UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Avanzado

Clase N° 16: React JS + Redux

Profesor: Ing. Leandro Rodolfo Gil Carrano
Email: leangilutn@gmail.com

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Redux

Redux

Comencemos enfatizando que Redux no tiene relación alguna con React. Puedes escribir aplicaciones Redux con React, Angular, Ember, jQuery o vanilla JavaScript.

Dicho esto, Redux funciona especialmente bien con librerías como React porque te permiten describir la interfaz de usuario como una función de estado, y Redux emite actualizaciones de estado en respuesta a acciones.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Redux - Instalación

Debemos ejecutar el comando:

npm install --save react-redux

```
E:\Personal2\sites\react\react-tutorial>npm install --save react-redux
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Redux - Componentes

	Componentes de Presentación	**Componentes Contenedores**
Propósito	Como se ven las cosas (*markup*, estilos)	Como funcionan las cosas (búsqueda de datos, actualizaciones de estado)
Pertinente a Redux	No	Yes
Para leer datos	Lee datos de los *props*	Se suscribe al estado en Redux
Para manipular datos	Invoca llamada de retorno (callback) desde los *props*	Envia acciones a Redux
Son escritas	Manualmente	Usualmente generados por React Redux

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Acciones

Redux - Acciones

Las acciones son un bloque de información que envía datos desde tu aplicación a tu store. Son la única fuente de información para el store. Las envias al store usando store.dispatch().

```
function addTodoWithDispatch(text) {
  const action = {
    type: ADD_TODO,
    text
  }
  dispatch(action)
}
```

Las acciones son objetos planos de JavaScript. Una acción debe tener una propiedad type que indica el tipo de acción a realizar. Los tipos normalmente son definidos como strings constantes.

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Reducers

Redux - Reducers

Las acciones describen que algo pasó, pero no especifican cómo cambió el estado de la aplicación en respuesta. Esto es trabajo de los reducers.

En Redux, todo el estado de la aplicación es almacenado en un único objeto. Es una buena idea pensar en su forma antes de escribir código. ¿Cuál es la mínima representación del estado de la aplicación como un objeto?

Para nuestra aplicación de tareas, vamos a querer guardar dos cosas diferentes:

El filtro de visibilidad actualmente seleccionado;
La lista actual de tareas.

Redux - Reducers

```
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Store

Redux - Store

El Store es el objeto que los reúne. El store tiene las siguientes responsabilidades:

- Contiene el estado de la aplicación;
- Permite el acceso al estado vía getState();
- Permite que el estado sea actualizado via dispatch(action);
- Registra los listeners vía subscribe(listener);
- Maneja la anulación del registro de los listeners vía el retorno de la función de subscribe(listener).

Es importante destacar que sólo tendrás un store en una aplicación Redux. Cuando desees dividir la lógica para el manejo de datos, usarás composición de reductores en lugar de muchos stores.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Flujo de datos

Redux - Flujo de datos

1. Haces una llamada a store.dispatch(action)
2. El store en Redux invoca a la función reductora que le indicaste. El store pasará dos argumentos al reductor: el árbol de estado actual y la acción.
3. El reductor raíz puede combinar la salida de múltiples reductores en un único árbol de estado.
   Como se estructura el reductor raíz queda completamente a tu discreción. Redux provee una función combineReducers() que ayuda, a "dividir" el reductor raíz en funciones separadas donde cada una maneja una porción del árbol de estado.
4. El store en Redux guarda por completo el árbol de estado devuelto por el reductor raíz.

Redux - Instalación

El reducer es una función pura que toma el estado anterior y una acción, y devuelve en nuevo estado.

Es muy importante que los reducer se mantengan puros. Cosas que nunca deberías hacer dentro de un reducer:

Modificar sus argumentos;

Realizar tareas con efectos secundarios como llamas a un API o transiciones de rutas.

Llamar una función no pura, por ejemplo Date.now() o Math.random().

Dados los mismos argumentos, debería calcular y devolver el siguiente estado. Sin sorpresas. Sin efectos secundarios. Sin llamadas a APIs. Sin mutaciones. Solo cálculos.

Ejemplo

Redux - Instalación

La mayoría de los componentes que escribiremos serán de presentación, pero necesitaremos generar algunos componentes contenedores para conectarlos al store que maneja Redux. Con esto y el resumen de diseño que mencionaremos a continuación no implica que los componentes contenedores deban estar cerca o en la parte superior del árbol de componentes. Si un componente contenedor se vuelve demasiado complejo (es decir, tiene componentes de presentación fuertemente anidados con innumerables devoluciones de llamadas que se pasan hacia abajo), introduzca otro contenedor dentro del árbol de componentes

Redux - Ejemplo

Queremos mostrar una lista de asuntos pendientes. Al hacer clic, un elemento de la lista se tachará como completado. Queremos mostrar un campo en el que el usuario puede agregar una tarea nueva. En el pie de página, queremos mostrar un toggle para mostrar todas las taras, sólo las completadas, o sólo las activas.

