React Wrapup

Springboard

Download Demo Code

React

- Props
- State
- Effects
- Router
- Redux

Additional Hooks

useCallback Motivation

Here's a common scenario in React:

- You have a function you want to call inside of an effect that depends on props or state.
- Since it depends on props / state, it should be listed as a dependency.
- But if the function is defined inside of the component, this can cause infinite render loops!

Effects and Functions: Missing Dependencies

demo/more-hooks/src/NumberFactMissingDep.js

```
function NumberFactMissingDep({
  baseUrl = "http://numbersapi.com/", initialNum = 42
}) {
  const [num, setNum] = useState(initialNum);
  const [fact, setFact] = useState("");
  async function getFact(newNum) {
    let response = await axios.get(`${baseUrl}/${newNum}?json`);
    setNum(newNum);
    setFact(response.data.text);
  }

  useEffect(() => { getFact(initialNum) }, []);

  return (
    <div>
      <NumberInput getFact={getFact} initialNum={initialNum} />
      {fact
        ? <div><h3>{num}</h3><p>{fact}</p></div>
        : <p>Loading...</p>}
    </div>
  );
}
```

```
⚠ ./src/NumberFactMissingDep.js                                    backend.js:6
  Line 10:64:  React Hook useEffect has missing dependencies: 'getFact' and 'initialNum'. Either
  include them or remove the dependency array  react-hooks/exhaustive-deps
```

Effects and Functions: Function Dependencies

demo/more-hooks/src/NumberFactNoCallback.js

```
function NumberFactNoCallback({
  baseUrl = "http://numbersapi.com/", initialNum = 42
}) {
  const [num, setNum] = useState(initialNum);
  const [fact, setFact] = useState("");
  async function getFact(newNum) {
    let response = await axios.get(`${baseUrl}/${newNum}?json`);
    setNum(newNum);
    setFact(response.data.text);
  }

  useEffect(() => { getFact(initialNum) }, [initialNum, getFact]);

  return (
    <div>
      <NumberInput getFact={getFact} initialNum={initialNum} />
      {fact
        ? <div><h3>{num}</h3><p>{fact}</p></div>
        : <p>Loading...</p>}
    </div>
  );
}
```

```
⚠ ./src/NumberFactNoCallback.js                          webpackHotDevClient.js:139
  Line 20:9:  The 'getFact' function makes the dependencies of useEffect Hook (at line 16) change on
  every render. To fix this, wrap the 'getFact' definition into its own useCallback() Hook  react-
  hooks/exhaustive-deps
```

useCallback

- **useCallback** is a built-in hook that accepts a function and an array of dependencies
- It returns a function that won't be re-declared on subsequent renders, as long as the dependencies don't change
- This allows you to add functions as dependencies to ***useEffect*** without hitting infinite render issues

demo/more-hooks/src/NumberFactUseCallback.js

```
function NumberFactUseCallback({
  baseUrl = "http://numbersapi.com/", initialNum = 42
}) {
  const [num, setNum] = useState(initialNum);
  const [fact, setFact] = useState("");
  const getFact = useCallback(async newNum => {
    let response = await axios.get(`${baseUrl}/${newNum}?json`);
    setNum(newNum);
    setFact(response.data.text);
  }, [baseUrl]);

  useEffect(() => { getFact(initialNum); }, [initialNum, getFact]);

  return (
    <div>
      <NumberInput getFact={getFact} initialNum={initialNum} />
      {fact
        ? <div><h3>{num}</h3><p>{fact}</p></div>
        : <p>Loading...</p>}
      <NumberDivisors num={num} />
    </div>
  );
}
```

useMemo

- **useMemo** is another built-in hook in React
- Like **useCallback**, but for remembering values other than functions
- Accepts a function returning a value and an array of dependencies
- React won't recompute the values if the dependencies stay the same
- Helpful for caching the results of expensive operations

useMemo Example

demo/more-hooks/src/NumberDivisors.js

```
import React, { useMemo } from "react";
import { getDivisors } from "./helpers";

function NumberDivisors({ num }) {
  // don't recompute the divisors
  // if the number is unchanged
  let divisors = useMemo(() => getDivisors(num), [num])

  return (
    <div>
      Here are all the divisors of {num}!
      <ul>
        {divisors.map(divisor => (
          <li key={divisor}>{divisor}</li>
        ))}
      </ul>
    </div>
  );
}
```

Redux Without Redux

useReducer

- Another built-in hook, **useReducer**, lets you use reducers without redux!
- ***useReducer*** accepts a reducer function and initial state.
- ***useReducer*** returns an array with the state and a dispatch function!

