

Class Methods and Event Handling



<https://github.com/learn-co-curriculum/react-hooks-class-events>



<https://github.com/learn-co-curriculum/react-hooks-class-events/issues/new>

Learning Goals

1. Write a helper method in a React class
2. Use a method as a callback function with the arrow function syntax

Introduction

Working with classes means getting used to a slightly different syntax for organizing our component code. We'll talk about how to define **methods** on our classes, and how to make sure that our callback functions are defined so that we can access the component data via the `this` keyword in those methods.

Defining Methods

In a typical function component, we can create helper functions *within* the component so that those functions have access to data that is in the scope of our component. For example:

```
function TodoList(props) {
  function displayTodos() {
    return props.todos.map((todo) => <li key={todo.id}>{todo.text}</li>);
  }

  return <ul>{displayTodos()}</ul>;
}
```

Since this `displayTodos` function is defined **within** the `TodoList` component, it has access to all the data in the component (including props) via its outer scope.

With a class component, we can define **methods** that will give us similar functionality:

```
class TodoList extends React.Component {  
  // NOTE: no function keyword before the method name!  
  displayTodos() {  
    return this.props.todos.map((todo) => <li key={todo.id}>{todo.text}</li>);  
  }  
  
  render() {  
    return <ul>{this.displayTodos()}</ul>;  
  }  
}
```

In this example, both `render` and `displayTodos` are **methods** on our component. To call the `displayTodos` method from the `render` method, we must write `this.displayTodos()`.

Instead of having access to shared data via props, these two methods can share data through `context`: both have the access to the same `this` object. So calling `this.props` in the `render` method will give us access to the same data as calling `this.props` in the `displayTodos` method.

We could also have written the same functionality all inside the `render` method:

```
render() {  
  function displayTodos() {  
    return this.props.todos.map((todo) => <li key={todo.id}>{todo.text}</li>);  
  }  
  
  return <ul>{displayTodos()}</ul>;  
}
```

However, it's a common practice to define functions (like `displayTodos` here) as their own standalone methods.

Classes With Event Handlers

When writing class components, it's a common practice to define an event handler as a method within the class. Consider this example:

```
class Clicker extends React.Component {  
  handleClick(event) {  
    console.log(`#${event.target.name} was clicked`);  
  }  
  
  render() {  
    return (  
      <div>  
        <button name="one" onClick={this.handleClick}>  
          One  
        </button>  
        <button name="two" onClick={this.handleClick}>  
          Two  
        </button>  
      </div>  
    );  
  }  
}
```

Here, both buttons are using the `handleClick` method as a callback function. So far, our code works as expected: clicking the button will run the callback function, and we'll see the name of the button being logged.

However, if we need to access the value of `this` inside of `handleClick`, we'll see something unexpected:

```
class Clicker extends React.Component {  
  handleClick(event) {  
    console.log(this); // => undefined
```

}

```
render() {
  console.log(this.props); // => { message: "hi" }
  return (
    <div>
      <button name="one" onClick={this.handleClick}>
        One
      </button>
      <button name="two" onClick={this.handleClick}>
        Two
      </button>
    </div>
  );
}
}

// passing in props
ReactDOM.render(<Clicker message="hi" />, document.querySelector("#root"));
```

Due to some of the [complex rules of the `this` keyword](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this>), when we use a class method as a callback function, **we will no longer have access to our component instance via the `this` keyword.**

In order to keep access to `this` inside of our event handler, we have several options:

1. Use an arrow function to **define** the event handler method:

```
class Clicker extends React.Component {
  handleClick = () => {
    console.log(this.props); // => { message: "hi" }
  };

  render() {
```

```
        return <button onClick={this.handleClick}>One</button>;
    }
}
```

This is the approach you'll see most commonly!

2. Use an arrow function to invoke the event handler method:

```
class Clicker extends React.Component {
  handleClick() {
    console.log(this.props); // => { message: "hi" }
  }

  render() {
    return <button onClick={() => this.handleClick()}>One</button>;
  }
}
```

Using an arrow function here works because we are invoking the `handleClick` method **on** the `this` object, and **arrow functions don't create their own context**.

3. Bind the event handler explicitly:

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

  handleClick(event) {
    console.log(this.props); // => { message: "hi" }
  }
}
```

```
render() {  
  return <button onClick={this.handleClick}>One</button>;  
}  
}
```

You'll see this last approach more often in older code bases.

Conclusion

To add functionality to our components, we can define additional methods within the class. Within those methods, we can access shared data via the `this` keyword.

When defining methods that are meant to be used as *callbacks*, you must use an arrow function or explicitly bind `this` to ensure your callback methods have access to the correct context via the `this` keyword.

You can also define *all* of your component's methods using arrow functions, if remembering all of these rules feels like too much! This is theoretically less performant, but it's unlikely you or your users will notice a difference.

Resources

- [React Handling Events ↗ \(https://reactjs.org/docs/handling-events.html\)](https://reactjs.org/docs/handling-events.html)
- [How do I bind a function to a component instance? ↗ \(https://reactjs.org/docs/faq-functions.html#how-do-i-bind-a-function-to-a-component-instance\)](https://reactjs.org/docs/faq-functions.html#how-do-i-bind-a-function-to-a-component-instance)