Webinar 6: Redux Middleware

@Carlos.Mentor.ReactND (Carlos Loureda)

Middleware

Wikipedia definition:

- **Middleware** is computer software that provides services to software applications beyond those available from the operating system. It can be described as "**software glue**".

“User” definition:

- Software services that “glue together” separate features in existing software. Something that is in the middle of something.

Redux Middleware

“… a third-party extension point between dispatching an action, and the moment it reaches the reducer.”

[Action] --> **[Middleware]** --> [Dispatcher]

Why Redux Middleware ?

Redux middleware appears to solve the problems we have when we want features like:

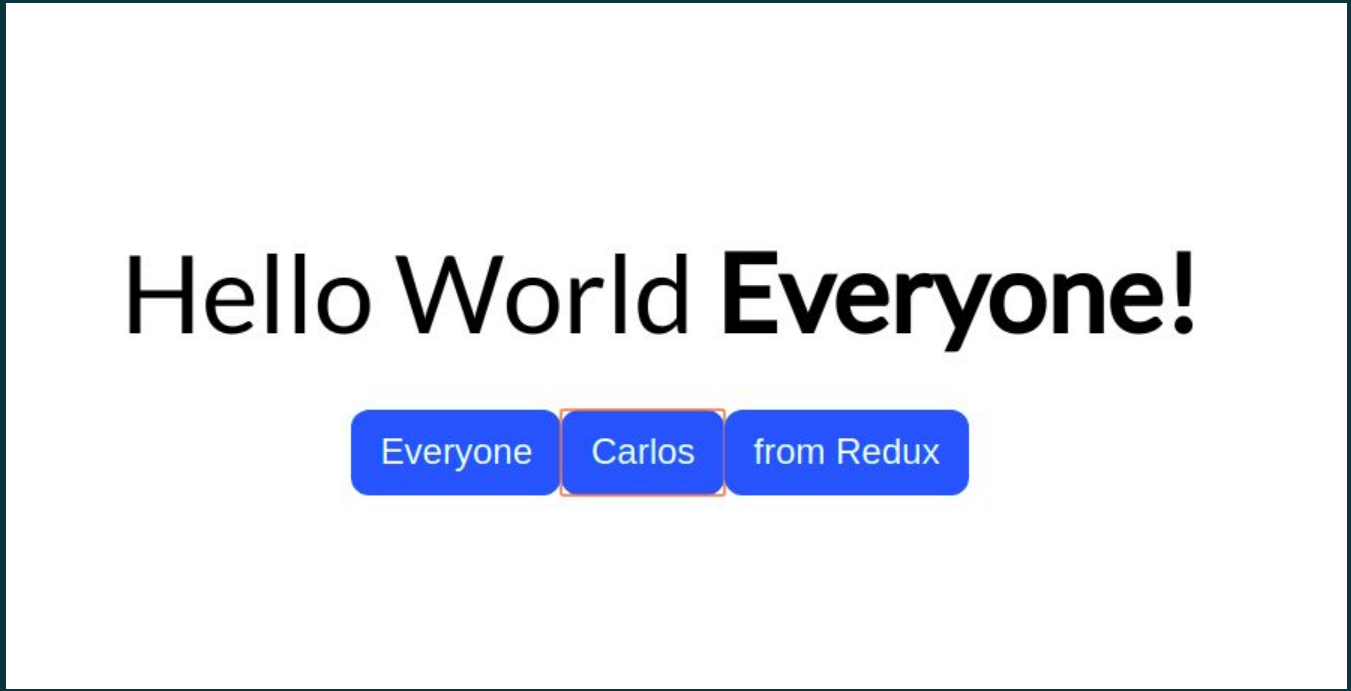
- producing a side effect
- redirecting the action
- dispatching supplementary actions

Implementation

Think about adding a checker in our hello-redux-world app.

We can implement by itself, but this means that we have to change a lot of the App code so it is better to implement it with Redux library.

