

React JS

Created by :
Sangeeta Joshi

Agenda

- Event Handling
- Forms
- Controlled Components
- Lifting State Up

Handling Events in React

- It is very similar to handling events on DOM elements.

-There are some syntactic differences:

React events: named using camelCase, not lowercase.

With JSX, pass function as event handler, not string

Html:

```
<button onclick="activateLasers()">  
  Activate Lasers  
</button>
```

Return false to prevent default behaviour

React:

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```

Call preventDefault() to prevent default behaviour

Handling Events in React

- **SyntheticEvent** : a cross-browser wrapper around the browser's native event.
- has the same interface as the browser's native event, including `stopPropagation()` and `preventDefault()`, except the events work identically across all browsers
- If you need the underlying browser event for some reason, simply use the **nativeEvent** attribute to get it.

Data flows down

- Any state is always owned by some specific component. It is called local or encapsulated
- It is not accessible to any component other than the one that owns and sets it.
- and any data or UI derived from that state can only affect components "**below**" them in the tree.
- This is commonly called a "top-down" or "unidirectional" data flow.

Lifting State Up

- Several components may need to reflect the same changing data.
- In such cases, we lift the shared state up to their closest common ancestor
- In React, sharing state is accomplished by moving it up to the closest common ancestor of the components that need it. This is called "lifting state up"

Lifting State Up

- Lifting state involves writing more "boilerplate" code than two-way binding approaches
- but as a benefit, it takes less work to find and isolate bugs.
- Since any state "lives" in some component and that component alone can change it, the surface area for bugs is greatly reduced.

Forms: Controlled Comp

HTML : form elements such as <input>, <select> typically maintain their own state and update it based on user input.

React : mutable state is typically kept in the state property of components, and only updated with `setState()`.

combine two by making React state: "single source of truth".

Then React component that renders a form also controls:

- what happens in that form on subsequent user input.

An input form element whose value is controlled by React in this way is called a "controlled component".

Forms: Controlled Comp

HTML : form elements such as <input>, <select> typically maintain their own state and update it based on user input.

React : mutable state is typically kept in the state property of components, and only updated with `setState()`.

combine two by making React state: "single source of truth".

Then React component that renders a form also controls:

- what happens in that form on subsequent user input.

An input form element whose value is controlled by React in this way is called a "controlled component".

Forms: UnControlled Comp

- sometimes its tedious to use controlled components, because :you need to write an event handler for every way your data can change and pipe all of the input state through a React component.
- Especially when you are converting a pre existing codebase to React, or integrating a React application with a non-React library.
- In these situations, you may use uncontrolled components, an alternative technique for implementing input forms.

Forms: UnControlled Comp

- Using controlled components to implement forms is generally recommended.
- In a controlled component, form data is handled by a React component.
- The alternative is uncontrolled components, where form data is handled by the DOM itself.
- To write an uncontrolled component, instead of writing an event handler for every state update, ***you can use a ref to get form values from the DOM.***

Types Checking

- As app grows, you can catch a lot of bugs with typechecking.
- For some applications, we can use JavaScript extensions like Flow or TypeScript to typecheck your whole application.
- But even if you don't use those, React has some built-in typechecking abilities.
- To run typechecking on the props for a component, you can assign the special `propTypes` property