Redux - Componentes de presentación

- TodoList es una lista que mostrará las tareas pendientes disponibles.
- todos: Array es un arreglo de tareas pendientes que contiene la siguiente descripción { id, text, completed }.
- onTodoClick(id: number) es un callback para invocar cuando un asunto pendientes es presionado.
- Todo es un asunto pendiente.
- text: string es el texto a mostrar.
- completed: boolean indica si la tarea debe aparecer tachada.
- onClick() es un callback para invocar cuando la tarea es presionada.
- Link es el enlace con su callback.
- onClick() es un callback para invocar cuando el enlace es presionado.
- Footer es donde dejamos que el usuario cambie las tareas pendientes visibles actualmente.
- App es el componente raíz que representa todo lo demás.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Redux - Componentes contenedores

También necesitaremos algunos componentes contenedores para conectar los componentes de presentación a Redux. Por ejemplo, el componente de presentación TodoList necesita un contenedor comoVisibleTodoList que se suscribe al store de Redux y debe saber cómo aplicar el filtro de visibilidad. Para cambiar el filtro de visibilidad, proporcionaremos un componente contenedor FilterLink que renderiza un Link que distribuye la debida acción al hacer clic:

- VisibleTodoList filtra los asuntos de acuerdo a la visibilidad actual y renderiza el TodoList.
- FilterLink obtiene el filtro de visibilidad actual y renderiza un Link.
- filter: string es el tipo del filtro de visibilidad.

Redux - Otros componentes

A veces es difícil saber si un componente debe ser componente de presentación o contenedor. Por ejemplo, a veces la forma y la función están realmente entrelazadas, como en el caso de este pequeño componente:

AddTodo es un campo de entrada con un botón "Añadir tarea" Técnicamente podríamos dividirlo en dos componentes, pero podría ser demasiado pronto en esta etapa. Está bien mezclar presentación y lógica en un componente que sea muy pequeño. A medida que crece, será más obvio cómo dividirlo, así que lo dejaremos en uno solo.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Componentes Presentación

Redux - components/todo.js

```
import React from 'react'
import PropTypes from 'prop-types'
import { List } from 'semantic-ui-react'

const Todo = ({ onClick, completed, text, highPriority }) => (
    <List.Item
        onClick={onClick}
        style={{
            textDecoration: completed ? 'line-through' : 'none'
        }}
    >
        <List.Icon name={highPriority ? 'bell' : 'none'}
            size='large'
            verticalAlign='middle' />
        <List.Content><List.Header>{text}</List.Header></List.Content>
    </List.Item>
)

Todo.propTypes = {
    onClick: PropTypes.func.isRequired,
    completed: PropTypes.bool.isRequired,
    text: PropTypes.string.isRequired,
    highPriority: PropTypes.bool.isRequired
}

export default Todo
```

Redux - components/todolist.js

```
import React from 'react'
import PropTypes from 'prop-types'
import Todo from './Todo'
import { Segment, List } from 'semantic-ui-react'

const TodoList = ({ todos, onTodoClick }) => (
  <Segment>
    <List divided relaxed>
      {todos.map((todo, index) => (
        <Todo key={index} {...todo} onClick={() => onTodoClick(index)} />
      ))}
    </List>
  </Segment>
)

TodoList.propTypes = {
  todos: PropTypes.arrayOf(
    PropTypes.shape({
      id: PropTypes.number.isRequired,
      completed: PropTypes.bool.isRequired,
      text: PropTypes.string.isRequired,
      highPriority: PropTypes.bool.isRequired
    }).isRequired
  ).isRequired,
  onTodoClick: PropTypes.func.isRequired
}

export default TodoList
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Redux - components/link.js

```
import React from 'react'
import PropTypes from 'prop-types'
import { Button } from 'semantic-ui-react'

const Link = ({ active, children, onClick }) => {
    if (active) {
        return <span>{children}</span>
    }

    return (
        <Button
            onClick={e => {
                e.preventDefault()
                onClick()
            }}
        >
            {children}
        </Button>
    )
}

Link.propTypes = {
    active: PropTypes.bool.isRequired,
    children: PropTypes.node.isRequired,
    onClick: PropTypes.func.isRequired
}
export default Link
```

Redux - components/footer.js

