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 new file called BookList.js, under new folder components (2)
// contains book list in ul, li tags...
Import file into App.js (3)

```
cd c(1)
C:\Workspace\nn\contextapp>

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

class BookList extends Component
{
  render() {
    return (
      <div className='book-list'>
        <ul>
          <li>Three men in a boat</li>
          <li>Rumi's book of poetry</li>
        </ul>
      </div>
    );
  }
}

export default BookList;

(3)
import './App.css';
import BookList from './components/BookList';

function App() {
 return (
  <div className="App">
    <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

Case study & comparison to props:
- We want the bottom 4 components of a large comp tree to share data from App.
- If we pass the data down as props, we would cover even components which don't need to use the props directly (these would be acting like "carriers").
=> it can get messy doing it via props, as your component tree grows bigger

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

ADDING A CONTEXT & PROVIDER

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

export const AuthContext = createContext();

class AuthContextProvider extends Component
{
 state = {
  isAuthenticated: false,
  anotherProperty: "geronimo"
 }

 render()
 {
  return (
   <AuthContext.Provider value={{...this.state}}>
    {this.props.children}
   </AuthContext.Provider>
  );
 }
}

export default AuthContextProvider;

(2)
import logo from './logo.svg';
import './App.css';
import BookList from './components/BookList';
import AuthContextProvider from './contexts/AuthContext';

function App() {
 return (
  <div className="App">
   <AuthContextProvider>
      <BookList />
   </AuthContextProvider>
  </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
```

Under folder contexts > new file AuthContext.js with content (1)
// our auth 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 AuthContext.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 (optional): **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.
<AuthContext.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 BookList by AuthContext.Provider.

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

In App.js, having BookList nested inside AuthContextProvider means BookList is attached to the props of AuthContextProvider

TIP:
Search for "React Developer Tools"; add extension to browser

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

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

ACCESSING CONTEXT

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

class BookList extends Component
{
  static contextType = AuthContext;
  render() {
    const {isAuthenticated, anotherProperty} = this.context;
    return (
      <div className='book-list'>
        {isAuthenticated? 'Logged in': 'Logged out'}
        <ul>
          <li>Three men in a boat</li>
          <li>Rumi's book of poetry</li>
          <li>{anotherProperty}</li>
        </ul>
      </div>
    );
  }
}

export default BookList;

(2)
import React, {Component} from 'react';
import { AuthContext } from '../contexts/AuthContext';

class BookList extends Component
{
  render() {
    return(
      <AuthContext.Consumer>{(context) => {
        const { isAuthenticated } = context;
        return(
          <div className='book-list'>
            <ul>
              <li>Three men in a boat</li>
              <li>Rumi's book of poetry</li>
            </ul>
            <br/>
            <div>
                { isAuthenticated? 'Logged in': 'Logged out' }
            </div>
          </div>
        );
      }} </AuthContext.Consumer>
    );
  }
}

export default BookList;
```

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

We'll exemplify (A) for BookList (1)

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 AuthContext
- this.context should be equal to whatever you passed into the value inside AuthContextProvider
- destructure this.context into separate variables

Note: Every time state changes in the context, anything consuming the AuthContext 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 (2)

Notes about approach (B):
- works with both CC and FC
- much like we have a <AuthContext.Provider> tag when creating a context, we also have a consumer i.e. <AuthContext.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 AuthContext 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.**

UPDATING CONTEXT DATA

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

class AuthToggle extends Component
{
  static contextType = AuthContext;

  render()
  {
    const {toggleAuth} = this.context;

    return(

      <button onClick={toggleAuth}>Log in/out</button>
    );
  }
}

export default AuthToggle;


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

export const AuthContext = createContext();

class AuthContextProvider extends Component
{
  state = {
    isAuthenticated: false,
    anotherProperty: "geronimo"
  }

  toggleAuth = () =>
  {
    this.setState({isAuthenticated: !this.state.isAuthenticated})
  }

  render()
  {
    return(
      <AuthContext.Provider value={{...this.state, toggleAuth: this.toggleAuth}}>
        {this.props.children}
      </AuthContext.Provider>
    );
  }
}

export default AuthContextProvider;
```

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

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

Inside AuthContext.js, define an array function called toggleAuth (2)
- that fct must live inside the provider class
- inside fct, call this.setState to change isAuthenticated 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.toggleAuth 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.

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