Example 1: Add a checker



BRANCH: [adding-checker]

/src/components/ButtonGroup.js

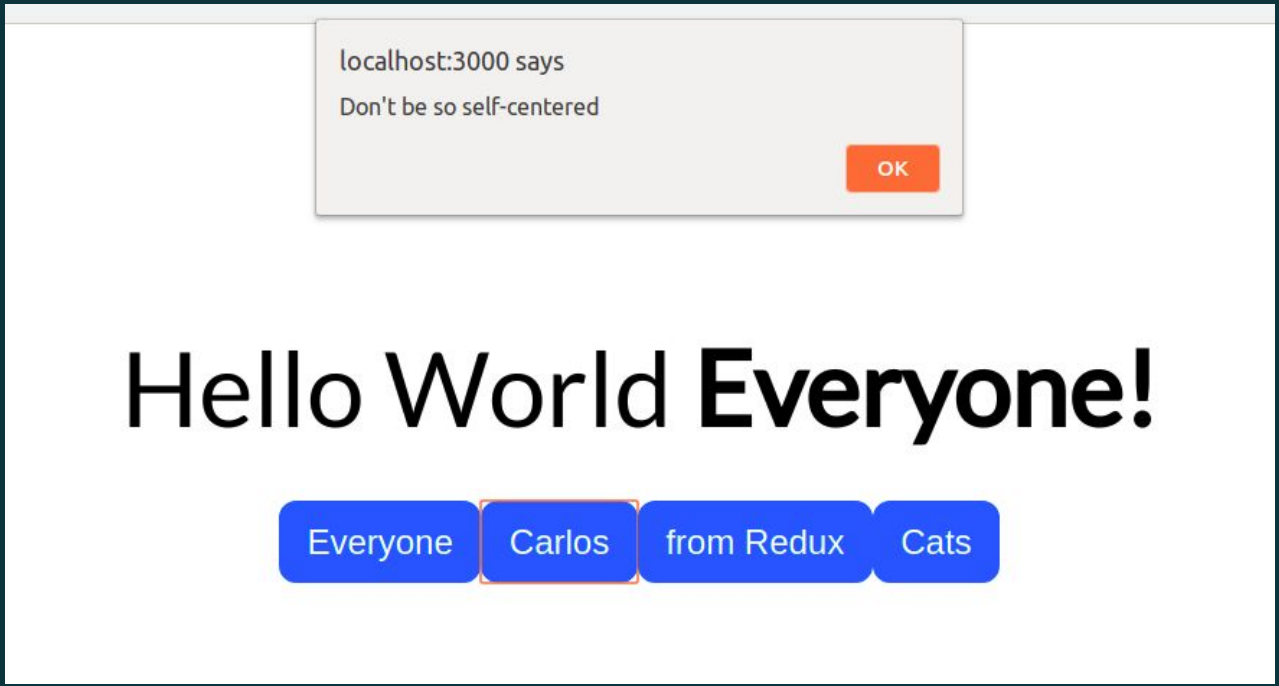
```
const onChangeName = e => {
 store.dispatch(setName(e.target.dataset.name));
};
```

/src/components/ButtonGroup.js

```
const checkAndDispatch = (store, action) => {
 if (action.type === "SET_NAME" && action.name === "Carlos") {
    return alert("Don't be so self-centered");
 }
 store.dispatch(action);
};

const onChangeName = e => {
 checkAndDispatch(store, setName(e.target.dataset.name));
};
```

Example 1: Add a checker



BRANCH: adding-checker

Example 2: Fetch an API

Example of Asynchronous calls.

We want to search for the name we are saying hello in a pictures API website, if we get results we show the image.

