EBook Gratis

APRENDIZAJE react-redux

Free unaffiliated eBook created from
Stack Overflow contributors.

#react-redux

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: react-redux

It is an unofficial and free react-redux ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official react-redux.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con react-redux

Observaciones

React Redux es una biblioteca que proporciona enlaces React para Redux.

Los componentes de React que conocen la tienda Redux se denominan "Contenedores", "Componentes inteligentes" o "Componente de orden superior" (HOC). Tales componentes, para usar Redux, necesitan:

- Suscríbete a la tienda para obtener actualizaciones de la tienda de Redux
- Acciones de despacho

Hacer esto a mano implicaría el uso de `store.subscribe` y `store.dispatch(action)` en contenedores React.

React Redux simplifica el enlace entre la tienda Redux y un componente contenedor React por medio de la función de `connect` , que asigna las propiedades del estado Redux y los creadores de acción a las propiedades del componente.

`connect` es una función que crea un componente de orden superior. Connect acepta 3 funciones (`mapStateToProps` , `mapDispatchToProps` , `mergeProps`) y devuelve un componente contenedor, que envuelve el componente original para convertirlo en un componente "conectado":

```
import { connect } from 'react-redux';

const Customers = { ... };
const mapStateToProps = (state) => { ... }
const mapDispatchToProps = (dispatch) => { ... }

export default connect(mapStateToProps, mapDispatchToProps)(Customers);
```

Vea la sección de ejemplos para un ejemplo completo.

Dado que todos los componentes del contenedor necesitan acceder a la tienda Redux, la forma recomendada es usar un componente `<Provider>` especial de React Redux, que pasa la tienda a todos los componentes secundarios (utilizando el contexto React internamente).

Documentación oficial: http://redux.js.org/docs/basics/UsageWithReact.html

Repo de GitHub: https://github.com/reactjs/react-redux

Versiones

Versión	Fecha de lanzamiento
5.0.3	2017-02-23

Versión	Fecha de lanzamiento
5.0.2	2017-01-11
5.0.1	2016-12-14
5.0.0	2016-12-14
4.4.6	2016-11-14
4.4.5	2016-04-14
4.4.4	2016-04-13
4.4.3	2016-04-12
4.4.0	2016-02-06
4.3.0	2016-02-05
4.2.0	2016-02-01
4.1.0	2016-01-28
4.0.0	2015-10-15
3.0.0	2015-09-24
2.0.0	2015-09-01
1.0.0	2015-08-24
0.5.0	2015-08-07
0.1.0	2015-07-12

Examples

Instalación o configuración

Usar `redux` directamente con `react` puede parecer un poco difícil, ya que para cada `component` que desea actualizar cuando la tienda cambia, debe suscribir ese componente a la `redux store`

React Redux se encarga de todo esto y hace que sea realmente fácil escribir componentes que pueden solicitar los datos que necesita del `redux store` y recibir notificaciones Solo cuando esos datos cambian. Esto nos permite escribir componentes realmente efectivos.

Para instalar `react-redux` todo lo que tienes que hacer es ejecutar este comando `npm`

```
npm install --save react-redux
```

Y tu estas listo.

Nota: React Redux depende de

- Reaccionar (Versión 0.14 o posterior) y
- Redux

Ejemplo completo

Supongamos que tenemos un contenedor "CustomersContainer" que conecta un componente tonto de "Clientes" a la tienda Redux.

En index.js:

```
import { Component }, React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './redux/rootReducer';
import CustomersContainer from './containers/CustomersContainer';

let store = createStore(rootReducer);

render(
  <Provider store={store}>
    <CustomersContainer />
  </Provider>,
  document.getElementById('root')
);
```

En CustomersContainer:

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

// Import action creators
import { fetchCustomers } from '../redux/actions';

// Import dumb component
import Customers from '../components/Customers';

// ES6 class declaration
class CustomersContainer extends Component {
  componentWillMount() {
    // Action fetchCustomers mapped to prop fetchCustomers
    this.props.fetchCustomers();
  }

  render() {
    return <Customers customers={this.props.customers} />;
  }
}

function mapStateToProps(state) {
  return {
```

```
    customers: state.customers
  };
}

// Here we use the shorthand notation for mapDispatchToProps
// it can be used when the props and action creators have the same name
const CustomersContainer = connect(mapStateToProps, { fetchCustomers })(CustomersContainer);

export default CustomersContainer;
```

Hola mundo usando React Redux

Esta guía asume que ya ha instalado `react` , `redux` , `react-router` y **`react-redux`** y que ha configurado `react` , `redux` y `react-router` . Si no lo ha hecho, hágalo.

Nota: Si bien `react-router` no es una dependencia de `react-redux` , es muy probable que lo usemos en nuestra aplicación de reacción para el enrutamiento y esto hace que sea realmente fácil para nosotros usar react-redux.

NOMBRE DE `app.js` **:** `app.js`

```
'use strict';

import React from 'react';
import { render } from 'react-dom';
import { Router, Route, Link, browserHistory, IndexRoute } from 'react-router';
import { Provider } from 'react-redux';
import store from './stores';

render(
  (
    <Provider store={ store }>
      <Router history={ browserHistory }>
        {/* all the routes here */}
      </Router>
    </Provider>
  ),
  document.getElementById('app')
);
```

Este archivo tendrá sentido para la mayoría de ustedes. Lo que estamos haciendo aquí es obtener la **tienda** de `./stores` y pasarla a todas las rutas utilizando el **`Provider`** **componentes de orden superior** proporcionado por `react-redux` .

Esto hace que la **tienda** esté disponible a través de nuestra aplicación.

Ahora, consideremos este escenario . Tenemos un componente `UserComponent` que obtiene los datos del reductor de `user` y tiene un botón que, al hacer clic, actualiza los datos en la tienda.

Estructura de la aplicación

Nuestro `rootReducer` tiene `user` reductor.

