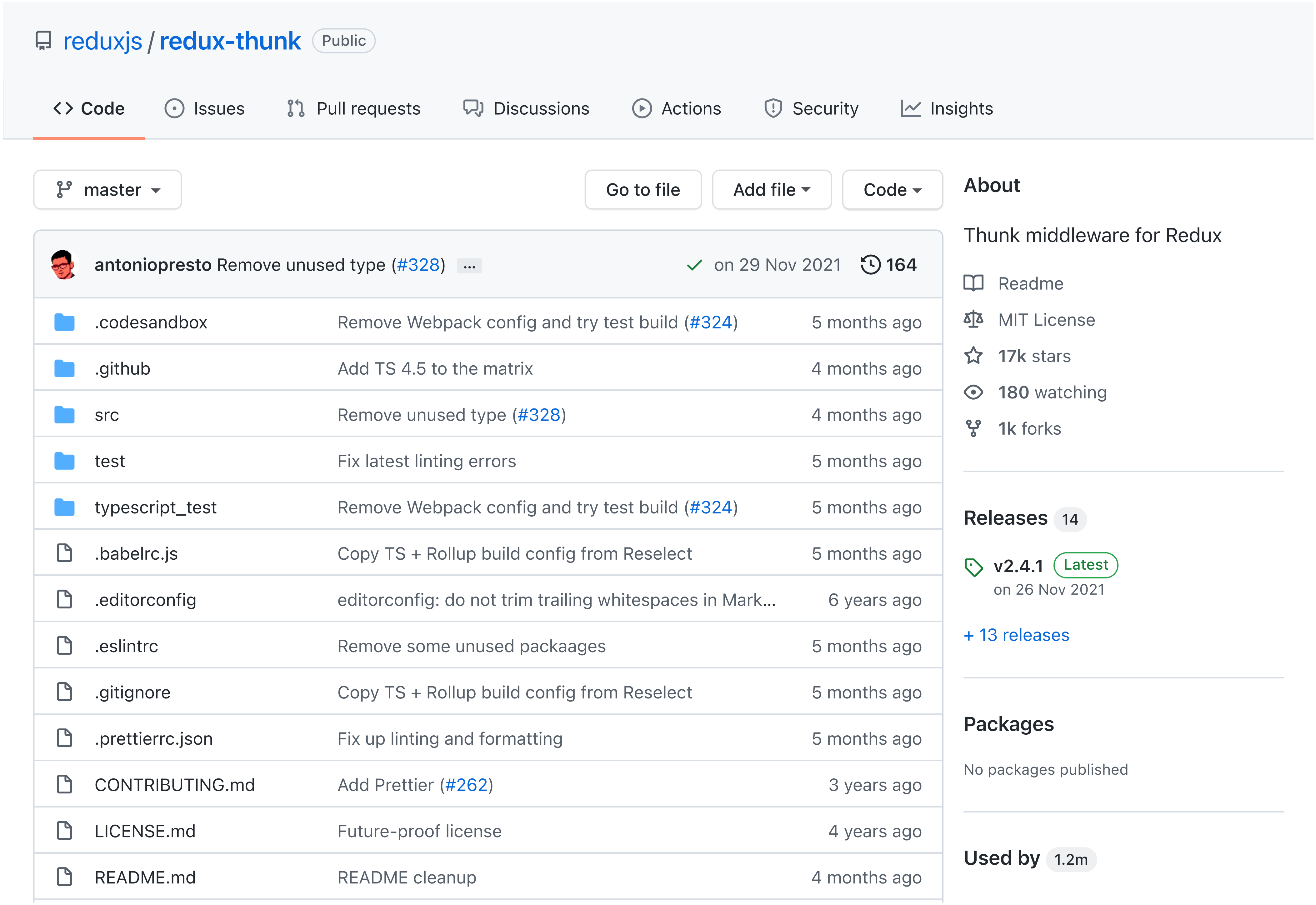

reduxjs / redux-thunk Public

antoniopresto Remove unused type (#328) … ✓ on 29 Nov 2021 164

.codesandbox	Remove Webpack config and try test build (#324)	5 months ago
.github	Add TS 4.5 to the matrix	4 months ago
src	Remove unused type (#328)	4 months ago
test	Fix latest linting errors	5 months ago
typescript_test	Remove Webpack config and try test build (#324)	5 months ago
.babelrc.js	Copy TS + Rollup build config from Reselect	5 months ago
.editorconfig	editorconfig: do not trim trailing whitespaces in Mark...	6 years ago
.eslintrc	Remove some unused packaages	5 months ago
.gitignore	Copy TS + Rollup build config from Reselect	5 months ago
.prettierrc.json	Fix up linting and formatting	5 months ago
CONTRIBUTING.md	Add Prettier (#262)	3 years ago
LICENSE.md	Future-proof license	4 years ago
README.md	README cleanup	4 months ago

About

Thunk middleware for Redux

Readme
MIT License
17k stars
180 watching
1k forks

Releases 14

v2.4.1 Latest
on 26 Nov 2021

+ 13 releases

Packages

No packages published

Used by 1.2m

extend-redux.d.ts	Remove some T's	5 months ago
jest.config.js	Convert test file to TS and Jest	5 months ago
package-lock.json	2.4.1	4 months ago
package.json	2.4.1	4 months ago
rollup.config.js	Update build setup to compile source as TS	5 months ago
tsconfig.json	Fix the rootdir stuff	5 months ago

README.md

Redux Thunk

Thunk middleware for Redux. It allows writing functions with logic inside that can interact with a Redux store's `dispatch` and `getState` methods.

For complete usage instructions and useful patterns, see the Redux docs **Writing Logic with Thunks** page.

build passing npm v2.4.1 downloads 15M/month

Installation and Setup

Redux Toolkit

If you're using our official Redux Toolkit package as recommended, there's nothing to install - RTK's `configureStore` API already adds the thunk middleware by default:

