codecademy

Stateless Components From Stateful Components

Stateful and Stateless Components

In React, a *stateful* component is a component that holds some state. *Stateless* components, by contrast, have no state. Note that both types of components can use props.

In the example, there are two React components. The `Store` component is stateful and the `Week` component is stateless.

```
class Store extends React.Component {
  constructor(props) {
    super(props);
    this.state = { sell: 'anything'
};
  }
  render() {
    return <h1>I'm selling
{this.state.sell}.</h1>;
  }
}

class Week extends React.Component {
  render() {
    return <h1>Today is
{this.props.day}!</h1>;
  }
}
```

React Programming Pattern

One of the most common programming patterns in React is to use stateful parent components to maintain their own state and pass it down to one or more stateless child components as props. The example code shows a basic example.

codecademy

```
// This is a stateless child
component.
class BabyYoda extends
React.Component {
  render() {
    return <h2>I am
{this.props.name}!</h2>;
  }
}

// This is a stateful Parent element.
class Yoda extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: 'Toyoda' };
  }

  // The child component will render
information passed down from the
parent component.
  render() {
    return <BabyYoda name=
{this.state.name} />;
  }
}
```

Changing Props and State

In React, a component should never change its own props directly. A parent component should change them.

State, on the other hand, is the opposite of props: a component keeps track of its own state and can change it at any time.

The example code shows a component that accepts a prop, `subtitle`, which never changes. It also has a state object which does change.

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { now: new Date() };
    this.updateTime
= this.updateTime.bind(this);
  }

  updateTime() {
    this.setState({ now: new Date()
});
  }

  render() {
    return (
      <div>
        <h1>It is currently
{this.state.now.toLocaleTimeString()}
</h1>
        <h2>{this.props.subtitle}
</h2>
        <button onClick=
{this.updateTime}>Update the
clock</button>
      </div>
    );
  }
}
```

Passing State Change Functions as Props

If a React parent component defines a function that changes its state, that function can be passed to a child component and called within the component to updating the parent component's state.

In this example, because `this.setState()` causes the `Name` component to re-render, any change to the `<input>` will update the `Name` component's state, causing a new render and displaying the new state value to the `<p>` tag content.

```
class Name extends React.Component {
  constructor(props) {
    super(props);
    this.state = { name: '' };
    this.handleNameChange
= this.handleNameChange.bind(this);
  }

  handleNameChange(e) {
    this.setState({
      name: e.target.value,
    });
  }

  render() {
    return (
      <div>
        <input onChange=
{this.handleNameChange} />
        <p>{this.state.name}</p>
      </div>
    );
  }
}
```

Event Handlers and State in React

Event handler functions in React are often used to update state. These handler functions often receive an event as an argument, which is used to update state values correctly.

In the example code, we use `event.target.value` to get the input's value.

```
class MyComponent extends
React.Component {
  constructor(props) {
    super(props);
    this.state = { text: '' };
    this.handleChange
= this.handleChange.bind(this);
  }

  handleChange(event) {
    this.setState({
      name: event.target.value,
    });
  }

  render() {
    return (
      <div>
        <input onChange=
{this.handleChange} value=
{this.state.text} />
        <p>You typed
{this.state.text}</p>
      </div>
    );
  }
}
```

Using Stateless Updaters and Presenters

A common React programming pattern is to use a parent stateful component to manage state and define state-updating methods. Then, it will render stateless child components.

One or more of those child components will be responsible for updating the parent state (via methods passed as props). One or more of those child components will be responsible for displaying that state.

In the example code, `StatefulParent` renders `<InputComponent>` to change its state and uses `<DisplayComponent>` to display it.

```
class StatefulParent extends
React.Component {
  constructor(props) {
    super(props);
    // Set up initial state here
    // Bind handler functions here
  }

  handlerMethod(event) {
    // Update state here
  }

  render() {
    return (
      <div>
        <InputComponent onChange=
{handler} />
        <DisplayComponent
valueToDisplay=
{this.state.valueToDisplay} />
      </div>
    );
  }
}
```