Event Handlers

How event handlers in React differ from the standard DOM.

WE'LL COVER THE FOLLOWING

- ^
- Comparison With Standard DOM
- Using The Bind Function
- Final thoughts
- Quick quiz on Event Handlers!

In the previous section, we covered React communication in React components. Building upon that, we will now discuss how events are handled in React.

Comparison With Standard DOM

React provides a series of attributes for handling events. The process is very similar to the one used in standard DOM. There are some differences such as using camel case or the fact that we pass a function, but overall it is pretty similar.

Using The Bind Function

Usually, we handle events in the component that contains the elements dispatching the events. Like in the example below, we have a click handler and we want to run a function or a method of the same component:

That's all fine because _handleButtonClick is a function and we do pass a function to the onClick attribute. The problem, however, is that, as it is, the code doesn't keep context. So, if we have to use this inside _handleButtonClick to refer to the current Switcher component, we will get an error.

```
class Switcher extends React.Component {
                                                                                         constructor(props) {
    super(props);
   this.state = { name: 'React in patterns' };
  render() {
    return (
      <button onClick={ this._handleButtonClick }>
        click me
      </button>
   );
 }
  _handleButtonClick() {
    console.log(`Button is clicked inside ${ this.state.name }`);
    // Uncaught TypeError: Cannot read property 'state' of null
  }
};
```

What we normally do is to use bind:

```
<button onClick={ this._handleButtonClick.bind(this) }>
  click me
</button>
```

However, this means that the bind function is called again and again because we may render the button many times. A better approach would be to create the bindings in the constructor of the component:

Facebook, by the way, recommends the same technique while dealing with functions that need the context of the same component.

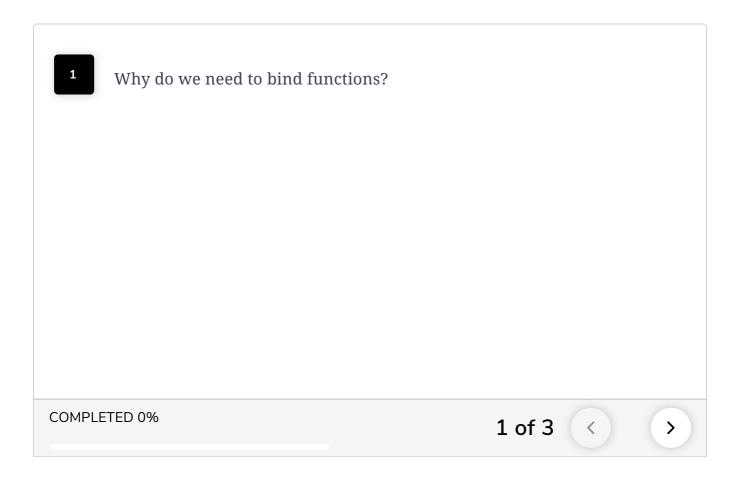
The constructor is also a nice place for partially executing our handlers. For example, we have a form but want to handle every input in a single function.

```
class Form extends React.Component {
                                                                                         constructor(props) {
   super(props);
   this._onNameChanged = this._onFieldChange.bind(this, 'name');
   this. onPasswordChanged =
     this._onFieldChange.bind(this, 'password');
  }
  render() {
    return (
        <input onChange={ this._onNameChanged } />
        <input onChange={ this._onPasswordChanged } />
      </form>
    );
  _onFieldChange(field, event) {
    console.log(`${ field } changed to ${ event.target.value }`);
  }
};
```

Final thoughts

Since we are mostly using HTML-like syntax, it makes sense that we also have DOM-like event handling. Consequently, there is not much additional information to learn about event handling in React since the authors of the library have done an excellent job of keeping what's already there. The following code shows how the event handling code would fit in the complete Form component:

Quick quiz on Event Handlers!



Now that we understand event handling with React components, we will take a closer look at what the components themselves are composed of.