upGrad

#RahoAmbitious

Full Stack Software Development

Course: Advanced Frontend Development Using React

Lecture On: Redux

upGrad

In the last class, we discussed...

- REST APIs are a way of communicating data between the client and server. They are stateless and support CRUD operations based on HTTP methods like GET, POST, PUT, PATCH and DELETE

- Common ways to handle authentication are session cookies and auth tokens

- OAuth 2.0 is a widely accepted protocol for handling API authentication

- You can use XMLHttpRequest, a library like axios or the JavaScript fetch API to send/receive data asynchronously from the client web app

Poll 1

What will be the status code of a GET REST API response when the entity being queried does not exist?

A.   200

B.   301

C.   400

D.   404

Poll 1 (Answer)

What will be the status code of a GET REST API response when the entity being queried does not exist?

A.   200

B.   301

C.   400

D.   **404**

Poll 2

Which of the following REST API methods is read only?

A.   GET

B.   POST

C.   PUT

D.   DELETE

Poll 2 (Answer)

Which of the following REST API methods is read only?

A.   GET

B.   POST

C.   PUT

D.   DELETE

upGrad

Today's Agenda

1. Introduction to Redux
   - The Problems with React's State Management
   - The Principles of Redux
2. Redux Flow
   - Redux Pattern
   - Redux Terminology
3. Using Redux with React

Introduction to Redux

The Problems with React's State Management

Each component maintaining its own state leads to:

- Issues with effective cross communication
- Reduced predictability
- Increased complexity

Each piece of code belonging to a component will need to work with the changes that all other pieces of code belonging to other components in the system are trying to make to the state

Introduction to Redux

Redux to the Rescue

- State management is one of the crucial tasks that needs to be handled in ReactJS

- When the application grows and various components are introduced in our application, it becomes harder to maintain the state consistency

- So, to make state management easier, Redux is introduced

- In Redux, using the State container along with the reducer makes the state management quite convenient

Redux to the Rescue

Redux ensures:

- The entire application state is wrapped in a store that handles all updates and notifies all code that subscribes to the store of updates to the state. No more prop drilling as any component can now directly subscribe to the central store

- All changes are made sequentially to ensure a predictable end result free from unexpected effects and race conditions

- The state is immutable, meaning that every change in state results in a totally new version of the state, leading to more predictable code

upGrad

Features of Redux

- **Predictable**: consistent behaviour and easy to test

- **Centralised application state and logic**: allows undo/redo actions and ability to persist state

- **Debuggable**: Redux's architecture allows to log changes, use time-travel debugging, and send complete error reports to a server

- **Flexible**: works with any UI layer, runs in different environments: client, server and native

The Principles of Redux

- **Single source of truth**
    the entire application state is stored centrally

- **State is read-only**
    immutability leads to better debuggability

- **Changes are done with pure functions**
    no side effects, meaning more predictable application flow

Poll 3

Which of these are the benefits of managing state using Redux?
(**Note:** More than one option may be correct.)

A. Distributed application state

B. Sequential state updates

C. Allows time travel debugging by undoing actions and going back in time

D. The Redux library is lightly coupled with React and hence best suited for state management in React

Poll 3 (Answer)

Which of these are the benefits of managing state using Redux?
(**Note:** More than one option may be correct.)

A. Distributed application state
B. **Sequential state updates**
C. **Allows time travel debugging by undoing actions and going back in time**
D. The Redux library is lightly coupled with React and hence best suited for state management in React

Poll 4

Which of the following are true for the store in Redux?
(**Note:** More than one option may be correct.)

A.   It is an object tree

B.   It is mutable

C.   It is immutable

D.   None of the above

Poll 4 (Answer)

Which of the following are true for the store in Redux?
(**Note:** More than one option may be correct.)

