CONTEXT API

YT Course by The Net Ninja / June 2019

Mind map by Ana-Maria Dobre

INTRO

Context API:
Clean and easy way to share state between components (without having to pass props down all of the time)

Hooks:
Tap into the inner workings of React in FC (that we can only normally do inside CC)*

Context API + Hooks => MAGIC!
=> Redux-like, state management logic, without having to install a third party library.

PRACTICE
Scaffold new React app (1) and add a few classes:
Navbar.js (2) // contains h1 and menu links in ul, li tags
BookList.js (3) // contains book list in ul, li tags
importing them into App.js (4)

```
(1)
C:\Workspace\nn\contextapp>npx create-react-app contextapp

(2)
import React, { Component, createContext} from 'react';

class Navbar extends Component
{
    render() {
        return (
            <nav>
                <h1>Context App</h1>
                <ul>
                    <li>Home</li>
                    <li>About</li>
                    <li>Contact</li>
                </ul>
            </nav>
        )
    }
}

export default Navbar;

(3)
import React, {Component} from 'react';

class BookList extends Component
{
    render() {
        return (
            <div className='book-list'>
                <ul>
                    <li>the way of kings</li>
                    <li>the name of the wind </li>
                    <li>the way of kings</li>
                </ul>
            </div>
        )
    }
}

export default BookList;

(4)
import logo from './logo.svg';
import './App.css';
import Navbar from './components/Navbar';
import BookList from './components/BookList';

function App() {
  return (
    <div className="App">
      <Navbar />
      <BookList />
    </div>
  );
}

export default App;
```

OVERVIEW

- Helps with SHARING STATE within a component tree
- Gives us a CENTRAL PLACE to store data / state & share it between components without having to pass it down as props
- Concept of storing data centrally, just like Redux
=> ALTERNATIVE TO REDUX
- Global type of data typically shared with Context Api: authenticated user, theme, preferred language

- Compare to props:
it can get messy doing it via props, as your component tree grows bigger

- Case study (Pic1) & comparison to props:
We want the bottom 4 components to share data from App.
If we pass the data down as props, we'd cover even components which don't use the props / state directly (these are acting like carriers).

Pic2
- create a new context (ThemeContext) in a new file
- provide it to our component tree (do this via Context Provider
// Context Provider is a React type; wraps whichever components needs access to it
- the wrapped components have access to the shared data (and no "carriers" needed)

ADDING A CONTEXT & PROVIDER

Under folder contexts > new file ThemeContext.js with content (1)
// our theme context will reside there
// To be used later inside App.js, like so (2)

Note:
When we create a context, we also need to create a **provider**, which is a tag that surrounds whichever components need to use that context

Comments about ThemeContext.js
- import React as usual
- import also the **createContext** function
// to create a context, stored in a constant (has no data yet)

- create a **CLASS component** that will contain some state
// suggestion for class name (opt): **context + suffix "Provider"**
- class will contain the state
// a few different properties inside state obj i.e. light, dark, etc
- from this class, return a JSX containing that provider tag i.e. **<ThemeContext.Provider>**
// **tag is the name of the context + dot + "Provider" suffix**
- on the provider tag, specify a **value** property, to which you can assign the state object
// we used the spread operator, like so: { ...this.state }

Go to App.js > surround Navbar & BookList by ThemeContext.Provider.

Back in ThemeContext.js, make sure to output children by doing {this.props.children}

In App.js, having Navbar nested inside ThemeContextProvider means Navbar is attached to the props of ThemeContextProvider

TIP:
Search for "React Developer Tools"; add extension
// Pic3

Have a look at the spread operator syntax with an example (3)

You can try out example from point (3) right here (4)

```
(1)
import React, { createContext, Component } from 'react';

export const ThemeContext = createContext();

class ThemeContextProvider extends Component {
    state = {
        isLightTheme: true,
        light: { syntax: '#555', ui: '#ddd', bg: '#eee' },
        dark: { syntax: '#ddd', ui: '#333', bg: '#555' }
    }
    render() {
        return(
            <ThemeContext.Provider value = {{...this.state}}>
                {this.props.children}
            </ThemeContext.Provider>
        );
    }
}

export default ThemeContextProvider;

(2)
import logo from './logo.svg';
import './App.css';
import Navbar from './components/Navbar';
import BookList from './components/BookList';
import ThemeContextProvider from './contexts/ThemeContext';

function App() {
  return (
    <div className="App">
      <ThemeContextProvider>
        <Navbar />
        <BookList />
      </ThemeContextProvider>
    </div>
  );
}

export default App;

(3)
const myObj = { isLight: true, light: 'blue', dark: 'black'}
var x = {...myObj}
console.log(x.light)
// returns "blue"

(4)
https://developer.mozilla.org/en-US/docs/Web/JavaScript
/Reference/Operators/Spread_syntax
```

