Time travelling with React and Redux

Miguel Molina - **@migrrrr**

What is Redux?

Seriously, what is Redux?

Redux is a predictable state container for JavaScript apps.

Why Redux and not Flux, Reflux, ...?

- Minimal API
- Predictable behaviour
- Easy to test
- Not coupled to any concrete framework

Redux three principles

- **Single source of truth** (all the state in one tree object inside a single store).
- **State is read-only**.
- **Mutations are written as pure functions.**

Redux architecture

Three kinds of components

Actions

- Describe that something happened
- Only way of mutating the state

Reducers

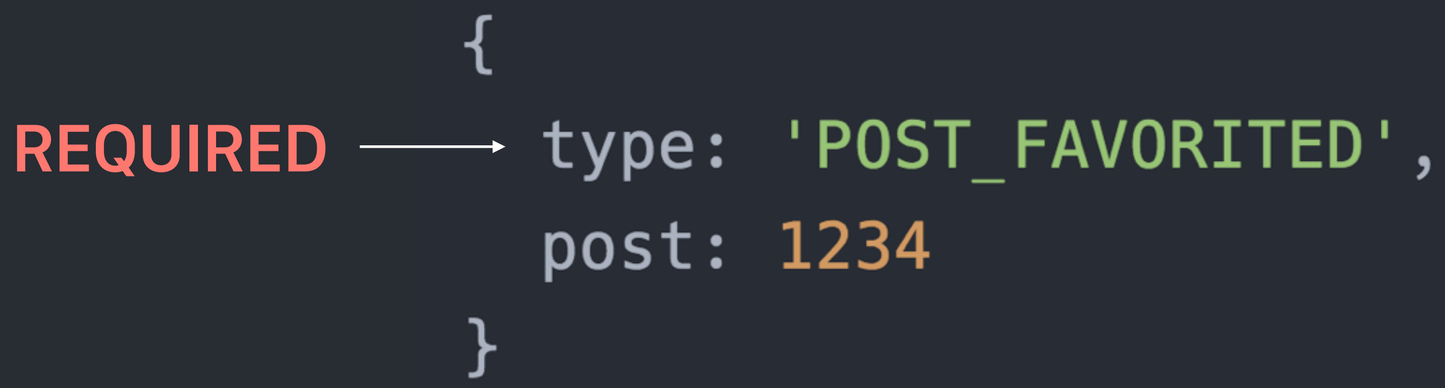
- Functions that define how the state changes after an action
- Can be composed

Store

- Holds state
- Allows access to state
- Allows state to be updated
- Registers listeners
- Just **one** per app

Actions

An **action is just a plain JavaScript object** describing that something happened.



The rest is up to you.

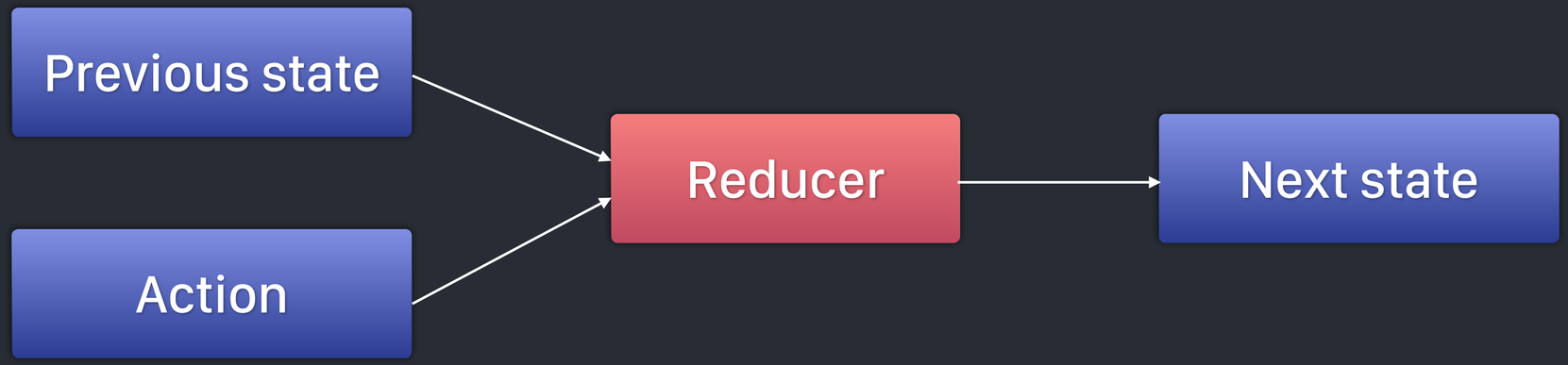
Action creators

To reduce boilerplate we can use **action creators**.
They are just **pure functions** that create actions.

```
function addTodo(text) {
  return {
    type: 'ADD_TODO',
    text,
    added: new Time()
  }
}
```

Reducers

A reducer is a **pure function** that receives a state and an action and **returns the resulting state**.