A.   **It is an object tree**

B.   It is mutable

C.   **It is immutable**

D.   None of the above

Redux Flow

upGrad
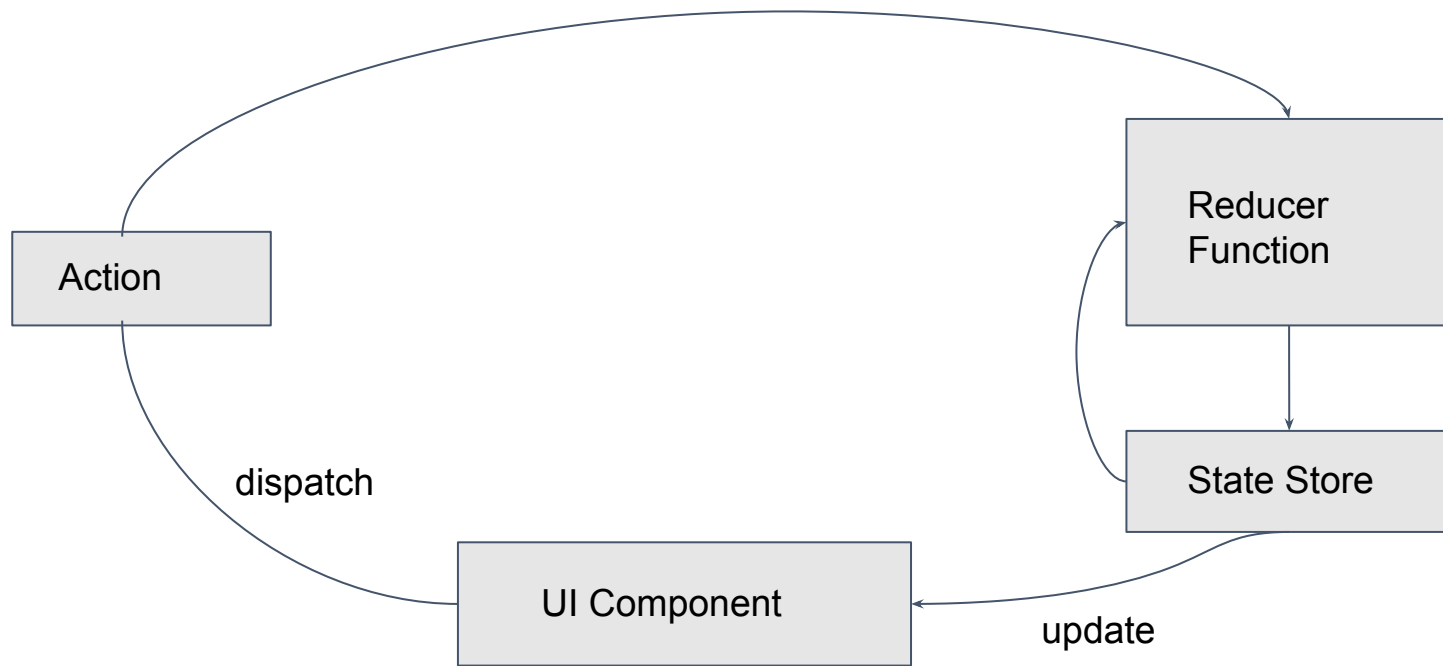
Action

- Only source of information from a view to change application state
- Represented as a plain JavaScript object
- Consists of a mandatory **type** and an optional **payload**

Redux Terminology

Reducer

- Function to produce next state from the previous state and an action
- Pure function, i.e., deterministic without any side effects
- There can only be a single root reducer that is generally composed of other individual reducers

Redux Terminology

Store

- The central application state can be composed of multiple logical 'slices' or namespaces

- The state is immutable, i.e., any change to the state actually produces an entirely new object rather than update the older state object itself

- Components can subscribe to the store directly and be notified of any changes to re-render

Poll 5

Which of the following is/are true regarding the Redux state?
(**Note:** More than one option may be correct.)

A. The Redux state is mutable

B. The Redux state is a single object

C. The Redux state transitions create a new state object that is a clone of the previous state with updates applied

D. The Redux state is composed of multiple individual state objects

Poll 5 (Answer)

Which of the following is/are true regarding the Redux state?
(**Note:** More than one option may be correct.)

A. The Redux state is mutable

B. The Redux state is a single object

C. The Redux state transitions create a new state object that is a clone of the previous state with updates applied

D. The Redux state is composed of multiple individual state objects

Poll 6

Which of the following options is a mandatory attribute of Redux actions?

A.   payload

B.   id

C.   type

D.   dispatch

Poll 6 (Answer)

Which of the following options is a mandatory attribute of Redux actions?

