Stateful DOM Components

Relevel
by Unacademy

Topics Covered

Keys:
What are Keys and how they are used?

Uncontrolled components:
What are uncontrolled components

Ref:
What are Refs and what are they used for?

Topics Covered

Controlled Components: What are controlled components and how are they different from uncontrolled components.

Practice Questions: Assignments

Keys

- Keys play a crucial role in identifying the changes performed on an array whether new elements are added or removed.
- They provide a stable identity for each element of an array.

Consider the following example:
We have an array called numbers containing the numbers from one to five.
We will then run a map on this array that will double the array element and return a list item displaying the doubled number, the return value will be stored in listItems.
We will pass a key to the key attribute in the list item, that is a string value of the number itself.

https://stackblitz.com/edit/react-tmbt3n?file=src/App.js

Keys

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>    {number*2}
  </li>
);

// Output
●    2
●    4
●    6
●    8
●    10
```

Keys

Normally, we provide our own ID field in our dataset that acts as a key as shown in the example

https://stackblitz.com/edit/react-ym7iwk?file=src/App.js

```
const listItems = items.map((item) =>
  <li key={item.id}>    {item.text}
  </li>
);
```

Keys

When we don’t have stable IDs for rendered items, we may use the item index as a key as a last resort:

https://stackblitz.com/edit/react-xqsk22?file=src/App.js

```
const listItems = items.map((item, index) =>
  // Only do this if items have no stable IDs
<li key={index}>    {item.text}
  </li>
);
```

Uncontrolled Components

- In HTML, form elements such as <input>, <textarea>, & <select> usually maintain their own state and update it based on the user input.
- When left unchecked, the default behaviours persist and might lead to inconsistencies.
- So an uncontrolled component is basically a collection form elements that is handled by DOM instead of React.js
- Now, since we will allow the default behaviors of the form elements to remain the same, instead of writing an event handler for updates like onChange, we will use an attribute called ref.

Refs

- Refs are React's version of references; they provide us with a way to access DOM nodes or React elements that are created in the render method.
- While driving on highways or streets, have you seen signs pointing towards KFC, McDonald's or Domino's?
- This is why a class was the only stateful component available.
- Classes allowed us to manage and modify the state effectively.
- Refs are like those signs, they do not exclusively contain or store something, but they point or refer to an element.

Refs

- Usually, in any React codebase, props are the only way to communicate between the components; props are the only way a parent component can send values to child components and interact with them.
- However, in certain cases, we would need to modify the child component outside of the normal data flow.
- Irrespective of whether the child is a React component or a DOM element, we can make the modifications via Refs.

Refs

Here are some occasions where Refs could be used:

- Managing focus, text selection, or media playback.
- Triggering animations that are imperative in nature.
- Integrating the codebase with third-party DOM libraries.

Note that usage of Ref makes the flow imperative in nature, which is the polar opposite of React's declarative philosophy; thus, avoid using Refs as much as possible.

- When you know what you want from your program then we follow the **Declarative programming** approach and when you know how to get to what we want we follow the **Imperative programming** approach.
- You have to provide step by step instruction to tell the compiler what you want to happen with the program in **Imperative programming**. In **Declarative programming**, you have to write your own code describing what you want your program to do.

Refs

How are Refs created?

- **React.createRef()** method is used in react to create a Ref.
- Now in order to assign these Refs to an element we make use of the **ref** attribute.

Using Ref in class

A typical class based component contains the following:.

- We have a class based component called Uncontrolled that extends React’s Component.
- In the constructor method we have declared a variable called input that is assigned to the Ref created using React.createRef().
- This input variable will be used for referring to the Ref.
- We then have a form component in the render method.
- It has an input field whose type is text and its instance is assigned to the input ref using the ref attribute.
- It then has an input type called submit.

Using Ref in class

```
class Uncontrolled extends Component {
  constructor(props) {
    super(props);
    this.input = React.createRef();
  }
  handleSubmit = (event) => {
    alert("Uncontrolled component: " +
this.input.current.value);
    event.preventDefault();
  };
```

Using Ref in class

```
render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Controlled Components

- Controlled components are the exact opposite of uncontrolled components.
- Instead of allowing the default HTML behaviour to persist, React takes complete authority over the form elements and their behaviour.
- In React, the modifiable state is typically kept in components' state property and only updated with the setState() method.

Controlled Components

- If we allow the default behaviour of form elements to exist, they may jeopardize our execution flow.
- Thus, we would make React be the sole owner of how the form behaves and what happens to the form on subsequent user input.
- So a form whose behaviour is controlled wholly by React is called a controlled component.

Example

```
class ControlledForm extends Component {
  state= {
    value: ''
  }
  handleChange =(event) =>{
    this.setState({ value: event.target.value });
  }
  handleSubmit =(event) => {
    alert("Controlled Name: " + this.state.value);
    event.preventDefault();
  }
```

Example

```
    render() {
        return (
          <form onSubmit={this.handleSubmit}>
            <label>
              Name:
              <input
                type="text"
                value={this.state.value}
                onChange={this.handleChange}
              />        </label>
            <input type="submit" value="Submit" />
          </form>    );  }  }
```

Example

In the previous example, we had the following:

- We have a class component called ControlledForm, that has a local state containing the property value.
- In the render method, we have a form component whose value attribute is assigned to the local value.
- Its onChange event is assigned to a callback handleChange.
- All HTML forms navigate to a page upon form submission, here we have our callback handleSubmit assigned to the onSubmit event.
- The event.preventDefault() method tells the browser to remove the form's default behaviour where it navigates to a page when a submit event is triggered.
- This is an example of a controlled component.

Assignment

1. Create a controlled form that accepts employee details and displays them based on validations.

Thank You!