```
const rootReducer = combineReducers({
    user: userReducer,
})
export default rootReducer;
```

Nuestro `userReducer` ve así

```
const default_state = {
  users: [],
  current_user: {
    name: 'John Doe',
    email: 'john.doe@gmail.com',
    gender: 'Male'
  },
  etc: {}
};

function userReducer( state=default_state, action ) {

  if ( action.type === "UPDATE_CURRENT_USER_DATA" ) {
    return Object.assign( {}, state, { current_user: Object.assign( {}, state.current_user, {
[action.payload.field]: action.payload.value } ) } );
  }
  else {
    return state;
  }

}

export default userReducer;
```

Y nuestro archivo de `actions` ve algo como esto.

```
export function updateCurrentUserData( data ) {
  return {
    type: "UPDATE_CURRENT_USER_DATA",
    payload: data
  }
}
```

Finalmente, trabajemos en nuestro componente.

`UserComponent.js` : `UserComponent.js`

```
'use strict';

import React from 'react';
import { connect } from 'react-redux';
import * as Action from './actions';

let UserComponent = (props) => {

  let changeUserDetails = (field, value) => {
    // do nothing
  }
```

```
  return(
    <div>
      <h1>Hello { props.current_user.name }</h1>
      <p>Your email address is { props.current_user.email }</p>
      <div style={{ marginTop: 30 }}>
        <button onClick={ () => { changeUserDetails('name', 'Jame Smith') } }>Change
Name</button>
        <button onClick={ () => { changeUserDetails('email', 'jane@gmail.com') } }>Change
Email Address</button>
      </div>
    </div>
  )

}

export default UserComponent;
```

Por supuesto, esto **no funcionará** , ya que todavía no lo hemos conectado a la tienda.

> En caso de que se lo pregunte, este es un **componente funcional sin estado** , ya que estamos usando `redux` y realmente no necesitamos un estado interno para nuestro componente, este es el momento adecuado para usarlo.

El método de `connect` provisto por `react-redux` toma en tres parámetros

mapStateToProps , **mapDispatchToProps** y el propio **componente** .

```
connect( mapStateToProps, mapDispatchToProps )(Component)
```

Añadamos **conectar** a nuestro componente **UserComponent** junto con **mapStateToProps** y **mapDispatchToProps**

Y también actualicemos nuestra función **changeUserDetails** , de modo que cuando se le **solicite** , `dispatch` una `action` a nuestros `reducers` , y en función del tipo de acción que nuestro reductor activará y realizará cambios en la tienda, y una vez que la tienda se actualice, volverá a aparecer la `react-redux` -Rendra nuestro componente con los nuevos datos.

¿Suena complicado? Realmente no lo es.

Nuestro **`UserComponent.js`** se verá como

```
'use strict';

import React from 'react';
import { connect } from 'react-redux';
import * as Action from './actions';

const mapStateToProps = ( state, ownProps ) => {
  return {
    current_user: state.user.current_user,
  }
}

const mapDispatchToProps = ( dispatch, ownProps ) => {
  return {
```

```
    updateCurrentUserData: (payload) => dispatch( Action.updateCurrentUserData(payload) ),
  }
}


let UserComponent = (props) => {

  let changeUserDetails = (field, value) => {
    props.updateCurrentUserData({ field: field, value: value });
  }

  return(
    <div>
      <h1>Hello { props.current_user.name }</h1>
      <p>Your email address is { props.current_user.email }</p>
      <div style={{ marginTop: 30 }}>
        <button onClick={ () => { changeUserDetails('name', 'Jame Smith') } }>Change
Name</button>
        <button onClick={ () => { changeUserDetails('email', 'jane@gmail.com') } }>Change
Email Address</button>
      </div>
    </div>
  )

}

const ConnectedUserComponent = connect(
  mapStateToProps,
  mapDispatchToProps
)(UserComponent)

export default ConnectedUserComponent;
```

Lo que hicimos aquí es agregado.

- **mapStateToProps** : Esto nos permite obtener los datos de la tienda y cuando **esos** datos cambian, nuestro componente se volverá a representar con los nuevos datos.

  Nuestro componente solo se volverá a representar si los datos de nuestro **componente** solicitan cambios en la tienda y no cuando otros datos cambian en la tienda.

- **mapDispatchToProps** : Esto nos permite `dispatch actions` a todos los reductores desde nuestro componente ... (podría ser cualquier componente), y según el `type` de acción, nuestro UserReducer activará y devolverá un nuevo estado con los datos actualizados.

- **ConnectedUserComponent** : Por último, conectamos nuestro componente a la tienda utilizando el método de `connect` pasando todos los parámetros y `exported` el componente conectado.

- También actualizamos nuestra función **changeUserDetails** para llamar al `method` en props y también pasar los datos. Y `props` a su vez envía el método que llamamos a todos los reductores.

NOTA:

- Si no devolvemos un nuevo estado desde el reductor, `react-redux` no volverá a renderizar nuestro componente.

Lea Empezando con react-redux en línea: https://riptutorial.com/es/react-redux/topic/5797/empezando-con-react-redux

Creditos

S. No	Capítulos	Contributors
1	Empezando con react-redux	Alexg2195, Community, Matteo Frana, Ori Drori, Random User, Thibaut Remy