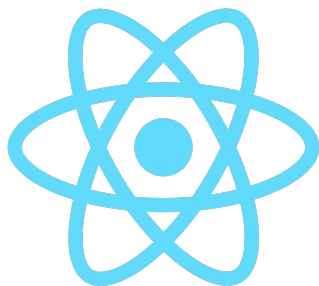


React Context vs Redux



Tomasz Fiechowski

 tfiechowski

Agenda

Why Context? 🤔

Context 🧐

- introduction,
- how to use

Usage of many Contexts 👤👤

Redux 📊

- comparison with Context

Testing 🛠️

- `InnerComponent` pattern, API as a prop

Why Context? 🤔



Dan Abramov @dan_abramov · Sep 11



When 10 people say “I dislike Redux” they might mean 10 completely different things. And they often have nothing to do with Redux itself, but with how the code they worked with is structured, or with the examples they learned from.



18



19



192



Dan Abramov @dan_abramov · Sep 11



If you copy paste some action creators and reducers handling FETCH actions over and over and over again, you’re probably using Redux in a different way than I imagined people would do. I’m sorry for all the repetitive code you felt you needed to write. That’s my fault.



17



25



204



Dan Abramov @dan_abramov · Sep 11



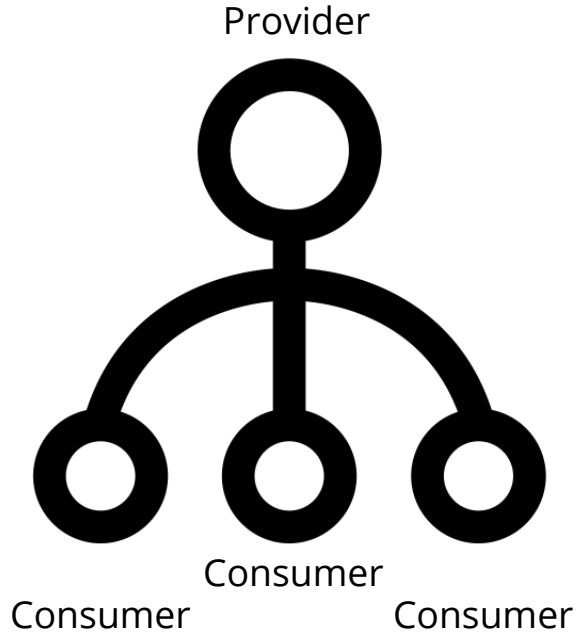
Same goes for having 20 actions with types like SET_A, SET_B, SET_C, and dispatching them from a long-ass async action creator. Also not how I imagined people would do it. Again, sorry I didn’t explain this clearly.