A. payload

B. id

C. **type**

D. dispatch

Using Redux with React

Installing Redux

- Redux is a pure JavaScript library
- To use with React, you need to use the react-redux library (official React binding for Redux)

```
npm install redux
npm install react-redux
```

upGrad

Example: Phone Directory

An app to view and add contact details (name and phone number)

1. Let's define the initial state:

```
const intialState = {
  contacts: []
}
```

Example: Phone Directory

2. Let's define the actions:

```
const actionAddContact = {
    type: 'ADD_CONTACT',
    payload: {
        name: '',
        phoneNumber: ''
    }
}
```

31

Example: Phone Directory

3. Let's define the reducer:

```
function contactReducer(state = initialState, action) {
    switch (action.type) {
      case 'ADD_CONTACT':
        return {
          ...state,
          contacts: [ ...state.contacts, action.payload ]
        };
      default:
        return state;
    }
}
```

Example: Phone Directory

4. Create the store and plug it into your app:

```javascript
import { Provider } from 'react-redux';
import { createStore } from 'redux';

const store = createStore(contactReducer);
function App() {
  return (
    <Provider store={store}>
      <PhoneBookContainer />
    </Provider>
  );
}
```

The <Provider /> makes the Redux store available to any nested components wrapped in the connect() function

5. Subscribe your container component (PhoneBookContainer) with the store

```javascript
import { connect } from 'react-redux';
import { createStore } from 'redux';

function PhoneBookContainer(props) {
  const showNewContactForm = () => { ... }
  const addContact = (contactDetails) => props.dispatch({
    type: 'ADD_CONTACT', payload: contactDetails
  });
  return (
    <div>
      <PhoneBook contacts={props.contacts} onSubmitContactForm={onAddContact}
        onClickAddContact={showNewContactForm} />
    </div>
  );
}
```

Example: Phone Directory

```
const mapStateToProps = (state) => ({
    contacts: state.contacts
  });

const PhoneBookContainerWithRedux =
connect(mapStateToProps)(PhoneBookContainer);
```

Here, you have assumed that you have a presentation component called PhoneBook that takes care of the rendering of the appropriate views and interactivity. You have to pass in all the necessary callbacks and props to this component from your container

upGrad

connect

The **connect** function provides the subscription bridge between the store and the component. Any changes to the store will cause the connected component to re-render

The connect function of react-redux library follows the observer design pattern, since the consumer (component) in this case is directly aware of the producer (store), creating a coupling

It is a general pattern to apply the connect function on container components and not presentational components

connect Parameters

The **connect** function accepts the following parameters:

- **mapStateToProps**
  A function to map a nested field within the store to a prop within the connected component

- **mapDispatchToProps**
  A function to define action dispatcher functions to be available as props within the connected component

mapStateToProps

Let's look at the previous example for mapStateToProps:

```
const mapStateToProps = (state) => ({
    contacts: state.contacts
});
```

This essentially specifies that a prop called 'contacts' be injected into the PhoneBookContainer component, which will contain the value of the contacts field within the central store state object. The mapStateToProps function returns an object containing a map of the props and their values and is invoked every time the store state changes

mapStateToProps

mapStateToProps can also use a second optional parameter containing a map of all the props and their values defined within the component itself. In this case, mapStateToProps would be invoked either when the store state changes or the component's own props change

For example, let's consider a component that only needs to render the currently selected contact, specified by a prop named 'id'

```
const mapStateToProps = (state, ownProps) => ({
    contact: state.contacts[ownProps.id]
});
```

mapDispatchToProps

Let's look at the previous example on using mapDispatchToProps:

You have defined this function in which you are calling **dispatch**, which is part of the container's props

```
const addContact = (contactDetails) => props.dispatch({
    type: 'ADD_CONTACT', payload: contactDetails
});
```

But who added this prop?

mapDispatchToProps

Basically, if you do not define a mapDispatchToProps in the call to **connect**, it injects a default prop called **dispatch** into the connected component. Now, let's say the component needs to dispatch a number of events; you can define separate dispatcher functions for these and inject them as props for added convenience:

```
const mapDispatchToProps = dispatch => ({
    addContact: dispatch({ type: 'ADD_CONTACT', payload: contactDetails })
})
```

Now, you can simply use **props.addContact** in our component

mapDispatchToProps

Like mapStateToProps, mapDispatchToProps can also take an optional second parameter containing a map of the component's own props and values. For example:

```
const mapDispatchToProps = (dispatch, ownProps) => ({
    editContact: (contactDetails) => dispatch({
      type: 'EDIT_CONTACT', payload: { ...contactDetails, id:
ownProps.id }
    })
})
```

mapDispatchToProps returns an object containing a map of the dispatcher functions to be injected as props to the connected component

