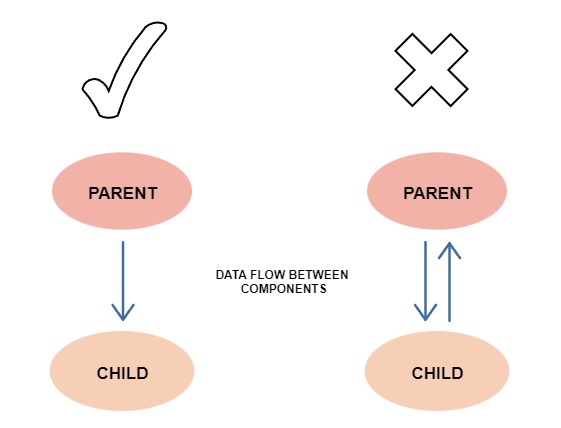
**React Data Flow**

React, a JavaScript library, uses unidirectional data flow. The data from the parent is known as **props**. You can only transfer data from parent to child and not vice versa.

This means that the child components cannot update or modify the data on their own, making sure that a clean data flow architecture is followed. This also means that you can control the data flow better.

****

***Effect of state changes***

In React, each state is owned by one component. Due to the unidirectional flow, the changes made on the state of the component will only affect the children and not the parents or siblings.

***Advantages of unidirectional data flow***

There are many advantages of unidirectional data flow, some of which are listed below:

***Debugging***

One-way data flow makes debugging much easier. When the developer knows where the data is coming from and where it is going, they can dry run (or use tools) to find the problems more efficiently.

***Better control***

Having data flow in one direction makes the program less prone to errors and gives the developer more control.

***Efficiency***

As the used libraries are wary of the limitations and specifications of the unidirectional flow, extra resources are not wasted, leading to an efficient process.

**Action, Dispatcher, Store & View**

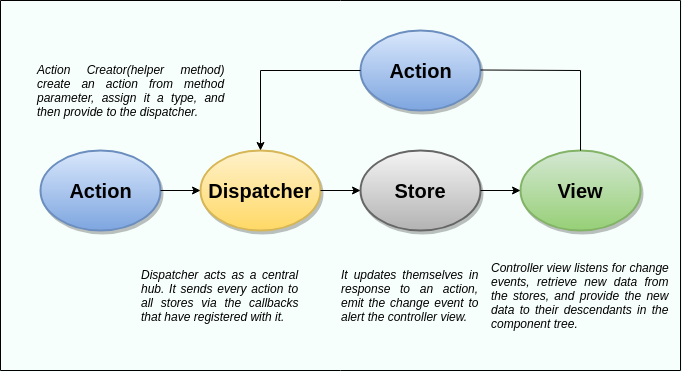
**Flux** is a programming concept, where the data is ***uni-directional***. This data enters the app and flows through it in one direction until it is rendered on the screen.

***Flux Elements***

Following is a simple explanation of the **flux** concept.

* **Actions** − Actions are sent to the dispatcher to trigger the data flow.
* **Dispatcher** − This is a central hub of the app. All the data is dispatched and sent to the stores.
* **Store** − Store is the place where the application state and logic are held. Every store is maintaining a particular state and it will update when needed.
* **View** − The **view** will receive data from the store and re-render the app.

***Structure and Data Flow***

******

***Actions***

The dispatcher method allows us to trigger a dispatch to the store and include a payload of data, which we call an action. It is an action creator or helper methods that pass the data to the dispatcher.

***Dispatcher***

It is a central hub for the React Flux application and manages all data flow of your Flux application. It is a registry of callbacks into the stores. It has no real intelligence of its own, and simply acts as a mechanism for distributing the actions to the stores. All stores register itself and provide a callback. It is a place which handled all events that modify the store. When an action creator provides a new action to the dispatcher, all stores receive that action via the callbacks in the registry.

