Redux VS MobX

——Min Zhou

Similarity of Redux and MobX

- Both Redux and MobX are state management libraries
- State is one of the most important things
- They can work well with many different frameworks
- They are perfect for React

Redux——An easiest application

```
//implemnting store

const createStore = (Reducer) => {

    let state;
    let listeners = [];

    const getState = () => state;

    const dispatch = (action) => {
        state = Reducer(state,action);
        listeners.forEach(listener => listener());
    };

    const subscribe = (listener) => {
        listeners.push(listener);
        return () => {
            listeners = listeners.filter(ele => ele !== listener);
        }
    }
    dispatch({});

    return {getState, dispatch, subscribe};
}
```

- Create a store object using the reducer as input, the object has three methods—getState(), dispatch() and subscribe();

Redux——An easiest application

```
// How Redux work with React

const Counter = ({value,onIncrement,onDecrement}) => {

    return (
        <h1>{value}</h1>
        <button onClick={onIncrement}>+</button>
        <button onClick={onDecrement}>-</button>
    )
}

const render = () => {
    ReactDOM.render(
        <Counter value={store.getState(),onIncrement={
            () => store.dispatch({type: 'add'})}} />,
        document.getElementById('root');
    )
}
const store = createStore(Reducer);
store.subscribe(render);
```

- Subscribe an action to the store, usually the action is render() function when you use react with redux.

MobX——An easiest application

- Make an observable state, using the @observable decorator or observable functions to make the state trackable for MobX.
- Using @computed decorator to create functions which can derive data from state automatically
- Using autorun function, whenever the observable state changes, this function would run automatically.

MobX——An easiest application

```
class ObservableTodoStore {
    @observable todos = [];
    @observable pendingRequests = 0;

    constructor() {
        mobx.autorun(() => console.log(this.report));
    }

    @computed get completedTodosCount() {
        return this.todos.filter(
            todo => todo.completed === true
        ).length;
    }

    @computed get report() {
        if (this.todos.length === 0)
            return "<none>";
        return `Next todo: "${this.todos[0].task}". ` +
            `Progress: ${this.completedTodosCount}/${this.todos.length}`;
    }

    addTodo(task) {
        this.todos.push({
            task: task,
            completed: false,
            assignee: null
        });
    }
}

const observableTodoStore = new ObservableTodoStore();
```

MobX——An easiest application

```
import { observable } from 'mobx';
import { observer } from 'react-mobx';
import React, { Component } from 'react';
import ReactDOM from 'react-dom';

const appState = observable({
  count: 0,
});
appState.increment = function() {
  this.count ++;
};
appState.decrement = function() {
  this.count --;
};

@observer
class Count extends Component {
  render() {
    return (<div>
      Counter: { appState.count } <br />
      <button onClick={this.handleInc}> + </button>
      <button onClick={this.handleDec}> - </button>
    </div>);
  }
  handleInc() {
    appState.increment();
  }
  handleDec() {
    appState.decrement();
  }
}

ReactDOM.render(<Count />, document.getElementById('root'));
```

- When you use MobX with React, Use @observer decorator from the mobx-react package to wrap the render function into autorun, So when the state is changed, the component would re-render

Difference of Redux and MobX

- Redux
  - single store
  - functional programming paradigm
  - immutable
  - pure
  - plain JavaScript
  - more boilerplate
  - normalized state
- MobX
  - multiple stores
  - object-oriented programming and reactive programming paradigms
  - mutable
  - impure
  - “magic” JavaScript
  - less boilerplate
  - denormalized state

Difference of Redux and MobX

- use Redux over MobX:
  - clear constraints
  - Easy test
  - mature best practices
  - used in complex applications
  - immutable data
  - in a bigger size & several developers / teams project

- use MobX over Redux:
  - short learning curve
  - simple to use (magic)
  - quick start
  - minimal boilerplate
  - used in lightweight applications
  - but can be used in bigger size projects too, when used with explicit constraints