

Outline

Intro

- ▶ Kia ora koutou . I'm Brooke Mitchell and this is my talk, I've got the keys! Using lenses and transducers simplify your life.
- ▶ Thank you to jenofdoom, the conference organizers, volunteers, and other speakers. I'm sad I'm missing anna's talk, her elm workshops are part of what got me in to functional programming. Thanks for having me for my first ever talk.
- ▶ I've a developer who has been using JavaScript in earnest for maybe three years.
- ▶ These are my experiences trying to simplify data manipulation in a redux app I built.

Welcome

- ▶ This talk is pretty code example heavy but try not to get too caught up in following every detail about what it happening.
- ▶ Since long code examples make me sleepy. There are helpful talk themed images breaking them up. Try to focus on those.

Slides and source code

- ▶ The focus of this talk is on trying functional techniques to manage objects and evaluating their usefulness.
- ▶ If you get interested you can try the examples from the slides in a ramda repl later.
- ▶ These are the github projects for the talk and the example project.

Me and Redux

- ▶ So we'll have to cover a little bit of ground understanding redux before we start. But we are about to learn is redux is about managing one large object called state.
- ▶ So this talk should hopefully also teach you about how to deal with pojos (plain old JavaScript objects) (A.K.A dictionaries, hash-maps) something we all use.
- ▶ Since this talk is about my experiences manipulating complex objects, we'd better talk is about how we access properties in objects, through keys. When we talk about keys, I think there is something slightly magical and metaphorical about them. For that reason I have been thinking about keys a lot, and so it made sense to include someone that talks about keys a lot, the one and only, DJ Khaled.

DJ Khaled

- ▶ play jay-z - the keys clip. . .
- ▶ For those unfamiliar with the work of DJ Khaled, you could briefly describe him as a music mogul and internet celebrity. While Khaled is an important music producer, snapchat celebrity and motivational speaker, most importantly for this talk, he has a life philosophy based around 'keys' to success. So we'll be talking about keys in both senses, managing objects and lifelong learnings.

Ramda

- ▶ I'm going to use the Ramda library and es2015 style functions in the examples in this talk. Just know that every any library should have these tools.
- ▶ I know we should be wary of tool dependence here, but ramda is a useful grab bag of time honoured functional programming tricks that it is handy to know. So I feel any time you spend learning it will be useful in other libraries and languages. Many other libraries do the same thing and they should be able to be used interchangeably.
- ▶ These are techniques you're going to see a lot in this talk. Most of the code you'll only need to the gist of, but if you want to pay attention now would be a good time.
- ▶ So the first amazingly useful functional feature we'll using is:
- ▶ Auto-currying. Comes by default in ramda. This means you can provide one argument at a time to get a back partially applied function.

Ramda

- ▶ Function composition. Perhaps the biggest major key. Take a bunch of nest functions and you can lay them out in the same order as you see them nested. Now you can really go to town stacking them up. They build up from the bottom just like the nest function first call is from the inside. You can also use pipe if you prefer thinking forwards.
- ▶ I'll also be using the point free style sometimes, despite appearances since partial application returns another function you can skip declaring arguments.


- ▶ And applicatives.

Perhaps the one people are least likely to have seen before. There are a number of ways to use them but for the examples in this talk I use them a bit like a reverse map, I take a list of functions and map an argument to that list and get a list of results, or partially applied functions if they don't have all arguments back. There are other many other ways to use their powers but that's all we require here.

Redux

- ▶ Anyway back to the app I was building. Sadly it was built as an internal tool for a closed source environment. But here is a smaller toy example that also shows some of the problems that as redux apps get larger.

...Show app.

- ▶ The problem with this app is because the state tree is large, editing values gets slow. If you run this example on the master branch chrome warns you the response time for ui is > 300 ms.
 - ▶ Two things you could do to fix that are use immutable data structures like immutable.js or register many subscriptions to state changes much further down the component tree. Instead of doing that lets just refactor and see what falls out.
- ▶ A good life philosophy I think, see what you can ignore.
- ▶ Before I show some of the refactorings I think are useful for dealing with objects, I'll try and offer as quick an overview of redux as possible. Please don't be mad if you are a redux expert.
- ▶ Here is the redux app flow from 10,000 feet: 

Redux Reducer

- ▶ Coincidentally a 'reducer' in redux is effectively a 'reduce' for a state object, I like to think of it like so...
- ▶ In reality that list of actions is provided by redux's dispatch function and unfolds over time, one action at a time, so its like an observer, but I think its a good way to conceptualize it.

Take special note of the line that looks like this:

```
return Object.assign({}, oldObject, newObject)
```

- ▶ Major key alert: don't mutate your state in a reducer. It will be ignored anyway in the diff comparison.
- ▶ Here's a link properly discussing mutation. Basically it means keys and values of an object are unchangeable. In practice that means we need to return a fresh new copy of the object every time.

Redux Mapper

- ▶ OK now for the mapper.
- ▶ Just a note that you don't need to use react with redux
- ▶ This is a state mapper using nanocomponent.
- ▶ An very cool new component library that works on any framework and implements react fiber for great performance. Its comparable with all the frameworks and x-to-js compilers (even elm) and frees us from writing the same components like infinite list each time a new framework is out. I really like this.
- ▶ Back on topic, my example app uses jsx but next time I'll check nanocomponent out.
- ▶ State mappers are like a `.map` where you pluck the desired items for an object and do any calculations require to get your data view ready.
- ▶ So reducing and mapping an immutable object. That is my summary of redux that should get us there.