+ 1,180,190

Contributors 48

+ 37 contributors

Languages

● TypeScript 89.7%
● JavaScript 10.3%

```
import { configureStore } from '@reduxjs/toolkit'

import todosReducer from './features/todos/todosSlice'
import filtersReducer from './features/filters/filtersSlice'

const store = configureStore({
  reducer: {
    todos: todosReducer,
    filters: filtersReducer
  }
})

// The thunk middleware was automatically added
```

Manual Setup

If you're using the basic Redux `createStore` API and need to set this up manually, first add the `redux-thunk` package:

```
npm install redux-thunk

yarn add redux-thunk
```

The thunk middleware is the default export.

▶ **More Details: Importing the thunk middleware**

Then, to enable Redux Thunk, use `applyMiddleware()`:

```
import { createStore, applyMiddleware } from 'redux'
import thunk from 'redux-thunk'
import rootReducer from './reducers/index'
```

```
const store = createStore(rootReducer, applyMiddleware(thunk))
```

Injecting a Custom Argument

Since 2.1.0, Redux Thunk supports injecting a custom argument into the thunk middleware. This is typically useful for cases like using an API service layer that could be swapped out for a mock service in tests.

For Redux Toolkit, the `getDefaultMiddleware` callback inside of `configureStore` lets you pass in a custom `extraArgument`:

```
import { configureStore } from '@reduxjs/toolkit'
import rootReducer from './reducer'
import { myCustomApiService } from './api'

const store = configureStore({
  reducer: rootReducer,
  middleware: getDefaultMiddleware =>
    getDefaultMiddleware({
      thunk: {
        extraArgument: myCustomApiService
      }
    })
})

// later
function fetchUser(id) {
  // The `extraArgument` is the third arg for thunk functions
  return (dispatch, getState, api) => {
    // you can use api here
  }
}
```

