Read Me · Redux

> Jing Chen, creator of Flux
>
> Bill Fisher, author of Flux documentation
>
> André Staltz, creator of Cycle

Developer Experience

I wrote Redux while working on my React Europe talk called "Hot Reloading with Time Travel". My goal was to create a state management library with minimal API but completely predictable behavior, so it is possible to implement logging, hot reloading, time travel, universal apps, record and replay, without any buy-in from the developer.

Influences

Redux evolves the ideas of Flux, but avoids its complexity by taking cues from Elm.

Whether you have used them or not, Redux only takes a few minutes to get started with.

Installation

To install the stable version:

```
npm install --save redux
```

This assumes you are using npm as your package manager.

If you don't, you can access these files on unpkg, download them, or point your package manager to them.

Most commonly people consume Redux as a collection of CommonJS modules. These modules are what you get when you import `redux` in a Webpack, Browserify, or a Node environment. If you like to live on the edge and use Rollup, we support that as well.

If you don't use a module bundler, it's also fine. The `redux` npm package includes precompiled production and development UMD builds in the `dist` folder. They can be used directly without a bundler and are thus compatible with many popular JavaScript module loaders and environments. For example, you can drop a UMD build as a `<script>` tag on the page, or tell Bower to install it. The UMD builds make Redux available as a `window.Redux` global variable.

The Redux source code is written in ES2015 but we precompile both CommonJS and UMD builds to ES5 so they work in any modern browser. You don't need to use Babel or a module bundler to get started with Redux.

Complementary Packages

Most likely, you'll also need the React bindings and the developer tools.

```
npm install --save react-redux
npm install --save-dev redux-devtools
```

Note that unlike Redux itself, many packages in the Redux ecosystem don't provide UMD builds, so we recommend using CommonJS module bundlers like Webpack and Browserify for the most comfortable development experience.

The Gist

The whole state of your app is stored in an object tree inside a single *store*.

The only way to change the state tree is to emit an *action*, an object describing what happened.

To specify how the actions transform the state tree, you write pure *reducers*.

That's it!

```
import { createStore } from 'redux'

/**
 * This is a reducer, a pure function with (state, action) => state signature.
 * It describes how an action transforms the state into the next state.
 *
 * The shape of the state is up to you: it can be a primitive, an array, an object,
 * or even an Immutable.js data structure. The only important part is that you
should
 * not mutate the state object, but return a new object if the state changes.
 *
 * In this example, we use a `switch` statement and strings, but you can use a
helper that
 * follows a different convention (such as function maps) if it makes sense for
your
 * project.
 */
```

```
function counter(state = 0, action) {
  switch (action.type) {
  case 'INCREMENT':
    return state + 1
  case 'DECREMENT':
    return state - 1
  default:
    return state
  }
}

// Create a Redux store holding the state of your app.
// Its API is { subscribe, dispatch, getState }.
let store = createStore(counter)

// You can use subscribe() to update the UI in response to state changes.
// Normally you'd use a view binding library (e.g. React Redux) rather than
subscribe() directly.
// However it can also be handy to persist the current state in the localStorage.

store.subscribe(() =>
  console.log(store.getState())
)

// The only way to mutate the internal state is to dispatch an action.
// The actions can be serialized, logged or stored and later replayed.
store.dispatch({ type: 'INCREMENT' })
// 1
store.dispatch({ type: 'INCREMENT' })
// 2
store.dispatch({ type: 'DECREMENT' })
// 1
```

Instead of mutating the state directly, you specify the mutations you want to happen with plain objects called *actions*. Then you write a special function called a *reducer* to decide how every action transforms the entire application's state.

If you're coming from Flux, there is a single important difference you need to understand. Redux doesn't have a Dispatcher or support many stores. Instead, there is just a single store with a single root reducing function. As your app grows, instead of adding stores, you split the root reducer into smaller reducers independently operating on the different parts of the state tree. This is exactly like there is just one root

component in a React app, but it is composed out of many small components.

This architecture might seem like an overkill for a counter app, but the beauty of this pattern is how well it scales to large and complex apps. It also enables very powerful developer tools, because it is possible to trace every mutation to the action that caused it. You can record user sessions and reproduce them just by replaying every action.

Learn Redux from Its Creator

Getting Started with Redux is a video course consisting of 30 videos narrated by Dan Abramov, author of Redux. It is designed to complement the “Basics” part of the docs while bringing additional insights about immutability, testing, Redux best practices, and using Redux with React. **This course is free and will always be.**

Thanks

- The Elm Architecture for a great intro to modeling state updates with reducers;
- Turning the database inside-out for blowing my mind;
- Developing ClojureScript with Figwheel for convincing me that re-evaluation should “just work”;
- Webpack for Hot Module Replacement;
- Flummox for teaching me to approach Flux without boilerplate or singletons;
- disto for a proof of concept of hot reloadable Stores;
- NuclearJS for proving this architecture can be performant;
- Om for popularizing the idea of a single state atom;
- Cycle for showing how often a function is the best tool;
- React for the pragmatic innovation.

Special thanks to Jamie Paton for handing over the `redux` NPM package name.

Logo

You can find the official logo on GitHub.

Change Log

This project adheres to Semantic Versioning.

Every release, along with the migration instructions, is documented on the Github Releases page.

Patrons

The work on Redux was funded by the community.

Meet some of the outstanding companies that made it possible: