¿Qué es REDUX?

Al ser un desarrollador web, siempre debe aprender nuevos marcos y bibliotecas. En los últimos tiempos, la biblioteca más popular y de moda es ReactJs. Cuando aprenda ReactJs, puede encontrarse con la administración estatal. La gestión del estado es crucial en las aplicaciones de ReactJ. Cuando se trata de administración de estado en ReactJs, **REDUX** es una de las bibliotecas más populares. Redux es una biblioteca para administrar datos en sus aplicaciones. Realmente no necesita usar Redux hasta que tenga algunos problemas relacionados con el estado o su aplicación tenga una estructura de datos muy compleja y la aplicación requiera muchas actualizaciones con frecuencia.

¿Por qué Redux?

Si. Debería preguntarse por qué debería utilizar Redux. La motivación es muy simple, Redux te permite predecir el estado de la aplicación en cualquier momento. Imagine que en su aplicación de página única, un modelo actualiza una vista y otro modelo actualiza otra vista, lo que a su vez provoca la actualización de otra vista. Si esto continúa, ya no comprenderá lo que está sucediendo en su aplicación y es muy difícil para usted depurar su aplicación. La idea principal de Redux es mantener los datos de su aplicación en un solo lugar. Eso significa que todos los datos de su aplicación se almacenarán en un solo objeto.

¿Cómo funciona Redux?

Redux mantiene todos sus datos en una sola tienda. Debe suscribirse a la tienda si necesita volver a renderizar la vista cuando cambia el estado. Debe enviar un evento cuando

desee actualizar el estado. En Redux, los datos fluyen en una sola dirección. La tienda es inmutable. Por cada cambio / evento / acción se creará una nueva tienda.

Aquí no voy a hablar sobre cómo conectar reaccionar con redux. Mi idea es decirte lo simple que es redux. Una vez que entiendas esto, estoy seguro de que te encantará Redux.

Te mostraré cómo funciona redux. Vamos a sumergirnos en ello. Lo primero es crear una tienda. Para crear una tienda, la biblioteca redux proporciona la función **createStore** . Para instalar redux, ejecute este comando. Supongo que todos estamos familiarizados con ES6 y npm.

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

Después de instalar import **createStore** de redux

```
importar {createStore} desde 'redux'
```

Antes de crear una tienda, debemos aprender dos cosas importantes. **Acciones** y **Reductores.**

Acciones: Las acciones son una carga útil de información que se envía desde la aplicación a la tienda. Las acciones no son más que eventos en su aplicación. por ejemplo, al hacer clic en el botón que desea actualizar los datos. Para enviar un evento a la tienda, debe enviar una acción. La acción contiene propiedades de tipo y datos. Los reductores utilizan la propiedad de tipo para modificar su tienda según el tipo que sea. La acción simple se ve así

```
{tipo: "UPDATE_STORE", datos: "datos nuevos"}
```

Podemos enviar una acción para almacenar usando la función de envío. La función de envío está disponible en el objeto de la tienda.

```
store.dispatch ({tipo: ”UPDATE_STORE”, datos: ”nuevos datos”});
```

Reductores: los reductores no son más que funciones puras y devuelven una nueva tienda cuando envía una acción. Los reductores devuelven una nueva tienda con nuevos datos o datos actualizados en respuesta a la acción que se desencadena. Basado en la acción solo los reductores modifica su tienda. Las acciones son solo eventos para actualizar su tienda. Los reductores son responsables de devolver una tienda nueva cuando desencadena una acción. Según la acción, se devolverá la nueva tienda. Los reductores típicos se ven así

```
function myReducer (store = initialStore, action) {
   switch (action.type) {
        ......................
   }
   // finalmente devuelve tienda
   return store;}
```

Para trabajar con la tienda es necesario pasarle reductor. **La** función **store.createStore ()** toma reductor y nos devuelve el objeto de la tienda. El objeto store tiene funciones ***dispatch ()*** , ***subscribe ()*** , ***unsubscribe ()*** y ***getState ()*** .

```
let store = createStore (reductor);
```

Si desea enviar una acción para almacenar, use la función **store.dispatch ()** . Para detectar cambios en la tienda, utilice la función **store.subscribe (callback)** . Para obtener datos, use la función **store.getState ()** . Aquí he escrito un ejemplo muy básico que muestra cómo funciona Redux.

En este ejemplo, simplemente sumamos / restamos el número para almacenar según la acción. Cuando enviamos **store.dispatch ({type: ”ADD”, number: 10})** reducer agrega un número a la tienda y devuelve una nueva tienda.

Espero que haya entendido los conceptos básicos de Redux. Hice todo lo posible para simplificar tanto como me fue posible. En mi próximo artículo me centraré en cómo

conectar reaccionar con redux. Puedes leer sobre redux aquí . Te sugiero que pases por redux antes de sumergirte en react-redux. Si tiene alguna sugerencia, no dude en comentar.