React Context

React Context

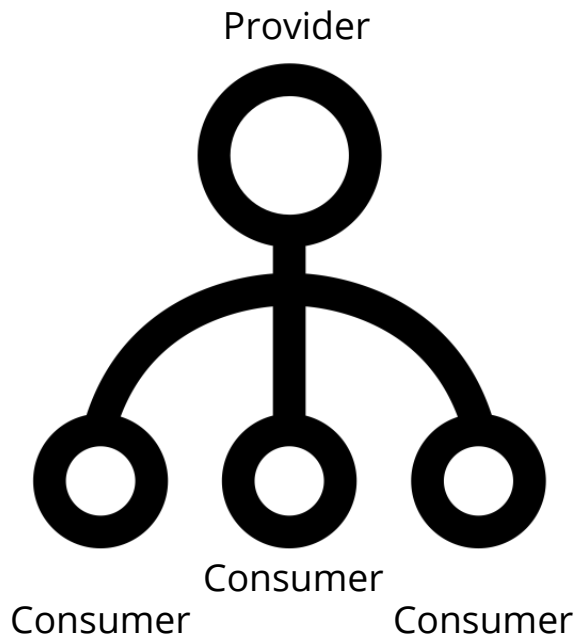
= Provider + Consumers

React Context

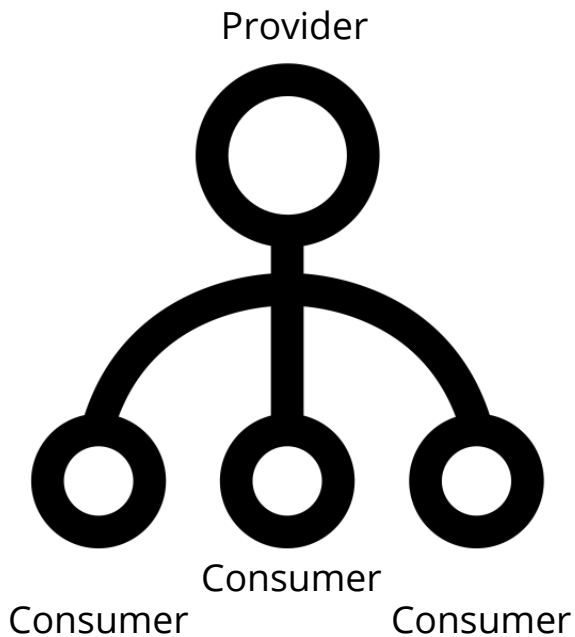


React Context

```
import React, { Component, createContext } from "react";  
  
const Context = createContext({});
```



React Context



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

```
const Context = createContext({});
```

```
export class PhotosProvider extends Component {
```

```
  constructor(props) {
```

```
    super(props);
```

```
    this.state = {
```

```
      photos: [],
```

```
    };
```

```
  }
```

```
  render() {
```

```
    return (
```

```
      <Context.Provider value={this.state}>
```

```
        {this.props.children}
```

```
      </Context.Provider>
```

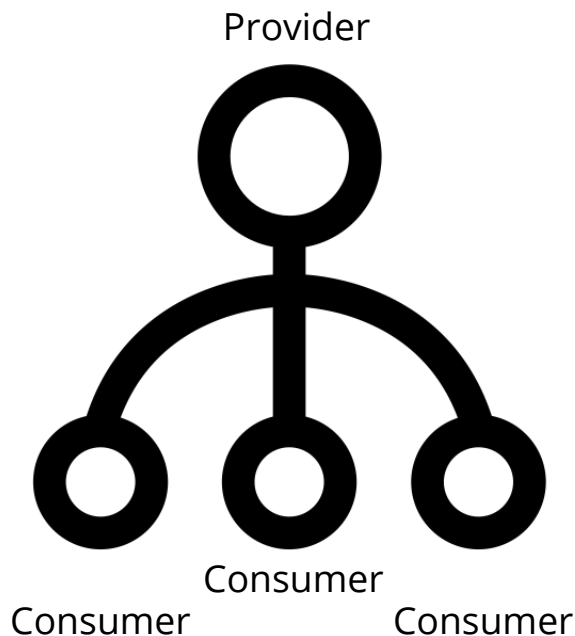
```
    );
```

```
  }
```

```
}
```

```
export default Context;
```

React Context + actions



```
import React, { Component, createContext } from "react";
import api from "./api";

const Context = createContext({});

export class PhotosProvider extends Component {
  constructor(props) {
    super(props);

    this.state = {
      photos: [],
      fetchPhotos: this.fetchPhotos
    };
  }

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

  fetchPhotos = async () => {
    const photos = await api.fetchPhotos();
    this.setState(state => ({ photos: [state.photos, ...photos] }));
  };
}

export default Context;
```

Provider

```
import { PhotosProvider } from "../contexts/PhotosContext/PhotosContext";
```

```
ReactDOM.render(  
  <PhotosProvider>  
    <App />  
  </PhotosProvider>,  
  document.getElementById("root")  
);
```

What's with the Consumers then?



