b>Count:</b> <em>{itemCount}</em></div>
```

Child to Parent:

- It's very easy to update the state of the parent from any child, you just have to pass a callback function as props to that child and whenever it needs to update the state of the parent, the function will be called.

```
<div className="card-header">
  <button onClick={() => setItemCount(itemCount - 1)} className="btn" >-</button>
  {item}
  <button onClick={() => setItemCount(itemCount + 1)} className="btn" >+</button>
</div>
```


4.3 React Events

4.3.1 React Events

- An event is an action that could be triggered as a result of the user action or a system-generated event.
- For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.
- React has its own event handling system which is very similar to handling events on DOM elements.
- The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.

4.3.2 Handling Events

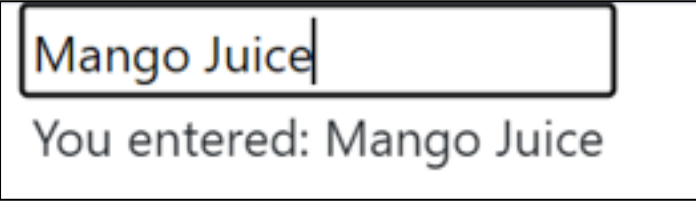
Handling events with react have some syntactic differences from handling events on DOM.

These are:

- React events are named as camelCase instead of lowercase.
- With JSX, a function is passed as the event handler instead of a string.

Example:

Let's add an input text box and add an onChange event. This event will return the value entered in the text box. And let's display the text below the text box.



Mango Juice

You entered: Mango Juice

4.3.2 Handling Events

Steps:

- Switch to the App.js file and create a state for the user name.

```
const [userName, setUserName] = useState("");
```

- Create an input text box just above the component render.

```
<div className='col-md-6 col-xs-12' >  
  <input type="text" required={true} />  
  <ExpenseEntryItem item={itemName} amount="30.00"  
    {setItemCount}/>  
</div>
```

- Create a function to handle on change event and set the value as the userName.

```
const handleChange = (eventArg) => {  
  setUserName(eventArg.target.value);  
}
```


4.3.2 Handling Events

Steps:

- Invoke the function on the change event of the input text box.

```
<input type="text" onChange={handleChange} required={true} />
```

- Display the user name below the text box.

```
<p>You entered: {userName}</p>
```

- View the output, any text entered in the text box will be shown below it on the change event.



4.4 Thinking in Components

4.4 Thinking in Components

React community has provided a direction on how to think in React way and build big, fast, and scalable applications.

- Step 1: Creating a simple mock service.
- Step 2: Break the functionality into smaller components.
- Step 3: If possible start by making a simple static version.
- Step 4: Identifying the Minimal representation of UI state.
- Step 5: Identify where the state should live.
- Step 6: Add two-way data flow if required.

4.4 Thinking in Components

Creating a simple mock service:

- If we need to make a server call and fetch data.
- We can create a mock service to start with and build a component to fetch and display data.
- Here we can include the processing of JSON in components and evaluate the expected result.

Break the functionality into smaller components:

- The first Thing React suggest is to create a smaller understandable design of boxes.
- These boxes will represent the association between the different components and the passing of data flow.
- Making of components should be based on the principle of single responsibility. That means a single component should handle a single task.
- Creating a component hierarchy will make the developer's task easier.

4.4 Thinking in Components

If possible start by making a simple static version:

- With the use of mock service and smaller components we can create a static version and build the app thereon.
- Creating a static version should not use state modifications. It should play with the passage of props and rendering Ui only. Keeping one-way data flow in React makes it simple and modular
- To make it more clear the difference and use case of props and state should be understood very well.

Identifying the Minimal representation of UI state:

- For making the app interactive the state should be able to trigger from the relevant component.
- Identifying the required mutable state is necessary to build the correct app.

4.4 Thinking in Components

Identify where the state should live:

- State can be shared between the multiple child components. Lifting the state up can be used to make communication between the multiple components. Using state management libraries like Redux solves a lot of issues arising from the state.
- React strongly recommends the one-way down flow of data to components.

Add two-way data flow if required:

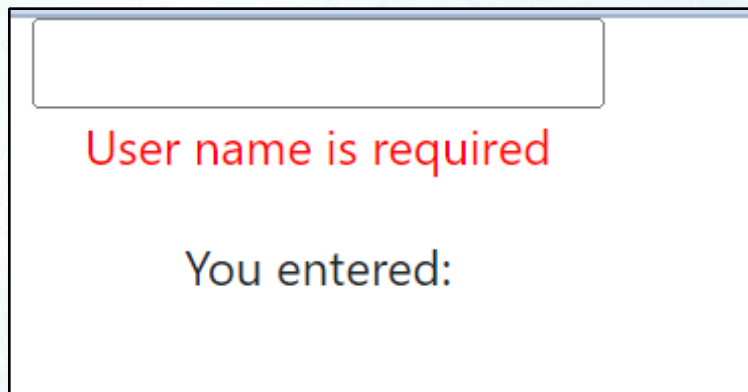
- The controlled component in form handling is the example of two-way data binding.

4.3 Assignment

4.3 Assignment

Outline:

To enhance the react app by adding the onBlur event handler to trigger the validation message for the input text box.



The diagram shows a rectangular form with a thin black border. Inside the form, at the top, is a white rectangular input field with a thin black border. Below the input field, the text "User name is required" is displayed in a red, sans-serif font. Further down, the text "You entered:" is displayed in a blue, sans-serif font.