If you need to pass in multiple values, combine them into a single object:

```
const store = configureStore({
  reducer: rootReducer,
  middleware: getDefaultMiddleware =>
    getDefaultMiddleware({
      thunk: {
        extraArgument: {
          api: myCustomApiService,
          otherValue: 42
        }
      }
    })
})

// later
function fetchUser(id) {
  return (dispatch, getState, { api, otherValue }) => {
    // you can use api and something else here
  }
}
```

If you're setting up the store by hand, the default `thunk` export has an attached `thunk.withExtraArgument()` function that should be used to generate the correct thunk middleware:

```
const store = createStore(
  reducer,
  applyMiddleware(thunk.withExtraArgument(api))
)
```

Why Do I Need This?

With a plain basic Redux store, you can only do simple synchronous updates by dispatching an action. Middleware extends the store's abilities, and lets you write async logic that interacts with the store.

Thunks are the recommended middleware for basic Redux side effects logic, including complex synchronous logic that needs access to the store, and simple async logic like AJAX requests.

For more details on why thunks are useful, see:

- **Redux docs: Writing Logic with Thunks**
  https://redux.js.org/usage/writing-logic-thunks
  The official usage guide page on thunks. Covers why they exist, how the thunk middleware works, and uesful patterns for using thunks.

- **Stack Overflow: Dispatching Redux Actions with a Timeout**
  http://stackoverflow.com/questions/35411423/how-to-dispatch-a-redux-action-with-a-timeout/35415559#35415559
  Dan Abramov explains the basics of managing async behavior in Redux, walking through a progressive series of approaches (inline async calls, async action creators, thunk middleware).

- **Stack Overflow: Why do we need middleware for async flow in Redux?**
  http://stackoverflow.com/questions/34570758/why-do-we-need-middleware-for-async-flow-in-redux/34599594#34599594
  Dan Abramov gives reasons for using thunks and async middleware, and some useful patterns for using thunks.

- **What the heck is a "thunk"?**
  https://daveceddia.com/what-is-a-thunk/
  A quick explanation for what the word "thunk" means in general, and for Redux specifically.

- **Thunks in Redux: The Basics**
  https://medium.com/fullstack-academy/thunks-in-redux-the-basics-85e538a3fe60
  A detailed look at what thunks are, what they solve, and how to use them.

You may also want to read the **Redux FAQ entry on choosing which async middleware to use**.

While the thunk middleware is not directly included with the Redux core library, it is used by default in our **`@reduxjs/toolkit` package**.

Motivation

Redux Thunk middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met. The inner function receives the store methods `dispatch` and `getState` as parameters.

An action creator that returns a function to perform asynchronous dispatch:

```
const INCREMENT_COUNTER = 'INCREMENT_COUNTER'

function increment() {
  return {
    type: INCREMENT_COUNTER
  }
}
```

```
function incrementAsync() {
  return dispatch => {
    setTimeout(() => {
      // Yay! Can invoke sync or async actions with `dispatch`
      dispatch(increment())
    }, 1000)
  }
}
```

An action creator that returns a function to perform conditional dispatch:

```
function incrementIfOdd() {
  return (dispatch, getState) => {
    const { counter } = getState()

    if (counter % 2 === 0) {
      return
    }

    dispatch(increment())
  }
}
```

What's a thunk?!

A thunk is a function that wraps an expression to delay its evaluation.

```
// calculation of 1 + 2 is immediate
// x === 3
let x = 1 + 2

// calculation of 1 + 2 is delayed
// foo can be called later to perform the calculation
// foo is a thunk!
let foo = () => 1 + 2
```

The term originated as a humorous past-tense version of "think".