Refactoring a real app

- ▶ Here is the examples troubled mapper. I decided to implement some feature creep, user management and routing. Again you dont need to read this, just get a bad feeling that all this logic shouldn't really be in a view.
- ▶ Key alert: use ramda/lodash 'get' instead to avoid throwing errors.

Bad map fix, step 1: create selectors

- ▶ My first step is to create selectors to get this property access out of the view and somewhere else. Usually I just make a selector file and work from there, it helps with testing, and we remove the any logic or intermediary functions from the view.
- ▶ We could go further but good enough I say, at least these are easily composable and testable now. We could take this even further and create an uber selector that combines all the selectors.

Bad map fix, step 2: Composing with ramda

Next step is to look for common property access and compose selectors together. You might find you'll doing one kind of property access in many places. Major key: compose selectors.

- ▶ Wayyy sweeter. tbh this is probably the sweet spot. Go deeper if needs require.

Alternative Step 2: Reselect

The alternative to composing selectors yourself is to use a library like `reselect` to help with the composition. My issue with `reselect` is it re-invents the wheel a bit when you could just take the time to learn composition and not sweat the difference when frameworks change.

- ▶ One free win you get with `reselect` is it memoizes the final function for you. This means that if anytime the result from all the selectors is the same, `createSelector` doesn't bother calculating the state again, if that final function is a big one that's helpful, usually it's not too bad though.

Memoizing, a key to success, sometimes.

Alternative Step 2.5: Ramda Reselect

- ▶ What you could do instead though is write it you self.
- ▶ Ok so this is my version of reselect in what could be one line, and probably good enough for most scenarios.
- ▶ This does the same thing as createSelector, takes the state, runs it through a list of selectors (except the last one) then applies those values to the last function, which has been memoized.
- ▶ Now we dont have to learn another library. There are other capabilities reselect has which I've never used. Like props, you'll notice if you go through my example that I pretty much never use props in components that subscribe to state changes.
- ▶ Thats another key I've found, focus on state for stateful components and just use props with pure components to keep things simple. Things don't always work out that way but I find that really helps me.
- ▶ I have a more fully featured version of ramda-reselect that lets you use props and passes reselects tests. Its up there as a npm library in case you ever want to use it, or hopefully just look at

Alternative Step 2.5: Ramda Reselect

- ▶ So I feel like we've slimmed down our mapper pretty nicely.

Now lets take a look at our reducer.

Awkward reducer

- ▶ Here is the real reducer for the roadworks editing app, this is the function for setting the new shape of the state called every time an 'action' is dispatched. Sorry its so long please allow your eyes to glaze over.
- ▶ Those ellipsis are a proposed es2017 shortcut to spread an object. You can think of those as the same as an object assign. This is where the slowness is.

Reducer refactor pt1

- ▶ Anyway that reducer was obviously too much. The advice you see in the redux docs is to break functions out, and I think you can easily see how to do that.
- ▶ To me breaking out functions feels a little dishonest. It looks nice but to me doesn't actually reduce complexity, now you just look in a different place. You have more loc and a single use function. Maybe inlining is fine and more honest.

Reducer refactor pt2

- ▶ How about trying something else, to actually reduce code.
- ▶ What if we had an abstraction that allows you to target a specific part of a deeply nested object
- ▶ Maybe then we could use our selectors in a reducer.
- ▶ This won't work. But I'm getting a feeling there is an abstraction for focusing on a section of an object that could be very useful.

I'm talking about...

Reducer refactor pt3. - Lenses!

- ▶ Now we can use one abstract for both mappers and reducers.
- ▶ To me this is way cleaner. And get ready for the second win, your lenses act as both getters and setters, so you get selectors for free when you write them. Major key.
- ▶ I can stop thinking in terms of reducers now and just think of writing a selector I'll use later to set the value too.
- ▶ One of the fun things about lenses is they look like they compose left to right. It's a little confusing
- ▶ Damn and its faster. This is pretty nice to look at, although we have to be aware of the tradeoffs.
- ▶ I find working with lenses to be a mental context switch. Depending on the situation it's not always worth it. Composed functions are often good enough.
- ▶ But what if we want to go deeper abstracting our reducer. We could try lenses cousins.

Reducer refactor pt4. - Transducers!

- ▶ Ok the title of this talk promised that there would be transducers as well.
- ▶ I was working replacing reducers with transducers I was finding that they weren't quite right for my use case and I ended up reverting a large chunk of the code base and focusing on composition instead.
- ▶ You can't always win.
- ▶ What reducers offer is another performance boost when you compose functions together. They do this by skipping intermediate memory allocation. It's kind of like putting all your functions in a blender.
- ▶ If you have code that is performing a large number of transformations on data transducers make sense.
- ▶ Transducers do this by generalizing your functions to all look like functions that you would pass to a reducer, composing that function and placing it inside a reducer.
- ▶ This is pretty mind bending to me.
- ▶ Swapping reducer functions for transducers is definitely an

Keys conclusion

- ▶ So thank you for listening to my first ever talk.
- ▶ Some things to take away.
 - ▶ The key is to have every key. Learn a bunch of techniques and compose them.
 - ▶ Don't play yourself. Refactor honestly.
 - ▶ Stay away from them. Don't get discouraged if things don't work out.
- ▶ Here are links to some talks inspired this one.

Outro

- ▶ I wish you the best of luck and may you all keep winning. play outro...