React Context - PhotosList helper

```
export default function PhotosList({ photos, fetchPhotos }) {  
  return (  
    <div>  
      {photos.map(photo => (  
        <div key={photo.id}>{photo.title}</div>  
      ))}  
  
      <button onClick={fetchPhotos}>Fetch more photos</button>  
    </div>  
  );  
}
```

React Context - Context.Consumer

```
import React from "react";
import PhotosList from "../components/PhotoList";
import PhotosContext from "../contexts/PhotosContext";

export default function PhotoListContextConsumer() {
  return (
    <PhotosContext.Consumer>
      ({ photos, fetchPhotos }) => (
        <PhotosList photos={photos} fetchPhotos={fetchPhotos} />
      )
    </PhotosContext.Consumer>
  );
}
```

React Context - createContext

```
import React, { Component } from "react";
import PhotosList from "../components/PhotoList";
import PhotosContext from "../contexts/PhotosContext";

export default class PhotoListContextType extends Component {
  static contextType = PhotosContext;

  render() {
    const { photos, fetchPhotos } = this.context;

    return <PhotosList photos={photos} fetchPhotos={fetchPhotos} />;
  }
}
```

React Context - useContext (hook)

```
import React, { useContext } from "react";
import PhotosList from "../components/PhotoList";
import PhotosContext from "../contexts/PhotosContext";

export default function PhotoListUseContext() {
  const { photos, fetchPhotos } = useContext(PhotosContext);

  return <PhotosList photos={photos} fetchPhotos={fetchPhotos} />;
}
```


React Context - summary

1. Context initialization

```
const Context = createContext({});
```

2. Provider render

```
ReactDOM.render(  
  <PhotosProvider><App /></PhotosProvider>,  
  document.getElementById("root")  
) ;
```

3. Access to the Context through: Consumer, contextType or useContext.

4. Profit!

React Context - summary

```
    constructor(props) {  
      super(props);  
  
      this.state = {  
        photos: [],  
        fetchPhotos: this.fetchPhotos  
      };  
    }  
  
    render() {  
      return (  
        <Context.Provider value={this.state}>  
          {this.props.children}  
        </Context.Provider>  
      );  
    }  
  }  
  
export default function PhotoListUseContext() {  
  const { photos, fetchPhotos } = useContext(PhotosContext);  
  
  return <PhotosList photos={photos} fetchPhotos={fetchPhotos} />;  
}
```

The diagram illustrates the flow of data in a React application using Context. It shows three code snippets. The first snippet is the `constructor` method of a class, which initializes `this.state` with an empty array `photos` and a `fetchPhotos` function. The second snippet is the `render` method, which returns a `<Context.Provider>` component. The `value` prop of the `Provider` is `this.state`. The third snippet is a function component `PhotoListUseContext` that uses `useContext(PhotosContext)` to retrieve the `photos` and `fetchPhotos` values. Two arrows indicate the data flow: one from `this.state` in the `render` method to the `value` prop of the `Context.Provider`, and another from the `value` prop of the `Context.Provider` to the `useContext` hook in the `PhotoListUseContext` function.

React Context - summary

```
render() {  
  return (  
    <Context.Provider value={{ photos: [] }}>  
      {this.props.children}  
    </Context.Provider>  
  );  
}
```



<https://reactjs.org/docs/context.html#caveats>

Usage of many Contexts



React Context - multiple contexts

Providers

```
render(  
  <BrowserRouter>  
    <ChangeRequestProvider>  
      <AvailabilityProvider>  
        <DeadlineProvider api={deadlineApi}>  
          <ColumnProvider>  
            <ShiftProvider>  
              <UserProvider>  
                <Page />  
              </UserProvider>  
            </ShiftProvider>  
          </ColumnProvider>  
        </DeadlineProvider>  
      </AvailabilityProvider>  
    </ChangeRequestProvider>  
  </BrowserRouter>,  
  window.document.getElementById('root'),  
);
```

Consumers

```
return (  
  <UserConsumer>  
    ({ currentTeam, user, isAdminInCurrentTeam, getUser }) => (  
      <ColumnConsumer>  
        ({ addColumns, columns, getColumn }) => (  
          <DeadlineConsumer>  
            ({ fetchDeadlines, deadlines }) => (  
              <ShiftConsumer>  
                ({  
                  fetchShifts,  
                  giveAwayShift,  
                  modifyShifts,  
                  shifts,  
                  suggestAdjustment,  
                  removeAdjustment,  
                  acceptAdjustment,  
                  rejectAdjustment,  
                  removeAdjustmentSuggestion,  
                  saveAdjustment,  
                  takeShift,  
                  undoGiveAwayShift,  
                }) => (  
                  <AvailabilityConsumer>  
                    ({ availabilities, fetchAvailabilities }) => (  