BRANCH: adding_fetch_api

/src/components/ButtonGroup.js

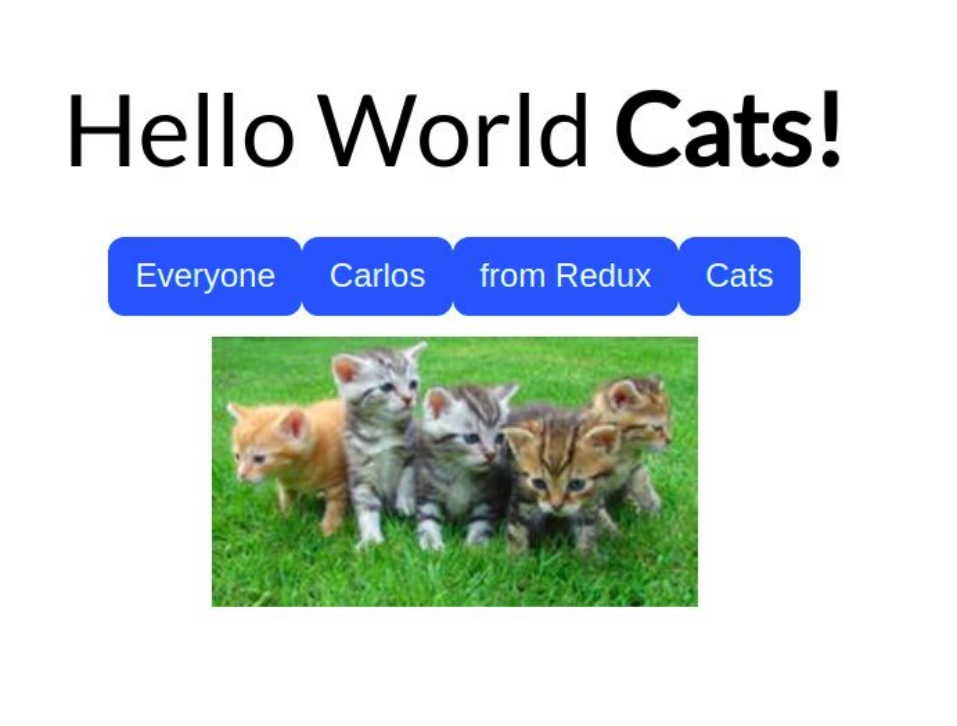
```
state = {
  photo: ""
};

fetchPhotos = query => {
  const options = {
    headers: {
      Authorization: PEXELS_CLIENT_ID
    }
  };
  fetch(`https://api.pexels.com/v1/search?query=${query}&per_page=1&page=1`, options)
    .then(response => response.json())
    .then(response => {
      if (
        response.total results &&
        response.photos &&
        response.photos.length
      ) {
        this.setState({ photo: response.photos[0].src.small});
      } else {
        this.setState({photo: ""});
      }
    })
};

onChangeName = e => {
  let query = e.target.dataset.name;
  this.fetchPhotos(query);
  this.checkAndDispatch(store, setName(query));
};
```

Also converted Functional component into a Class Component

Example 2: Fetch an API



BRANCH: [adding_fetch_api]

Example 2: Fetch an API - **Thunk**

Add **thunk** for doing Asynchronous requests

BRANCH: adding_thunks_for_picture

What's redux-thunk ?

It is redux middleware library. Redux only knows how to use normal actions creators, those which return normal-plain objects.

When we want to concatenate action calls or do asynchronous calls we need to be able to make this action creators return functions, in the *redux-thunk* example these are thunks.

A thunk is a “function” that will return a value that is calculated on runtime.

Other libraries: redux-saga, redux-promise, redux-observable

yarn add redux-thunk

/src/store/index.js

```
import { createStore, applyMiddleware } from "redux";

import thunk from "redux-thunk";

export const store = createStore(
 reducer,
 initialState,
 applyMiddleware(thunk)
);
```

/src/actions/index.js

```
export const setName = text => ({ type: "SET_NAME", name: text });
```

/src/actions/index.js

```
export const fetchPhotoFromAPI = query => dispatch => {
 const options = {
  headers: {  Authorization: “SET YOUR PEXEL ID” }
 };
 fetch(`https://api.pexels.com/v1/search?query=${query}&per_page=1&page=1`, options)
  .then(response => response.json())
  .then(response => {
   let photo_url = "";
   if (response.total_results && response.photos && response.photos.length) {
    photo_url = response.photos[0].src.small;
   }
   dispatch(addPhoto(photo_url));
  });
};

export const ADD_PHOTO = "ADD_PHOTO";
export const addPhoto = photo_url => ({ type: ADD_PHOTO, photo_url: photo_url });
```

This is the **thunk,** the function that will be called whenever the query is resolved.

/src/components/ButtonGroup.js

```
onChangeName = e => {
  let query = e.target.dataset.name;
  this.fetchPhotos(query);
  this.checkAndDispatch(store, setName(query));
};
```

/src/components/ButtonGroup.js

```
import { setName, fetchPhotoFromAPI } from "../actions";
. . .

 onChangeName = e => {
  let query = e.target.dataset.name;
  store.dispatch(fetchPhotoFromAPI(query));
  this.checkAndDispatch(store, setName(query));
 };
```

/src/reducers/index.js

```
import { SET_NAME, ADD_PHOTO } from "../actions";
export default (state, action) => {
 console.log("action:", action);
 switch (action.type) {
   case SET_NAME:
     return {
       ...state,
       name: action.name
     };
   case ADD_PHOTO:
     return {
       ...state,
       photo_url: action.photo_url
     };
   default:
     return state;
 }
};
```

Example 1: checker as middleware

Move the checker into a custom middleware

BRANCH: adding_middleware_checker

```
function checker(store) {
    return function(next) {
        return function(action) {
            // ...
            return next(action);
        };
    };
}


const checker = store => next => action => {
 };
```

This first function.

