

BINDING EVENT HANDLERS

by Nataliya Revutska

softserve

THIS IN EVENT HANDLERS

When you define a component using an [ES6 class](#), a common pattern is for an event handler to be a method on the class.

If you forget to bind `this.clickHandler` and pass it to `onClick`, this will be undefined when the function is actually called.

```
export default class BindingDemonstration extends Component {  
  
  constructor(props) {  
    super(props)  
  
    this.state = {  
      counter: 0  
    }  
  }  
  
  clickHandler() {  
    this.setState(prevState => ({counter: prevState.counter + 1}))  
  }  
  
  render() {  
    return (  
      <div>  
        <div>Counter: {this.state.counter}</div>  
        <button onClick={this.clickHandler}>Click me</button>  
      </div>  
    )  
  }  
}
```

Don't forget to bind event handler!

BINDING IN RENDER

We can bind in *render* method.

The problem with this syntax is that a different callback is created each time the BindingDemonstration renders. In most cases, this is fine.

However, if this callback is passed as a prop to lower components, those components might do an extra re-rendering.

```
export default class BindingDemonstration extends Component {  
  
  constructor(props) {  
    super(props)  
  
    this.state = {  
      counter: 0  
    }  
  }  
  
  clickHandler() {  
    this.setState(prevState => ({counter: prevState.counter + 1}))  
  }  
  
  render() {  
    return (  
      <div>  
        <div>Counter: {this.state.counter}</div>  
        <button onClick={this.clickHandler.bind(this)}>Click me</button>  
      </div>  
    )  
  }  
}
```

ARROW FUNCTION IN RENDER

We can create an arrow function in *render* method.

The problem with this syntax is the same as with binding in render.

```
export default class BindingDemonstration extends Component {  
  
  constructor(props) {  
    super(props)  
  
    this.state = {  
      counter: 0  
    }  
  }  
  
  clickHandler() {  
    this.setState(prevState => ({counter: prevState.counter + 1}))  
  }  
  
  render() {  
    return (  
      <div>  
        <div>Counter: {this.state.counter}</div>  
        <button onClick={() => this.clickHandler()}>Click me</button>  
      </div>  
    )  
  }  
}
```

BINDING IN CONSTRUCTOR

Binding event handler in constructor is considered to be a good practice.

```
export default class BindingDemonstration extends Component {  
  constructor(props) {  
    super(props)  
  
    this.state = {  
      counter: 0  
    }  
  
    this.clickHandler = this.clickHandler.bind(this);  
  }  
  
  clickHandler() {  
    this.setState(prevState => ({counter: prevState.counter + 1}))  
  }  
  
  render() {  
    return (  
      <div>  
        <div>Counter: {this.state.counter}</div>  
        <button onClick={this.clickHandler}>Click me</button>  
      </div>  
    )  
  }  
}
```

PUBLIC CLASS FIELDS SYNTAX

Defining event handler as a public class field is also considered to be a good practice.

```
export default class BindingDemonstration extends Component {  
  
  constructor(props) {  
    super(props)  
  
    this.state = {  
      counter: 0  
    }  
  }  
  
  clickHandler = () => {  
    this.setState(prevState => ({counter: prevState.counter + 1}))  
  }  
  
  render() {  
    return (  
      <div>  
        <div>Counter: {this.state.counter}</div>  
        <button onClick={this.clickHandler}>Click me</button>  
      </div>  
    )  
  }  
}
```



FUTURE

softserve