Reducers

```
let initialState = [];
let id = 0;

function todoReducer(state = initialState, action) {
  switch (action.type) {
  case 'ADD_TODO':
    let { text, added } = action;
    return [...state, { id: id++, text, added }];
  case 'COMPLETE_TODO':
    return [
      ...state.slice(0, action.index),
      ...state.slice(action.index + 1)
    ];
  }
}
```

Combining reducers

We can **combine multiple reducers by composing them** using the `combineReducers` function from redux.

```
import { combineReducers } from 'redux';

const combinedReducer = combineReducers({
  foo: fooReducer,
  bar: barReducer,
  baz: bazReducer
});
```

Combining reducers

Each reducer manages its own part of the global state.

Managed by `fooReducer` ⟶ `foo: /* stuff */,`

Managed by `barReducer` ⟶ `bar: /* stuff */,`

Managed by `bazReducer` ⟶ `baz: /* stuff */`

```
{
}
```

Store

The store **holds the application state**. When it receives a new action passes the current state and the action to the reducers to get the new state.

We can **create the store** with the `createStore` function.

```
import { createStore } from 'redux';
import { fooReducer } from './reducers';

let store = createStore(fooReducer);
```

Store

The store has **3 important methods**.

- `dispatch`: dispatch an action
- `getState`: return the current state
- `subscribe`: subscribe to state updates

```
let unsubscribe = store.subscribe(() => {
  console.log(store.getState());
});

store.dispatch({type: 'ADD_TODO', text: 'New todo'});
```

That's it

Really, not kidding. That's pretty much all the API you need to know.

You can find out more here: http://rackt.github.io/redux/

How to integrate it with React?

- You can integrate Redux in your app using `subscribe` and `dispatch` yourself because it is not coupled to any framework.
- The easier way to do it is using `react-redux`, a package that already does all that.

react-redux API

The root component of your app must be wrapped in a `Provider` component (that is not completely true but let's say it is), which will pass the store down your hierarchy.

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

React.render(
  <Provider store={store}>
    {() => <MyRootElement />}
  </Provider>,
  document.body
);
```

react-redux API

You can connect components to the store using the `connect` function. The function passed to connect will determine to which parts of the state we want to subscribe.

```
import { connect } from 'react-redux';
import { Foo } from './components';

export default connect(state => state)(Foo);
```

You can view the full `connect` API here: https://github.com/rackt/react-redux/blob/master/docs/api.md

react-redux API

Let’s say we have a state that looks like this


```
{
  foo: { /* stuff */ },
  bar: { /* stuff */ },
}
```

We can subscribe to just one part by returning just the properties we want


```
export default connect(state => ({
  foo: state.foo
}))(Foo);
```

Smart and dumb components

With the previous API in mind we have **two types of components**

Smart components

↓

Are **subscribed** to the state and can dispatch actions. They will pass down the needed props to their children.

Dumb components

↓

They will **not** be **subscribed** to the state. Rendered by their parents.

Smart and dumb components

For example, imagine a catalog of products.

Catalog component (smart)

Is subscribed to the state to get the products.
Renders the products and passes to them the needed data.

Product component (dumb)

It is rendered by the catalog and receives the product to render from it.

Smart and dumb components

```
const Catalog = React.createClass({
  render() {
    let products = this.props.products.map(p => {
      return <Product product={p} />;
    });
    return (
      <div className='catalog'>
        {products}
      </div>
    );
  }
});
export default connect(state => ({
  products: state.products
}))(Catalog);
```

```
const Product = React.createClass({
  render() {
    return (
      <div className='product'>
        <img src={this.props.product.picture} />
        <h3>{this.props.product.name}</h3>
        <span className='product__price'>
          {this.props.product.price}
        </span>
      </div>
    );
  }
});
export default Product;
```

What about time travelling?

- As we have seen before, Redux maintains a state that is only mutated with actions. **The same state and the same action result in the same resulting state.**
- That allows the application to be time-travelled, because we can get to a point by **applying to the state a series of actions**.

Redux Dev Tools

2 things need to be done to integrate Redux Dev Tools.

- Enhance the store using the provided middleware.
- Render the DebugPanel in our app.

Redux Dev Tools

Enhance the store using the provided middleware.

```
import { compose, createStore } from 'redux';
import { devTools } from 'redux-devtools';
import { myCombinedReducers } from './reducers';

const finalCreateStore = compose(devTools())(createStore);
export default finalCreateStore(myCombinedReducers);
```

Redux Dev Tools

Render the DebugPanel in our app.

```
import { DevTools, DebugPanel, LogMonitor } from 'redux-devtools/lib/react';
React.render(
  (<div>
    <Provider store={store}>
      {() => <MyRootComponent />}
    </Provider>
    <DebugPanel right top bottom>
      <DevTools store={store} monitor={LogMonitor} />
    </DebugPanel>
  </div>),
  document.body
);
```

Example time!

- We will built a very simple clone of the twitter timeline using all we have learnt before. It will allow us to post tweets and then fav and retweet them.
- We will also demonstrate how Redux Dev Tools work.

The source code is available at https://github.com/mvader/reactmad-redux-example

Go build awesome apps!

Embrace the power of redux!