**Event Delegation**

You have probably already encountered delegation of authority and tasks. The meaning of this word is very important, isn’t it? And the main point is in centralizing decision-making, thereby reducing the number of decision-making points. This is the work of a leader.

Perhaps the key points of delegation:

* Centralization (which makes it possible to monitor events)
* Tracking (from whom the event came)
* Filtering (decide to react or not)

In fact, you do not handle an event on each child DOM element, but handle them on some parent DOM element and then decide what to do and how.

Event delegation - is a method of attaching event handlers not to the elements from which you actually want to handle events, but to a higher-level element.

***When to apply?***

1. In any case, when you work with a large number of interactive elements. We get a lot of elements that can be removed and added quite intensively. If you accidentally did not clear the events of an element that was previously removed from the DOM by any line of code, you will get zombies that can eat up the memory.
2. The solution to the problem of handling events in parent elements with a large nesting of child elements. For example, if you click on the most nested child element, click events will be triggered on each of the parent elements. And here you can avoid using e.stopPropagation(). This is why it is not recommended to use event chain interruption in handlers, since if you need to handle an event on the parent component, then you will not succeed.

***Example:***

*ReactEventDeligation.js*

import React, { Component } from "react";

class ReactEventDeligation extends Component {

  constructor(props) {

    super(props);

    this.root = React.createRef();

    this.button = React.createRef();

  }

  componentDidMount() {

    this.root.current.addEventListener(

      "click",

      console.log("div capture"),

      true

    );

    this.root.current.addEventListener(

      "click",

      console.log("div bubble"),

      false

    );

    this.button.current.addEventListener(

      "click",

      console.log("button capture"),

      true

    );

    this.button.current.addEventListener(

      "click",

      console.log("button bubble"),

      false

    );

  }

  onClick = () => {

    console.log("React onClick for button");

  };

  render() {

    return (

      <div ref={this.root}>

        <h3>React Event Deligation</h3>

        <button onClick={this.onClick} ref={this.button}>

          Click me!

        </button>

      </div>

    );

  }

}

export default ReactEventDeligation;

*App.js*

import "./App.css";

import ReactEventDeligation from "./components/ReactEventDeligation";

function App() {

  return (

    <div className="App">

      <ReactEventDeligation />

    </div>

  );

}

export default App;

**Stateless Components**

Stateless components are those components which don’t have any state at all, which means you can’t use *this.setState* inside these components. It is like a normal function with no render method. It has no lifecycle, so it is not possible to use lifecycle methods such as *componentDidMount* and other hooks. When react renders our stateless component, all that it needs to do is just call the stateless component and pass down the props.

A stateless component can render props, whereas a stateful component can render both props and state. A significant thing to note here is to comprehend the syntax distinction. In stateless components, the props are displayed like *{props.name}*but in stateful components, the props and state are rendered like *{this.props.name}* and *{this.state.name}* respectively. A stateless component renders output which depends upon props value, but a stateful component render depends upon the value of the state. A functional component is always a stateless component, but the class component can be stateless or stateful.

***When should I make a component stateful or stateless?***

It’s pretty straightforward that you should make your component stateful whenever you want to have a dynamic output (means that the output will change whenever the state changes), and you want to share the properties of parent component with the children components. On the other side, if there is no state necessity, you should make the component stateless.

Stateless components are more elegant and usually are the right choice for building the presentational components because they are just functions, you won’t find it challenging to write and understand them, and moreover, they are very straightforward to test.

***Example:***

*ReactStateless.js*

import React from "react";

function ReactStateless(props) {

  return (

    <div>

      <h3>React Stateless Component</h3>

      <h4>

        Full Name:{" "}

        <span style={{ color: "red" }}>

          {props.first\_name} {props.last\_name}

        </span>

      </h4>

    </div>

  );

}

export default ReactStateless;

*App.js*

import "./App.css";

import ReactStateless from "./components/ReactStateless";

function App() {

  return (

    <div className="App">

      <ReactStateless first\_name="Sharlin" last\_name="Lins" />

    </div>

  );

}

export default App;

**Synthetic Event**

In order to work as a cross-browser application, react has created a wrapper same as the native browser. In order to avoid creating multiple implementations for multiple methods for multiple browsers, creating common names for all events across browsers. Another benefit is that it increases the performance of the application as React reuse the event object.

***Syntax:***

* e.preventDefault() prevents all the default behaviour by the browser.
* e.stopPropagation() prevents the call to the parent component whenever a child component gets called.

***Example:***

*ReactSyntheticEvent.js*

import React from "react";

function ReactSyntheticEvent() {

  const clickHandler = (e) => {

    console.log(e);

  };

  return (

    <div>

      <h3>React Synthetic Event</h3>

      <button onClick={(e) => clickHandler(e)}>Click</button>

    </div>

  );

}

export default ReactSyntheticEvent;

*App.js*

import "./App.css";

import ReactSyntheticEvent from "./components/ReactSyntheticEvent";

function App() {

  return (

    <div className="App">

      <ReactSyntheticEvent />

    </div>

  );

}

export default App;

**React.js Event Handlers**

In this React tutorial, we will get to know event handlers in React for HTML elements such as button and input elements. You will learn how to use a button with its *onClick* event and how to define and use different kinds of event handlers. Essentially we will go through three kinds of event handlers: event handlers, inline event handlers and callback event handlers.

