Redux, ngRedux, & Sagas

A Gentle Introduction for Angular Developers

Jesse Warden | NationJS | September 16th, 2016

Who

- Jesse Warden
- JavaScript, Little Bit of Node

What We'll Cover

- What & Why of Redux
- ngRedux
- Sagas

What & Why of Redux

- What: Predictable State Container
- Why: helps create consistent & predictable applications, easier to test

Er… wat

- Er, What: who’s changing my data, when, and where in a predictable fashion

Er… Why?

- Er, Why: more hints as to where the defect are… and code you write is easier to unit test

Predictable How

- comes from Functional Programming
- pure functions / avoiding see effects
- examples of pure functions

State Waaaat

- "Who is the logged in user right now?"

State

- the data your front-end application shows and edits
- Where you put things
- Often called mutable state

Examples of State

- window state
- Angular Factory state

Examples of Changing State

- set variable
- have a function do it
- Object.assign

Ok, we get predictable state

- predictable: pure functions
- state: my data in t3h RAM

The Bubble Problem

- When you start refactoring imperative code to use pure functions, you run into the bubble problem: mutable state bubbles up and out state.
- Someone, SOMEWHERE, has to eventually store the state by using a var vs. const.
- If you abstract it into a safe container, you've created Redux.

The Bubble Problem

- Imagine you can’t ever use var, only const

Refactor Imperative

Redux

- Data for entire app in a single object. You only change it by dispatching actions with your new value. To actually change the data, you use pure functions.

- Data for entire app spread out over multiple classes. You change through method calls. To change data, you'd use getter/setters, or $watchers.

Initial State

- data right now
- our data model
- starts as a basic domain
- eventually tree gets pretty big and specialized
- show default Object

Actions

- what happened / what do you want to change?
- show basic action
- show different types
- show WHY action creators (pure functions)

Reducers

- change your data in response to what happened
- pure as possible
- like Array.reduce
- _.reduce
- talk about initial state again
- both in switch default
- and in ES6 default
- combineReducers shrinks size, not a requirement

Store

- holds your state. There is only 1.
- you access it through getState
- update state via dispatch(action)
- for views/GUI, listen via subscribe(listener)
- can set default state via 2nd param of createStore

Data Flow

- store.dispatch(action)
- reducer handles change request
- new state tree, saves it
- new state of your app via store.subscribe(listener)

Async

- show the 3 states
- state machines
- Thunks using Promises
- Sagas

Sagas

- handling async through pure generator functions

ngRedux

- $ngReduxProvider (setup)
- $ngRedux (connect)
- $onDestroy (unsubscribe)
- mapStateToThis
- logger

File Organization

- Node module to ES6 Module
- sock drawer vs. features
- reducer & saga per feature

Conclusions

- Redux gives you a 2kb functional programming framework that ensures your data is kept as pure **as possible**.
- clear flow of data (action > reducer > store > subscribe)
- single data store, scale to multiple functions & class files

Resources

1. Eric Elliot on What a Pure Function Is https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-pure-function-d1c076bec976
2. Learn Array Comprehensions http://reactivex.io/learnrx/
3. Jesse Warden's Beginner's Guide to Functional Programming http://jessewarden.com/2016/08/beginners-guide-to-functional-programming-part-1.html
4. Lodash https://lodash.com/docs
5. Dan Abramov teaches Redux on egghead.io https://egghead.io/lessons/javascript-redux-the-single-immutable-state-tree
6. Redux Documentation http://redux.js.org/docs/api/
7. Redux Saga Documentation http://yelouafi.github.io/redux-saga/

Questions?

- Jesse Warden
- jesse@jessewarden.com
- @jesterxl on Twitter