React Context

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Context
- UseReducer
- LocalStorage

Passing data from parent to child components

- Global data: variables, database, files
- Props
- Navigation
- React Context

React context

- Create context
- Context Provider
- Context Consumer

Create context

- Using the function createContext to create a context object that can be shared between Provider and Consumer
- You can initialize the default value for context

```
const MyContext = createContext({});
MyContext.displayName = "MyContext";
```

Context Provider

- Provide the context to all descendants
- Set the value for context by using the prop 'value'

```
function App() {
const [state, setState] = useState({ count: 1 });
return (
<div className="App">
<MyContext.Provider value={state}>
<Consumer1 />
<Consumer2/>
</MyContext.Provider>
</div>
);
}
```

Context Consumer

- Should be placed inside any target provider

```
function Consumer1(){
return(
<MyContext.Consumer>
{value => <p>Consumer</p>}
</MyContext.Consumer>
)
}
function Consumer2() {
const context = useContext(MyContext);
return <p>Consumer</p>;
}
```

Consuming Multiple Contexts

```
// Code snipped of the provider
...
return (
  <ThemeContext.Provider value={theme}>
    <UserContext.Provider value={signedInUser}>
      <Layout />
    </UserContext.Provider>
  </ThemeContext.Provider>
);
...

// A component may consume multiple contexts
function Content() {
  return (
    <ThemeContext.Consumer>
      {theme => (
        <UserContext.Consumer>
          {user => (
            <ProfilePage user={user} theme={theme} />
          )}
        </UserContext.Consumer>
      )}
    </ThemeContext.Consumer>
  );
}
```

Football App

- How to avoid unintentional re-rendering consumers when re-rendering the App (provider)

Football App

- First Version of App Component (Provider)

```
const [state, setState] = useState({
count1: 0,
count2: 0
});

<MyContext.Provider value={state}>
<Team1 name="Chicago"/>
<Team2 name="Iowa"/>
</MyContext.Provider>
```

Football App

```
const Team1 = function({name}) {
console.log('rendered  ${name}');
const state = useContext(MyContext);
return (
<div>
<p>
{name}: {state}
</p>
<hr/>
</div>
);
};
```

```
const Team2 = function({name}) {
console.log('rendered  ${name}');
const state = useContext(MyContext);
return (
<div>
<p>
{name}: {state}
</p>
<hr/>
</div>
);
};
```

Football App

- Fixed Version of App Component (Provider)

```
<MyContext.Provider value={state.count1}>
<Team1 name="Chicago"/>
</MyContext.Provider>
<MyContext.Provider value={state.count2}>
<Team2 name="Iowa"/>
</MyContext.Provider>
```

Football

- Fixed Version of Consumers

const Team1 = React.memo(…)

const Team2 = React.memo(…)

Before you use React Context

- For front-end application, re-rendering is an expensive task and it can cause the poor experience with users like blinking, slow response...
- Use the context for simple data that do not change often such theme, language, and user preference.
- Provide only a portion of data for consumers

useReducer

- It is an alternative for useState if the state is complex
- Define the reducer function

```
function counterReducer(state, action) {
switch (action.type) {
case ACTIONS.INC_COUNTER1:
return { ...state, count1: state.count1 + 1 };
case ACTIONS.INC_COUNTER2:
return { ...state, count2: state.count2 + 1 };
default:
return state;
}
}
```

useReducer

- Use it inside component

```
const [state, dispatch] = useReducer(counterReducer, {
count1: 0,
count2: 0
});
const increaseCount1 = () => {
dispatch({ type: ACTIONS.INC_COUNTER1 });
}
```

localstorage

- It is a storage (key-value pairs) for a website
- Browsers keep this data even after closing the website
- Only store the data as string
- Each domain has its own local storage
- It is a global object which is supported by most modern browsers like Chrome, Firefox, Safari, IE, Edge...

localStorage

- API

localStorage.setItem(key, value): set key-value pair to localstorage

localStorage.getItem(key): Return the value of this key, or null

localStorage.removeItem(key): Remove key-value pair

localStorage.clear(): Cleanup the localstorage for this domain

localStorage

- Store data to the localStorage

```
try {
localStorage.setItem('key', 'value');
console.log('Data stored successfully.');
} catch (error) {
console.error('Error storing data:', error);
}
```

localstorage

- Retrieve data for the localStorage

```
try {
const value = localStorage.getItem('key');
if (value !== null) {
console.log('Retrieved value:', value);
} else {
console.log('Key not found in localStorage.');
}
} catch (error) {
console.error('Error retrieving data:', error);
}
```

localstorage

- Remove data from the localStorage

```
try {
localStorage.removeItem('key');
console.log('Data removed successfully.');
} catch (error) {
console.error('Error removing data:', error);
}
```

Summary

- Context
- UseReducer
- LocalStorage