Batching Multiple Dispatch Actions

One drawback of dispatch is that each time dispatch is called, the component must be re-rendered as this changes the state. Now, consider a scenario where a single user interaction might dispatch multiple actions, e.g., a button click might add a new contact as well as update the view in some way at the same time. You can batch these two dispatch calls into a single one, thereby causing a re-render to happen only once and not twice

```
import { batch } from 'react-redux';

batchDispatch = () => {
  batch(() => {
    props.dispatch(...);
    props.dispatch(...);
  });
};
```

Phone Directory

In the Phone Directory app, you could move your entire subscribers list you were maintaining in the local state of the component to a Redux data store:

```
import {useDispatch} from "react-redux";

const dispatch = useDispatch();
```

And, in the loadData function, you can do the following:

```
dispatch({"type":"SET_SUBSCRIBERS", payload: data});
```

You have to update our event handlers to dispatch the corresponding actions instead now to update the Redux store

Code Reference

Poll 7

Which of the following functions is used to subscribe a React component to the store state?

A. dispatch

B. mapStateToProps

C. subscribe

D. connect

Poll 7 (Answer)

Which of the following functions is used to subscribe a React component to the store state?

A. dispatch
B. mapStateToProps
C. subscribe
D. **connect**

Poll 8

What kind of components do you typically 'connect' to the Redux store?

A. Pure components

B. Uncontrolled components

C. Presentational components

D. Container components

Poll 8 (Answer)

What kind of components do you typically 'connect' to the Redux store?

A. Pure components

B. Uncontrolled components

C. Presentational components

D. Container components

Poll 9

Which prop gets injected into a React component, that is connected to the Redux store, if you do not provide an argument for the mapDispatchToProps function in the call to connect?
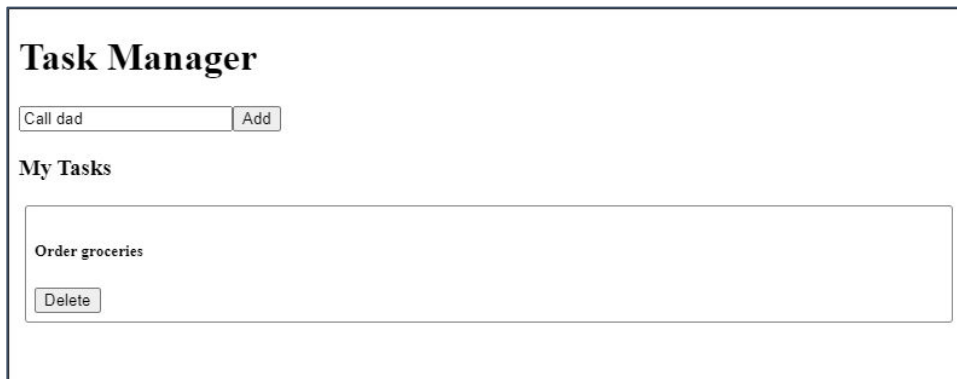
A.   dispatch

B.   reducer

C.   batch

D.   store

Poll 9 (Answer)

Which prop gets injected into a React component, that is connected to the Redux store, if you do not provide an argument for the mapDispatchToProps function in the call to connect?

A.   dispatch

B.   reducer

C.   batch

D.   store

Build a React app for maintaining a list of tasks. The app should allow adding a new task to a list and deleting a task from the list. Here's a screenshot of how the basic app should look like:

The stub code is provided here
The solution code is provided here

All the code used in today's session can be found in the

link provided below:

https://github.com/upgrad-edu/react-hooks/tree/Session12

Doubt Clearance (5 minutes)

Important Questions

Important Questions

1. What is Redux? What are its components?

2. What is the Flux? How is it different from Redux?

3. What are the core principles of Redux?

4. What is the difference between mapStateToProps() and mapDispatchToProps()?

5. How to dispatch an action on load?What is the significance of the Store in Redux?

Key Takeaways

- Redux makes state management centralised, more predictable, debuggable and flexible

- In the Redux architecture, views subscribe to data from the central store, and the only way to change state is to dispatch actions that go through reducer functions to generate a new version of the changed state

- The connect function from the react-redux library is used to connect the view to the store, which re-renders each time the state updates

The following task is to be completed after today's session:

MCQs

Full Stack Software Development

In the next class, we will discuss…

1. Testing Frontend Code
2. Testing React Apps
3. Jest

upGrad

#RahoAmbitious

Thank You!