Current Skill Redux thunk

What's redux-thunk?

Thunk is another word for a function but it's not just any old function. It's a special (and uncommon) name for a function that's returned by another.

```
function wrapper_function() {
 // this one is a "thunk" because it defers work for later:
 return function thunk() {
   // it can be named, or anonymous
   console.log("do stuff now");
 };
}
```

You already know this pattern. You just don't call it "thunk." If you want to execute the "do stuff now" part, you have to call it like wrapper_function()() – calling it twice, basically.

So how does this apply to Redux?

Well, by now, we know that Redux has a few main tools: there are "actions", "action creators", "reducers", and "middleware."

Actions are just objects. As far as Redux is concerned, out of the box actions must be plain objects, and they must have a type property. Aside from that, they can contain whatever you want – anything you need to describe the action you want to perform.

Actions look like this:

```
// 1. plain object
// 2. has a type
// 3. whatever else you want
{
 type: "USER_LOGGED_IN",
 userName: "dave"
}
```

Redux thunk:

Since it's kind of annoying to write those objects by hand all the time (not to mention error-prone), Redux has the option of "action creators" to stamp these things out:

```
function userLoggedIn() {
 return {
   type: "USER_LOGGED_IN",
   username: "dave"
 };
}
```

It's the same exact action, but now you can "create" it by calling the userLoggedIn function. This just adds one layer of abstraction.

Instead of writing the action object yourself, you call the function, which returns the object. If you need to dispatch the same action in multiple places around your app, writing action creators will make your job easier.

Redux thunk:

Actually Redux action does nothing. They are nothing but objects. To make them complete a concrete action (Calling an API, trigger another function ...) we have to make sure that the code lives inside a function. This function (also called the thunk) is a bundle of work to be done. Right now our action creator is performing an action. Like it's shown in the example below

```
function getUser() {
 return function() {
   return fetch("/current_user");
 };
}
```

Redux thunk:

If only there was some way to teach Redux how to deal with functions as actions.

Well, this is exactly what redux-thunk does: it is a middleware that looks at every action that passes through the system, and if it's a function, it calls that function. That's all it does.

The only thing I left out of that little code snippet is that Redux will pass two arguments to thunk functions:

Dispatch, so that they can dispatch new actions if they need to;

getState, so they can access the current state.

Here's an example to illustrate it:

```
function logOutUser() {
 return function(dispatch, getState) {
   return axios.post("/logout").then(function() {
     // pretend we declared an action creator
     // called 'userLoggedOut', and now we can dispatch it
     dispatch(userLoggedOut());
   });
 };
}
```

the getState function can be useful for deciding whether to fetch new data or return a cached result, depending on the current state.

Redux thunk:

Set up redux-thunk in your project.

If you have a project that already has Redux set up, adding redux-thunk will be a two-step process.

First, **install the package**.

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

Then, wherever you have your Redux setup code, you need to **import redux-thunk and insert its middleware into Redux:**

```
// You've probably already imported createStore from 'redux'
// You'll need to also import applyMiddleware
import { createStore, applyMiddleware } from "redux";

// Import the `thunk` middleware
import thunk from "redux-thunk";
```

```
// Import your existing root reducer here.
// Change this path to fit your setup!
import rootReducer from "./reducers/index";

// The last argument to createStore is the "store enhancer".
// Here we use applyMiddleware to create that based on
// the thunk middleware.
const store = createStore(rootReducer, applyMiddleware(thunk));
```

Just make sure you wrap thunk in the applyMiddleware call or else it won't work.

After this, you're all set: you can now dispatch functions that do whatever you need them to.

‹ Previous next ›