First, we will start with a button example in React for a specific **onClick event handler**. It's the most basic example on how to handle events in React with an **event handler** (also called **event handler function** or **handler**). A button has a *onClick* attribute which receives a function. This function is called every time the event is triggered (here: when clicking the button):

***Example:***

import React from 'react';

function App() {

  const handleClick = () => {

    console.log('Button click ...');

  };

  return (

    <div>

      <button type="button" onClick={handleClick}>

        Event Handler

      </button>

    </div>

  );

}

export default App;

***Inline Event Handler***

Inline event handlers, also called **inline handlers**, give us lots of new options by using an event handler directly in JSX:

***Example:***

import React from 'react';

  function App() {

    const [count, setCount] = React.useState(0);

    return (

      <div>

        Count: {count}

        <button

          type="button"

          onClick={() => setCount(count + 1)}

        >

          Increase Count

        </button>

      </div>

    );

  }

export default App;

Inline handlers are also used to pass a parameter to a more universal handler which is defined outside of the JSX:

***Example:***

import React from "react";

function App() {

  const [count, setCount] = React.useState(0);

  function handleCount(delta) {

    setCount(count + delta);

  }

  return (

    <div>

      Count: {count}

      <button type="button" onClick={() => handleCount(1)}>

        Increase Count

      </button>

      <button type="button" onClick={() => handleCount(-1)}>

        Decrease Count

      </button>

    </div>

  );

}

export default App;

***Callback Event Handler***

There are **callback event handlers** or **callback handlers** in short. They are used when a child component needs to communicate to a parent component. Since [React props](https://www.robinwieruch.de/react-pass-props-to-component/) are only passed down the component tree, a callback handler, which is a function at its core, is used to communicate upward.

***Example:***

import React from "react";

function App() {

  const [text, setText] = React.useState("");

  // 1

  function handleTextChange(event) {

    setText(event.target.value); // 3

  }

  return (

    <div>

      <MyInput inputValue={text} onInputChange={handleTextChange} />

      {text}

    </div>

  );

}

// 2

function MyInput({ inputValue, onInputChange }) {

  return <input type="text" value={inputValue} onChange={onInputChange} />;

}

export default App;

A callback handler is defined somewhere *(1)*, used somewhere else *(2)*, but calls back to the place where its defined *(3)*. This way, it's possible to communicate from child to parent components. A callback handler is passed down via React props and communicates up when the function is called.

**Creating A Dynamic UI in React**

Dynamic UI design is the practice of designing a user interface to adapt according to user input. When designing a dynamic user interface (UI), you control the fields that users see and interact with, according to specified conditions. Dynamic UI content is a key element of an intent-driven UI design. Choices made by users drive what controls are displayed so that only relevant fields are presented.

***Example:***

*ReactDynamicUI.js*

import React, { useState } from "react";

function ReactDynamicUI() {

  const [inputFields, setInputFields] = useState([{ username: "", age: "" }]);

  const handleChange = (index, event) => {

    let data = [...inputFields];

    data[index][event.target.name] = event.target.value;

    setInputFields(data);

  };

  const addFields = () => {

    let newfield = { username: "", age: "" };

    setInputFields([...inputFields, newfield]);

  };

  const submitHandler = (e) => {

    e.preventDefault();

    console.log(inputFields);

  };

  const removeFields = (index) => {

    let data = [...inputFields];

    data.splice(index, 1);

    setInputFields(data);

  };

  return (

    <div>

      <h3>React Dynamic UI</h3>

      <form>

        {inputFields.map((input, index) => {

          return (

            <div key={index}>

              <input

                className="mt-1 mx-3"

                type="text"

                name="username"

                placeholder="Name"

                value={input.username}

                onChange={(event) => handleChange(index, event)}

              />

              <input

                className="mt-1 mx-3"

                type="text"

                name="age"

                placeholder="Age"

                value={input.age}

                onChange={(event) => handleChange(index, event)}

              />

              <button

                className="btn btn-danger btn-sm px-2"

                onClick={() => removeFields(index)}

                type="button"

              >

                Remove Field

              </button>

            </div>

          );

        })}

      </form>

      <button

        className="btn btn-success btn-sm mt-2 px-4"

        onClick={addFields}

        type="button"

      >

        Add

      </button>

      <button

        className="btn btn-primary btn-sm mt-2 mx-2 px-4"

        onClick={submitHandler}

        type="submit"

      >

        Submit

      </button>

    </div>

  );

}

export default ReactDynamicUI;

*App.js*

import "./App.css";

import ReactDynamicUI from "./components/ReactDynamicUI";

function App() {

  return (

    <div className="App">

      <ReactDynamicUI />

    </div>

  );

}

export default App;