Called when calling **applyMiddleware**. It has **store** so middleware can access the store.

Argument: store instance

```
function checker(store) {
  return function(next) {
    return function(action) {
      // ...
      return next(action);
    };
  };
}


const checker = store => next => action => {
 };
```

This **2nd** function.

This is the following action to be run after that middleware

The argument is a function that needs to be called with an action as parameter, so that action later will be sent to the next middleware.

```
function checker(store) {
    return function(next) {
        return function(action) {
            // ...
            return next(action);
        };
    };
}


const checker = store => next => action => {
 };
```

Where the logic for our middleware lives.

The action that **dispatch** was called with.

```
function checker(store) {
    return function(next) {
        return function(action) {
            // ...
            return next(action);
        };
    };
}


const checker = store => next => action => {
 };
```

After the logic of our middleware is runned, we can break the cascade or go to the next.

The value to use as the return value of the dispatch call

/src/middleware/checker.js

```
const checker = store => next => action => {
 if (action.type === "SET_NAME" && action.name === "Carlos") {
   return alert("Don't be so self-centered");
 }
 next(action);
};
export default checker;
```

/src/store/index.js

```
import checker from "../middleware/checker";
. . .
export const store = createStore(
 reducer,
 initialState,
 applyMiddleware(checker, thunk)
);
```

Now we remove the checker function call and definition from ButtonGroup.js as it will be runned ‘automatically’

/src/components/ButtonGroup.js

```
onChangeName = e => {
  let query = e.target.dataset.name;
  store.dispatch(fetchPhotoFromAPI(query));
  this.checkAndDispatch(store, setName(query));
};
```

/src/components/ButtonGroup.js

```
onChangeName = e => {
  let query = e.target.dataset.name;
  store.dispatch(fetchPhotoFromAPI(query));
  store.dispatch(setName(query));
};
```

Last improvement

Putting previous actions inside an unique action

BRANCH: merging_action_calls

/src/actions/index.js

```
export const SAY_HELLO = "SAY_HELLO";

export const sayHello = name => dispatch => {
  dispatch(setName(name));
  dispatch(fetchPhotoFromAPI(name));
};
```

/src/components/ButtonGroup.js

```
onChangeName = e => {
  let query = e.target.dataset.name;
  store.dispatch(sayHello(query));
};
```

Use cases for Redux Middleware

- Logging
- Crash reporting
- Routing
- Handling asynchronous requests

You are not using Redux middleware enough

- Also possibly one of the most underused features of Redux.
- Middlewares add a nice encapsulation for store behaviour that does not form part of the data, and can also make testing a lot easier.

Example 1: Encapsulating Your API

Example 2: localStorage and Cookies

Advanced example 1: Encapsulating Your API

```
const fetch = (url, params) => ({
 type: 'FETCH',
 url,
 params,
});

const fetchMiddleware = fetchImplementation => store => next => action => {
 if (action.type === 'FETCH') {
   const { url, params } = action;
   const token = store.getState().token;
   _.set(params, 'headers.token', token);
   return fetchImplementation(url, params);
 } else {
   return next(action);
 }
};

const middleware = applyMiddleware(fetchMiddleware(window.fetch));
const store = createStore(reducers, middleware);

// Example action
const getUser = id => async ({ dispatch }) => {
 const result = await dispatch(fetch(`http://api.website.com/${id}`, { method: 'GET' }));
 ...
};
```

Advanced example 2: LocalStorage and Cookies

```
const middleware = () => store => next => action => {
    // Get the state before and after the action was performed
    const previousToken = store.getState().token;
    next(action);
    const nextToken = store.getState().token;
    // Respond to changes
    if (nextToken !== previousToken) localStorage.setItem('token', nextToken);
};

// Get initial state from localStorage
const token = localStorage.getItem('token');
const initialState = token ? _.set(defaultState, 'token', token) : defaultState;
const middlewares = applyMiddleware(middleware());
const store = createStore(reducers, initialState, middlewares);
```

Questions ?

Resources

- You aren't using Redux Middleware enough
- https://redux.js.org/advanced/middleware

UDACITY

Thank you! Stay Udacious!