ACCESSING CONTEXT

Two ways:
- using contextType (A)
- using context consumer (B)

We'll exemplify (A) for BookList (1)
// to be applied similarly for the other component, Navbar, see (2)

Notes about approach (A):
- preferred approach (for some) when using CC
- declare static **contextType** (mandatory name) inside comp and assign it specific context, in our case ThemeContext
- this.context should be equal to whatever you passed into the value inside ThemeContextProvider
- destructure this.context into separate variables
- define a new constant, theme // based on those variables (may use ternary operator)
- use theme variable at will

Note: Every time state changes in the context, anything consuming the ThemeContext will be updated as well.

Summary: introduce static prop contextType, set it equal to whatever context you want to use, then get access to that context object on this.

We'll exemplify (B) for BookList (3)
// to be applied similarly for the other component, Navbar, see (4)

Notes about approach (B):
- works with both CC and FC
- much like we have a <ThemeContext.Provider> tag when creating a context, we also have a consumer i.e. <ThemeContext.Consumer> (to consume the context).
- inside the consumer tag, it is expected to pass in a function, which takes in a context param and returns JSX

Summary: use the consumer tag of the ThemeContext previously created. Inside consumer, we use a fct that takes in that context object as a param. Therefore, inside this fct, we have access to that context object.

```
(1)
import React, {Component} from 'react';
import { ThemeContext } from '../contexts/ThemeContext';

class BookList extends Component
{
    static contextType = ThemeContext;
    render() {
        const { isLightTheme, light, dark } = this.context;
        const theme = isLightTheme? light : dark;

        return (
            <div className='book-list' style = {{color: theme.syntax, background: theme.bg}}>
                <ul>
                    <li>the way of kings</li>
                    <li>the name of the wind </li>
                    <li>the way of kings</li>
                </ul>
            </div>
        )
    }
}

export default BookList;

(2) see in Notes // N1

(3)
import React, {Component} from 'react';
import { ThemeContext } from '../contexts/ThemeContext';

class BookList extends Component
{
    render() {
        return(
            <ThemeContext.Consumer>{(context) => {
                const { isLightTheme, light, dark } = context;
                const theme = isLightTheme ? light : dark;
                return(
                    <div className='book-list' style = {{color: theme.syntax, background: theme.bg}}>
                        <ul>
                            <li>the way of kings</li>
                            <li>the name of the wind </li>
                            <li>the way of kings</li>
                        </ul>
                    </div>
                );
            }} </ThemeContext.Consumer>
        );
    }
}

export default BookList;

(4) see in Notes // N2
```

UPDATING CONTEXT DATA

How do we go about changing shared state?
Idea: press a button to change the theme.

Create (inside components folder) ThemeToggle.js (1)
// purpose: output a button that user may use to change theme

Inside ThemeContext.js, define an array function called toggleTheme (2)
- that fct must live inside the provider class
- inside fct, call this.setState to change isLightTheme to its opposite value
- I want to pass this fct down as a property inside the obj associated with the value property
=> now we should have access to this fct inside any comp that consumes this context

Note: you may also use anyOtherName: this.toggleTheme but in this case, call anyOtherName from consumers

Summary:
Interact with the data by defining a fct inside context provider class, that can edit the state; then pass that fct into the value prop of the provider, therefore when we consume the context, we can have access to that fct.

```
(1)
import React, { Component } from "react";
import { ThemeContext } from "../contexts/ThemeContext";

class ThemeToggle extends Component
{
    static contextType = ThemeContext;
    render() {
        const { toggleTheme } = this.context;
        return(
            <button onClick={toggleTheme}>Toogle the theme</button>
        );
    }
}

export default ThemeToggle;

(2)
import React, { createContext, Component } from 'react';

export const ThemeContext = createContext();

class ThemeContextProvider extends Component {
    state = {
        isLightTheme: true,
        light: { syntax: '#555', ui: '#ddd', bg: '#eee' },
        dark: { syntax: '#ddd', ui: '#333', bg: '#555' }
    }
    toggleTheme = () => {
        this.setState({ isLightTheme: !this.state.isLightTheme});
    }
    render() {
        return(
            <ThemeContext.Provider value = {{...this.state, toggleTheme: this.toggleTheme}}>
                {this.props.children}
            </ThemeContext.Provider>
        );
    }
}

export default ThemeContextProvider;
```

Notes

FC: functional components
CC: class components
***: refers to things like:**
useState, or kind of tapping into a lifecycle method
opt: optional
fct: function

Ideas to research later:
- component composition