useReducer + useContext

- By combining context with ***useReducer***, we can create the same behavior that Redux provides.
- Idea: Apply ***useReducer*** high up in our component hierarchy, use context to share data / dispatch farther down.

useReducer Example

demo/redux-without-redux/src/rootReducer.js

```
import * as t from "./actionTypes";

function rootReducer(state, action) {
  if (action.type === t.ADD_MEME) {
    return {
      ...state,
      memes: [
        ...state.memes, { ...action.meme }
      ]
    };
  }
  if (action.type === t.REMOVE_MEME) {
    return {
      ...state,
      memes: state.memes
                  .filter(m => m.id !== action.id)
    };
  }
  return state;
}

export default rootReducer;
```

demo/redux-without-redux/src/App.js

```
import React, { useReducer } from "react";
import NewMemeForm from "./NewMemeForm";
import MemeList from "./MemeList";
import rootReducer from "./rootReducer";
import DispatchContext from "./dispatchContext";
import MemeContext from "./memeContext";
import "./App.css";

function App() {
  const [state, dispatch] = useReducer(
    rootReducer,
    { memes: [] }
  );

  return (
    <div id="App">
      <DispatchContext.Provider value={dispatch}>
        <MemeContext.Provider value={state.memes}>
          <NewMemeForm />
          <MemeList />
        </MemeContext.Provider>
      </DispatchContext.Provider>
    </div>
  );
}

export default App;
```

demo/redux-without-redux/src/MemeList.js

```
import React, { useContext } from "react";
import Meme from "./Meme";
import MemeContext from "./memeContext";

function MemeList() {
  const memes = useContext(MemeContext);

  return (
    <div>
      {memes.map(m => (
        <Meme
          key={m.id}
          topText={m.topText}
          botText={m.bottomText}
          url={m.url}
          id={m.id}
        />
      ))}
    </div>
  );
}
```

demo/redux-without-redux/src/Meme.js

```
import React, { useContext } from "react";
import DispatchContext from "./dispatchContext";
import { removeMeme } from "./actions";
import "./Meme.css";

function Meme({ topText, botText, url, id }) {
  const dispatch = useContext(DispatchContext);
  const remove = () => dispatch(removeMeme(id));

  return (
    <div id="foo" className="Meme">
      <div className="container">
        <span className="text-t">{topText}</span>
        <img src={url} alt="a meme" />
        <span className="text-b">{botText}</span>
        <button onClick={remove}>
          DELETE
        </button>
      </div>
    </div>
  );
}
```

Multiple Contexts

- When following this pattern, it's recommended to separate ***dispatch*** into its own context
- State changes frequently, but ***dispatch*** never does, so having a separate ***DispatchProvider*** avoids unnecessary re-renders

React.memo

- Another way to boost performance is with ***React.memo***
- ***React.memo*** is a HOC that will prevent a component from re-rendering if its props are unchanged.
- ***React.memo*** still allows components to re-render if they have state or consume context

React.memo Example

demo/redux-without-redux/src/Meme.js

```
import React, { useContext } from "react";
import DispatchContext from "./dispatchContext";
import { removeMeme } from "./actions";
import "./Meme.css";

function Meme({ topText, botText, url, id }) {
  const dispatch = useContext(DispatchContext);
  const remove = () => dispatch(removeMeme(id));

  return (
    <div id="foo" className="Meme">
      <div className="container">
        <span className="text-t">{topText}</span>
        <img src={url} alt="a meme" />
        <span className="text-b">{botText}</span>
        <button onClick={remove}>
          DELETE
        </button>
      </div>
    </div>
  );
}

export default React.memo(Meme);
```

Webpack

Create React App includes Webpack:

- Lets you use JS modules (***import*** / ***export***)
- Combines your JS into one file
- Can easily use NPM modules in your JS

You can use this in your non-CRA projects: Webpack Getting Started

Babel

Babel transpiles JSX/ultra-modern JS into conventional JS

- You can experiment with this online
- Or install it via the command-line tools

Useful Add-Ons

PropTypes

Can document/verify that types of props are as expected:

```
$ npm install prop-types
```

```
import PropTypes from 'prop-types';

class Greeting extends Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

PropTypes docs

Styled Components

Can make "CSS-wrapped components" from your components:

```
const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: palevioletred;
`;

const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`;

class Demo extends Component {
  render() {
    return (
      <Wrapper>
        <Title> Hello World! </Title>
      </Wrapper>
    );
  }
}
```

Getting Started With Styled Components