```

React Context - Consumer nesting

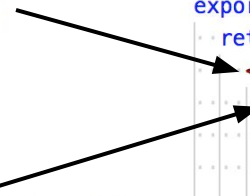
```
return (  
  <ThemeConsumer>  
    ({ { theme } }) => (  
      <PhotosConsumer>  
        ({ { photos, fetchPhotos } }) => (  
          <PhotosPage photos={photos} fetchPhotos={fetchPhotos} theme={theme} />  
        )  
      )  
    )  
  )  
</ThemeConsumer>  
);
```

React Context - Consumer nesting

Helper:

```
function ContextConsumer({ children }) {  
  return (  
    <PhotosContext.Consumer>  
      {photos => (  
        <ThemeContext.Consumer>  
          {theme => children({ photos, theme })}  
        </ThemeContext.Consumer>  
      )}  
    </PhotosContext.Consumer>  
  );  
}
```

```
export default function PhotoListContextConsumer() {  
  return (  
    <ContextConsumer>  
      ({ { photos: { photos, fetchPhotos }, theme: { theme } }) => (  
        <PhotosList  
          photos={photos}  
          fetchPhotos={fetchPhotos}  
          theme={theme}  
        />  
      )  
    </ContextConsumer>  
  );  
}
```



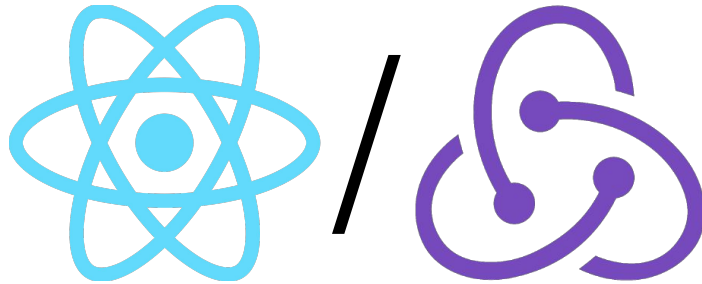
The diagram consists of two black arrows. The first arrow originates from the `ContextConsumer` function definition in the left code block and points to the `<ContextConsumer>` JSX element in the right code block. The second arrow originates from the `children` prop of the `ContextConsumer` function in the left code block and points to the function argument `{ { photos: { photos, fetchPhotos }, theme: { theme } }}` in the right code block.

React Context - Consumer nesting

```
export default function PhotoListUseContext() {  
  const { photos, fetchPhotos } = useContext(PhotosContext);  
  const { theme } = useContext(ThemeContext);  
  
  return <PhotosList photos={photos} fetchPhotos={fetchPhotos} theme={theme} />;  
}
```



React Context/Redux



React Context/Redux

```
import React, { Component, createContext } from "react";
import api from "../api";

const Context = createContext({});

export class PhotosProvider extends Component {
  constructor(props) {
    super(props);

    this.state = {
      photos: [],
      fetchPhotos: this.fetchPhotos
    };
  }

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

  fetchPhotos = async () => {
    const photos = await api.fetchPhotos();
    this.setState(state => ({ photos: [state.photos, ...photos] }));
  };
}

export default Context;
```

```
1 import api from "../contexts/PhotosContext/api";
2
3 export function addPhotos(photos) {
4   return {
5     type: "photos/ADD_PHOTOS",
6     payload: {
7       photos
8     }
9   };
10 }
11
12 export function fetchPhotos() {
13   return async dispatch => {
14     const photos = await api.fetchPhotos();
15     dispatch(addPhotos(photos));
16   };
17 }
18
19 function getInitialState() {
20   return {
21     photos: []
22   };
23 }
24
25 export default function photosReducer(state = getInitialState(), action) {
26   const { type, payload } = action;
27   switch (type) {
28     case "photos/ADD_PHOTOS":
29       return Object.assign({}, state, {
30         photos: [...state.photos, payload.photos]
31       });
32     default:
33       return state;
34   }
35 }
```

