Ref a DOM Element

In this lesson, we'll learn how to interact with DOM nodes in React.

Why interact with the DOM?

Sometimes you need to interact with your DOM nodes in React. The `ref` attribute gives you access to a node in your elements. That is usually an anti pattern in React, because you should use its declarative way of doing things and its unidirectional data flow. We learned about it when we introduced the first search input field, but there are certain cases where you need access to the DOM node.

The official documentation mentions three:

- to use the DOM API to manage focus, media playback etc
- to invoke imperative DOM node animations
- to integrate with a third-party library that needs the DOM node (e.g. D3.js)

Referring to the DOM in the Search component

We'll use the Search component as an example. When the application renders for the first time, the input field should be focused. This is one case where we need access to the DOM API. The `ref` attribute can be used in both functional stateless components and ES6 class components. In this example, we need a lifecycle method, so the approach is showcased using the `ref` attribute with an ES6 class component.

The initial step is to refactor the functional stateless component to an ES6 class component.

```
class Search extends Component {
  render() {
    const {
      value,
```

```
      onChange,
      onSubmit,
      children
    } = this.props;

    return (
      <form onSubmit={onSubmit}>
        <input
          type="text"
          value={value}
          onChange={onChange}
        />
        <button type="submit">
          {children}
        </button>
      </form>
    );
  }
}
```

The `this` object of an ES6 class component helps us to reference the DOM element with the `ref` attribute.

```
class Search extends Component {
  render() {
    const {
      value,
      onChange,
      onSubmit,
      children
    } = this.props;

    return (
      <form onSubmit={onSubmit}>
        <input
          type="text"
          value={value}
          onChange={onChange}
          ref={el => this.input = el}
        />
        <button type="submit">
          {children}
        </button>
      </form>
    );
  }
}
```

Now you can focus the input field when the component mounted by using the `this` object, the appropriate lifecycle method, and the DOM API.

```
class Search extends Component {
  componentDidMount() {
    if (this.input) {
      this.input.focus();
```

```
    }
  }

  render() {
    const {
      value,
      onChange,
      onSubmit,
      children
    } = this.props;

    return (
      <form onSubmit={onSubmit}>
        <input
          type="text"
          value={value}
          onChange={onChange}
          ref={el => this.input = el}
        />
        <button type="submit">
          {children}
        </button>
      </form>
    );
  }
}
```

The input field should be focused when the application renders. We access to the `ref` in a functional stateless component without the `this` object using the following functional stateless component:

```
const Search = ({
  value,
  onChange,
  onSubmit,
  children
}) => {
  let input;
  return (
    <form onSubmit={onSubmit}>
      <input
        type="text"
        value={value}
        onChange={onChange}
        ref={el => this.input = el}
      />
      <button type="submit">
        {children}
      </button>
    </form>
  );
}
```

Now we can access the input DOM element. In our case it wouldn't help much since there's no lifecycle method in a functional stateless component to trigger

the focus. So we are not going to make use of the input variable here. In the

future, though, you may encounter cases where it makes sense to use a functional stateless component with the `ref` attribute.

Further Reading

- Read about the usage of the ref attribute in React
- Read about the ref attribute in general in React