Composition

Any return value from the inner function will be available as the return value of `dispatch` itself. This is convenient for orchestrating an asynchronous control flow with thunk action creators dispatching each other and returning Promises to wait for each other’s completion:

```
import { createStore, applyMiddleware } from 'redux'
import thunk from 'redux-thunk'
import rootReducer from './reducers'

// Note: this API requires redux@>=3.1.0
const store = createStore(rootReducer, applyMiddleware(thunk))

function fetchSecretSauce() {
  return fetch('https://www.google.com/search?q=secret+sauce')
}

// These are the normal action creators you have seen so far.
// The actions they return can be dispatched without any middleware.
// However, they only express “facts” and not the “async flow”.

function makeASandwich(forPerson, secretSauce) {
  return {
    type: 'MAKE_SANDWICH',
    forPerson,
    secretSauce
  }
}

function apologize(fromPerson, toPerson, error) {
  return {
    type: 'APOLOGIZE',
    fromPerson,
    toPerson,
    error
  }
}

function withdrawMoney(amount) {
  return {
    type: 'WITHDRAW',
    amount
  }
}

// Even without middleware, you can dispatch an action:
store.dispatch(withdrawMoney(100))

// But what do you do when you need to start an asynchronous action,
// such as an API call, or a router transition?

// Meet thunks.
// A thunk in this context is a function that can be dispatched to perform asyn
// activity and can dispatch actions and read state.
// This is an action creator that returns a thunk:
function makeASandwichWithSecretSauce(forPerson) {
  // We can invert control here by returning a function - the "thunk".
  // When this function is passed to `dispatch`, the thunk middleware will inte
  // and call it with `dispatch` and `getState` as arguments.
  // This gives the thunk function the ability to run some logic, and still int
  return function (dispatch) {
    return fetchSecretSauce().then(
        sauce => dispatch(makeASandwich(forPerson, sauce)),
        error => dispatch(apologize('The Sandwich Shop', forPerson, error))
      )
    }
  }

  // Thunk middleware lets me dispatch thunk async actions
  // as if they were actions!

  store.dispatch(makeASandwichWithSecretSauce('Me'))

  // It even takes care to return the thunk’s return value
  // from the dispatch, so I can chain Promises as long as I return them.

  store.dispatch(makeASandwichWithSecretSauce('My partner')).then(() => {
    console.log('Done!')
  })

  // In fact I can write action creators that dispatch
  // actions and async actions from other action creators,
  // and I can build my control flow with Promises.

  function makeSandwichesForEverybody() {
    return function (dispatch, getState) {
      if (!getState().sandwiches.isShopOpen) {
        // You don’t have to return Promises, but it’s a handy convention
        // so the caller can always call .then() on async dispatch result.

        return Promise.resolve()
      }

      // We can dispatch both plain object actions and other thunks,
      // which lets us compose the asynchronous actions in a single flow.

      return dispatch(makeASandwichWithSecretSauce('My Grandma'))
        .then(() =>
          Promise.all([
            dispatch(makeASandwichWithSecretSauce('Me')),
            dispatch(makeASandwichWithSecretSauce('My wife'))
          ])
        )
        .then(() => dispatch(makeASandwichWithSecretSauce('Our kids')))
        .then(() =>
          dispatch(
            getState().myMoney > 42
              ? withdrawMoney(42)
              : apologize('Me', 'The Sandwich Shop')
          )
        )
    }
  }

  // This is very useful for server side rendering, because I can wait
  // until data is available, then synchronously render the app.

  store
    .dispatch(makeSandwichesForEverybody())
    .then(() =>
      response.send(ReactDOMServer.renderToString(<MyApp store={store} />))
    )

  // I can also dispatch a thunk async action from a component
  // any time its props change to load the missing data.

  import { connect } from 'react-redux'
  import { Component } from 'react'

  class SandwichShop extends Component {
    componentDidMount() {
      this.props.dispatch(makeASandwichWithSecretSauce(this.props.forPerson))
    }
    componentDidUpdate(prevProps) {
      if (prevProps.forPerson !== this.props.forPerson) {
        this.props.dispatch(makeASandwichWithSecretSauce(this.props.forPerson))
      }
    }

    render() {
      return <p>{this.props.sandwiches.join('mustard')}</p>
    }
  }

  export default connect(state => ({
    sandwiches: state.sandwiches
  }))(SandwichShop)
```

License

MIT