```
import React from 'react'
import FilterLink from '../containers/FilterLink'
import { VisibilityFilters } from '../actions'
import { Button } from 'semantic-ui-react'

const Footer = () => (
    <Button.Group>
        <FilterLink filter={VisibilityFilters.SHOW_ALL}>All</FilterLink>
        <FilterLink filter={VisibilityFilters.SHOW_ACTIVE}>Active</FilterLink>
        <FilterLink filter={VisibilityFilters.SHOW_COMPLETED}>Completed</FilterLink>
        <FilterLink filter={VisibilityFilters.SHOW_HIGH_PRIORITY}>High Priority</FilterLink>
    </Button.Group>
)
export default Footer
```

Redux - components/app.js

```
import React from 'react'
import './App.css'
import Footer from './components/Footer'
import AddTodo from './containers/AddTodo'
import VisibleTodoList from './containers/VisibleTodoList'

const App = () => (
  <div className="App">
    <AddTodo />
    <VisibleTodoList />
    <Footer />
  </div>
)
export default App;
```

Componentes Contenedores

Redux - visibletodolist.js

Es el momento de conectar los componentes de presentación a Redux mediante la creación de algunos contenedores.
Generamos los componentes contenedores con la función connect() de la librería React Redux, ya que proporciona muchas optimizaciones útiles para evitar re-renders innecesarios. (Un beneficio de utilizar esta librería es que usted no tiene que preocuparse por la implementación del método **shouldComponentUpdate** recomendado por React para mejor rendimiento.)

Redux - visibletodolist.js

Para usar c**onnect()**, es necesario definir una función especial llamada **mapStateToProps** que indica cómo transformar el estado actual del store Redux en los props que desea pasar a un componente de presentación. Por ejemplo, **VisibleTodoList** necesita calcular todos para pasar a TodoList, así que definimos una función que filtra el **state.todos** de acuerdo con el **state.visibilityFilter**, y lo usamos en su **mapStateToProps**

Redux - visibletodolist.js

```
const getVisibleTodos = (todos, filter) => {
    switch (filter) {
        case 'SHOW_ALL':
            return todos
        case 'SHOW_COMPLETED':
            return todos.filter(t => t.completed)
        case 'SHOW_ACTIVE':
            return todos.filter(t => !t.completed)
        case 'SHOW_HIGH_PRIORITY':
            return todos.filter(t => t.highPriority)
        default:
            throw new Error('Unknown filter: ' + filter)
    }
}

const mapStateToProps = state => {
    return {
        todos: getVisibleTodos(state.todos, state.visibilityFilter)
    }
}
```

Redux - visibletodolist.js

Además de leer el estado, los componentes contenedores pueden enviar acciones. De manera similar, puede definir una función llamada mapDispatchToProps() que recibe el método dispatch() y devuelve los callback props que deseas inyectar en el componente de presentación.

```
const mapDispatchToProps = dispatch => {
    return {
        onTodoClick: id => {
            dispatch(toggleTodo(id))
        }
    }
}

const VisibleTodoList = connect(
    mapStateToProps,
    mapDispatchToProps
)(TodoList)

export default VisibleTodoList
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Redux - filterlink.js

```
import { connect } from 'react-redux'
import { setVisibilityFilter } from '../actions'
import Link from '../components/Link'

const mapStateToProps = (state, ownProps) => {
  return {
    active: ownProps.filter === state.visibilityFilter
  }
}

const mapDispatchToProps = (dispatch, ownProps) => {
  return {
    onClick: () => {
      dispatch(setVisibilityFilter(ownProps.filter))
    }
  }
}

const FilterLink = connect(
  mapStateToProps,
  mapDispatchToProps
)(Link)

export default FilterLink
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Transferir al store

Redux - visibletodolist.js

Todos los componentes contenedores necesitan acceso al store Redux para que puedan suscribirse a ella. Una opción sería pasarlo como un prop a cada componente contenedor. Sin embargo, se vuelve tedioso, ya que hay que enlazar store incluso a través del componentes de presentación ya que puede suceder que tenga que renderizar un contenedor allá en lo profundo del árbol de componentes.

La opción que recomendamos es usar un componente React Redux especial llamado <Proveedor> para mágicamente hacer que el store esté disponible para todos los componentes del contenedor en la aplicación sin pasarlo explícitamente. Sólo es necesario utilizarlo una vez al renderizar el componente raíz

Redux - index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import { Provider } from 'react-redux'
import { createStore } from 'redux'
import todoApp from './reducers'

const store = createStore(todoApp)

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