The dispatcher's API has five methods. These are:

|  |  |
| --- | --- |
| ***Methods*** | ***Descriptions*** |
| *register()* | *It is used to register a store's action handler callback.* |
| *unregister()* | *It is used to unregisters a store's callback.* |
| *waitFor()* | *It is used to wait for the specified callback to run first.* |
| *dispatch()* | *It is used to dispatches an action.* |
| *isDispatching()* | *It is used to checks if the dispatcher is currently dispatching an action.* |

***Stores***

It primarily contains the application state and logic. It is similar to the model in a traditional MVC. It is used for maintaining a particular state within the application, updates themselves in response to an action, and emit the change event to alert the controller view.

***Views***

It is also called as controller-views. It is located at the top of the chain to store the logic to generate actions and receive new data from the store. It is a React component listen to change events and receives the data from the stores and re-render the application.

**Event Handling in React**

There are many events that get triggered by the system such as clicking, page loading, keypress, scrolling, etc. React provides event handling which is similar to event handling on DOM elements. React events are synthetic events, a cross-browser wrapper around the browser’s native event. It works the same across all browsers.

React event handling is similar to HTML with some changes in syntax, such as:

* React uses camelCase for event names while HTML uses lowercase.
* Instead of passing a string as an event handler, we pass a function in React.

|  |  |
| --- | --- |
| *HTML* | *ReactJS* |
| <button onclick="clickHandler()">  Clicked  </button> | <button onClick={clickHandler}>  Clicked  </button> |
| <form onsubmit="console.log('clicked'); return false">  <button type="submit">Submit</button>  </form> | function Form() {  function handleClick(e) {  e.preventDefault();  console.log('Clicked');  }  return (  <form onSubmit={handleClick}>  <button type="submit">Submit</button>  </form>  );  } |

Changing state in onClick event listener:- We have to call this.setState method (for class component) or useState hook (for functional component) inside the onClick handler to update state value.

*Example: Class Component*

*ReactEventCC.js*

import React, { Component } from "react";

export class ReactEventCC extends Component {

  constructor(props) {

    super(props);

    this.state = {

      count: 0,

    };

    this.clickHandler = this.clickHandler.bind(this);

  }

  clickHandler = () => {

    this.setState({ count: this.state.count + 1 });

  };

  render() {

    return (

      <div>

        <h3>React Event Class Component</h3>

        <h4>Count Value is {this.state.count}</h4>

        <button onClick={this.clickHandler}>Increase Count</button>

      </div>

    );

  }

}

export default ReactEventCC;

*App.js*

import "./App.css";

import ReactEventCC from "./components/ReactEventCC";

function App() {

  return (

    <div className="App">

      <ReactEventCC />

    </div>

  );

}

export default App;

*Example: Functional Component*

*ReactEventFC.js*

import React, { useState } from "react";

function ReactEventFC() {

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

  const clickHandler = () => {

    setCount(count + 1);

  };

  return (

    <div>

      <h3>React Event Functional Component</h3>

      <h4>Count Value is: {count}</h4>

      <button onClick={clickHandler}>Increase Count</button>

    </div>

  );

}

export default ReactEventFC;

*App.js*

import "./App.css";

import ReactEventFC from "./components/ReactEventFC";

function App() {

  return (

    <div className="App">

      <ReactEventFC />

    </div>

  );

}

export default App;

***Binding Event Handler***

If you have used JavaScript before then you are familiar with the ‘this’ keyword and how it works. If you don’t bind the ‘this’ keyword with the event handler, then it will point to undefined in the callback. This is JavaScript-specific behaviour and is not related to React. There are five different ways to bind ‘this’ keyword to the event handler, listed below.

1. Binding Inside the Constructor
2. Binding by Passing Arrow Function Inside the Event Listener
3. Binding Directly When Passing the Function
4. Binding with The Help of Arrow Function
5. Use Functional Component with Arrow Function
6. Call Multiple Functions Inside an onClick Event Handler
7. Passing Argument to Event Handlers

***1. Binding Inside the Constructor***

We can bind the ‘this’ keyword to the event handler in the constructor when using the class component.