React Context/Redux + components

```
1 import React, { useContext } from "react";
2 import PhotosList from "../components/PhotoList";
3 import PhotosContext from "../contexts/PhotosContext";
4
5 export default function PhotoListUseContext() {
6   const { photos, fetchPhotos } = useContext(PhotosContext);
7
8   return <PhotosList photos={photos} fetchPhotos={fetchPhotos} />;
9 }
10
11
12
13
```

```
1 import connect from "react-redux";
2 import PhotosList from "../components/PhotoList";
3 import { fetchPhotos } from "../redux/Photos";
4
5 const mapStateToProps = state => ({ photos: state.photos });
6
7 const mapDispatchToProps = () => ({ fetchPhotos });
8
9 export default connect(
10   mapStateToProps,
11   mapDispatchToProps
12 )(PhotosList);
13
```

Testing! 

Context recap

import api from "../api"



```
import React, { Component, createContext } from "react";
import api from "../api";

const Context = createContext({});

export class PhotosProvider extends Component {
  constructor(props) {
    super(props);

    this.state = {
      photos: [],
      fetchPhotos: this.fetchPhotos
    };
  }

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

  fetchPhotos = async () => {
    const photos = await api.fetchPhotos();
    this.setState(state => ({ photos: [state.photos, ...photos] }));
  };
}

export default Context;
```

Testing: Inner Component

api as a prop

```
import React, { Component, createContext } from "react";
import api from "../api";

const Context = createContext({});

export class PhotosProviderInner extends Component {
  static propTypes = {
    api: PropTypes.object
  };

  constructor(props) { ...
  }

  render() { ...
  }

  fetchPhotos = async () => {
    const photos = await this.props.api.fetchPhotos();
    this.setState(state => ({ photos: [state.photos, ...photos] }));
  };
}

export function PhotosProvider(props) {
  return <PhotosProviderInner api={api} {...props} />;
}

export default Context;
```

Testing: Redux

```
import configureMockStore from 'redux-mock-store'
import thunk from 'redux-thunk'
import * as actions from '../actions/ToDoActions'
import * as types from '../constants/ActionTypes'
import fetchMock from 'fetch-mock'
import expect from 'expect' // You can use any testing library

const middlewares = [thunk]
const mockStore = configureMockStore(middlewares)

describe('async actions', () => {
  afterEach(() => {
    fetchMock.restore()
  })

  it('creates FETCH_TODOS_SUCCESS when fetching todos has been done', () => {
    fetchMock.getOnce('/todos', {
      body: { todos: ['do something'] },
      headers: { 'content-type': 'application/json' }
    })

    const expectedActions = [
      { type: types.FETCH_TODOS_REQUEST },
      { type: types.FETCH_TODOS_SUCCESS, body: { todos: ['do something'] } }
    ]
    const store = mockStore({ todos: [] })

    return store.dispatch(actions.fetchTodos()).then(() => {
      // return of async actions
      expect(store.getActions()).toEqual(expectedActions)
    })
  })
})
```

Testing: Context

```
import { mount } from "enzyme";
import React from "react";
import { PhotosProviderInner } from "../PhotosContext";

describe("PhotosContext", () => {
  it("Should properly update photos after fetching", async () => {
    const api = {
      fetchPhotos: () => [{ id: 37, title: "Back in Black" }]
    };

    const wrapper = mount(<PhotosProviderInner api={api} />);
    await wrapper.instance().fetchPhotos();
    const { photos } = wrapper.state();

    expect(photos).toMatchSnapshot();
  });
});
```


Project + presentation (PDF):

`github.com/tfiechowski/meetjs-wdi-context`



plzzz: `bit.ly/meetjs-wdi-context`

`tomasz.fiechowski@outlook.com`