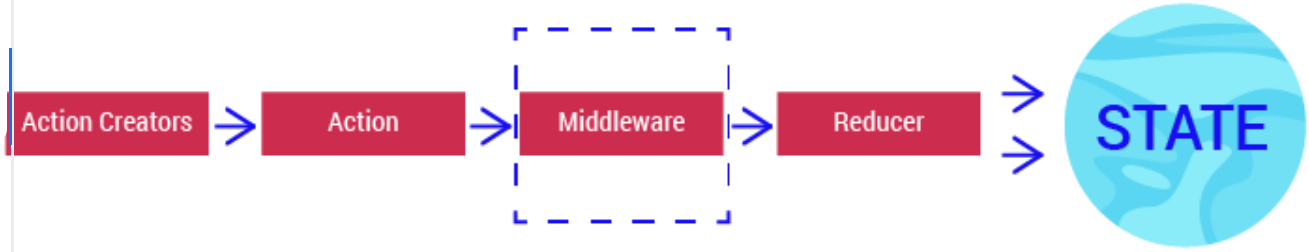
Advantages of Middleware



What is Middleware?

Redux middleware intermediates Action Creators and Reducers. The Middleware intercepts the action object before a Reducer receives it and gives it the functionality to perform additional actions or enhancements with respect to the action dispatched.

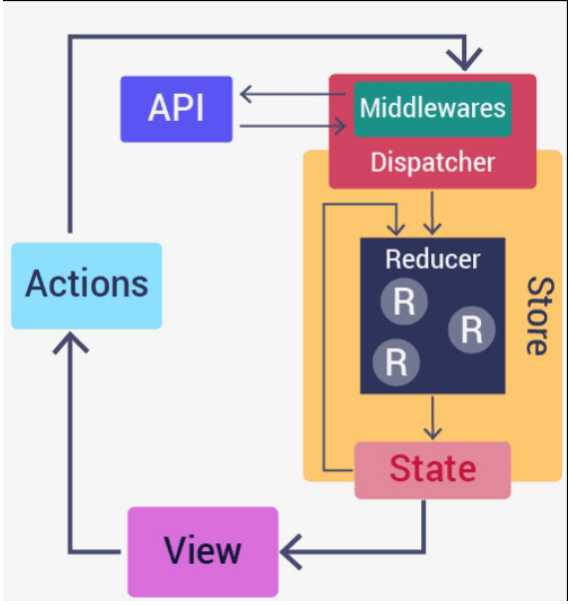
Advantages of Middleware

Why use it?

The action/reducer pattern is very clean for updating the state within an application. But what if we need to communicate with an external API? Or what if we want to log all of the actions that are dispatched? We need a way to run side effects without disrupting our action/reducer flow.

Middleware allows for side effects to be run without blocking state updates.

We can run side effects (like API requests) in response to a specific action or in response to every action that is dispatched (like logging). There can be numerous Middlewares that an action passes through before ending in a reducer.



Advantages of Middleware

The Redux middleware syntax is a mouthful: a middleware function is a function that returns a function and that returns a function.

The first function takes the `store` as a parameter, the second takes a `next` function as a parameter, and the third takes the `action` dispatched as a parameter.

The `store` and `action` parameters are the current Redux store and the action being dispatched.

The real magic is found in the `next()` function.

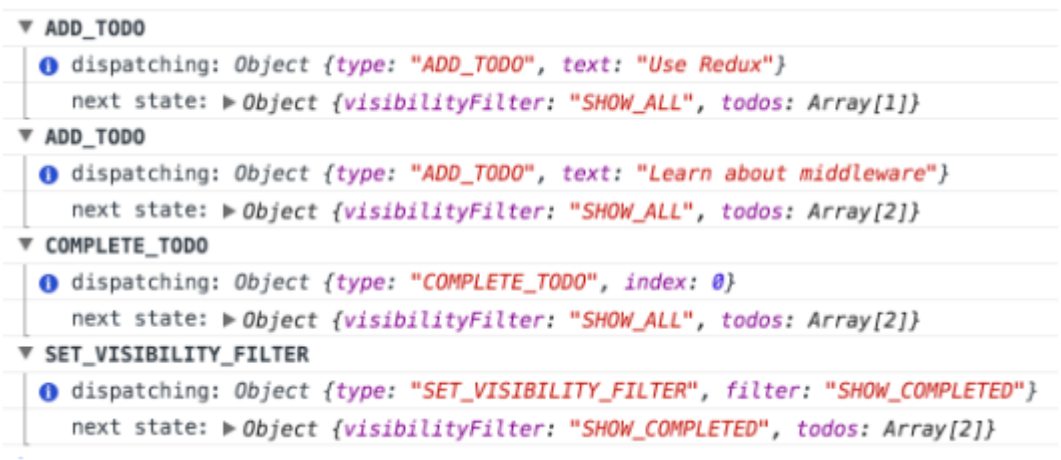
The `next()` function is what you call to say "this middleware is done executing, pass this action to the next middleware". In other words, middleware can be asynchronous.

```
const reduxMiddleware = store => next => action => {

 . . . . . .

next(action);
};
```

Logging action



Logging: one of the benefits of Redux is that it makes state changes predictable and transparent. Every time an action is dispatched, the new state is computed and saved. The state cannot change by itself, it can only change as a consequence of a specific action.

Wouldn't it be nice if we logged every action that happens in the app, together with the state computed after it? When something goes wrong, we can look back at our log and figure out which action corrupted the state.

The logger function:

Middleware is applied in the state initialization stage with the enhancer applyMiddlware()

```
import { createStore, applyMiddleware  } from "redux";
```

Logger is the middleware function that will log action and state.

```
const logger = store => next => action => {
 console.log('dispatching', action)
 let result = next(action)
 console.log('next state', store.getState())
 return result
}
```

Now only modify the store.

```
const store = createStore(reducer, applyMiddleware(logger));
```

‹ Previous

next ›