*Example:*

import React, { Component } from "react";

class EventBind extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      message: "Event Bind",

    };

    this.clickHandler = this.clickHandler.bind(this);

  }

  clickHandler() {

    this.setState({

      message: "change state",

    });

  }

  render() {

    return (

      <>

        <div>{this.state.message}</div>{" "}

        <button onClick={this.clickHandler}> Click </button>

      </>

    );

  }

}

export default EventBind;

***2. Binding by Passing Arrow Function Inside the Event Listener***

We can use the arrow function as the function in the event listener. In this case we don’t need to bind the ‘this’ keyword. But this method has its own drawbacks.

*Example*

import React, { Component } from "react";

class EventBind extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      message: "Event Bind",

    };

  }

  clickHandler() {

    this.setState({

      message: "change state",

    });

  }

  render() {

    return (

      <>

        <div>{this.state.message}</div>{" "}

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

      </>

    );

  }

}

export default EventBind;

***3. Binding Directly When Passing the Function***

Bind the ‘this’ keyword directly when passing a function in the event listener.

*Example*

import React, { Component } from "react";

class EventBind extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      message: "Event Bind",

    };

  }

  clickHandler() {

    this.setState({

      message: "change state",

    });

  }

  render() {

    return (

      <>

        <div>{this.state.message}</div>{" "}

        <button onClick={this.clickHandler.bind(this)}> Click </button>

      </>

    );

  }

}

export default EventBind;

***4. Binding with The Help of Arrow Function***

Make the event handler function with an arrow function. In this case, we don’t need to bind the ‘this’ keyword.

*Example*

import React, { Component } from "react";

class EventBind extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      message: "Event Bind",

    };

  }

  clickHandler = () => {

    this.setState({

      message: "change state",

    });

  };

  render() {

    return (

      <>

        <div>{this.state.message}</div>{" "}

        <button onClick={this.clickHandler}> Click </button>

      </>

    );

  }

}

export default EventBind;

*Binding using an arrow function*

import React, { Component } from "react";

class EventBind extends React.Component {

    clickHandler = event => {

      console.log("Clicked")

    }

    render() {

      return (

        <button onClick={this.clickHandler}>Click</button>

      )

    }

  }

export default EventBind;

***5. Use Functional Component with Arrow Function***

We use the functional component along with the arrow function then we don’t need to bind ‘this’ keyword.

*Example*

import React, { Component } from "react";

const EventBind = () => {

  const [steps, setSteps] = useState(0);

  function clickHandler() {

    setSteps(steps + 1);

  }

  return (

    <>

      <div>{steps}</div> <button onClick={clickHandler}> Click </button>

    </>

  );

};

export default EventBind;

***6. Call Multiple Functions Inside an onClick Event Handler***

We can call multiple functions inside the onClick event handler in React. To do so we just need to call them one by one.

*Example*

import React, { Component } from "react";

class EventBind extends React.Component {

  printMessage() {

    console.log("This is a message");

  }

  showAlert() {

    alert("button was clicked");

  }

  clickHandler() {

    this.printMessage();

    this.showAlert();

  }

  render() {

    return <button onClick={this.clickHandler.bind(this)}>Click</button>;

  }

}

export default EventBind;

***7. Passing Argument to Event Handlers***

Sometimes we need to pass an extra parameter to the event handler. We can do that using two of the methods above.

***Example:***  
Pass the event and an extra parameter, like id, in the event listener using the arrow function.

<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>

Or we can use them by directly binding them when passing the event handler in the event listener and passing an extra parameter there.

<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>

**Keyboard Events in React**

With your generic knowledge of events in React so far, you can now implement keyboard events in React. As mentioned earlier, there are two keyboard events that can be used, the *onKeyDown()* and *onKeyUp()* events.

Example:

import React from "react";

function ReactKeyboardEvent() {

  return (

    <div>

      <h3>React Keyboard Event</h3>

      <label>First Name: </label>

      <input

        name="FirstName"

        onKeyDown={(e) => console.log("onKeyDown:", e.key, e.code)}

        onKeyUp={(e) => console.log("onKeyUp:", e.key, e.code)}

      />

    </div>

  );

}

export default ReactKeyboardEvent;