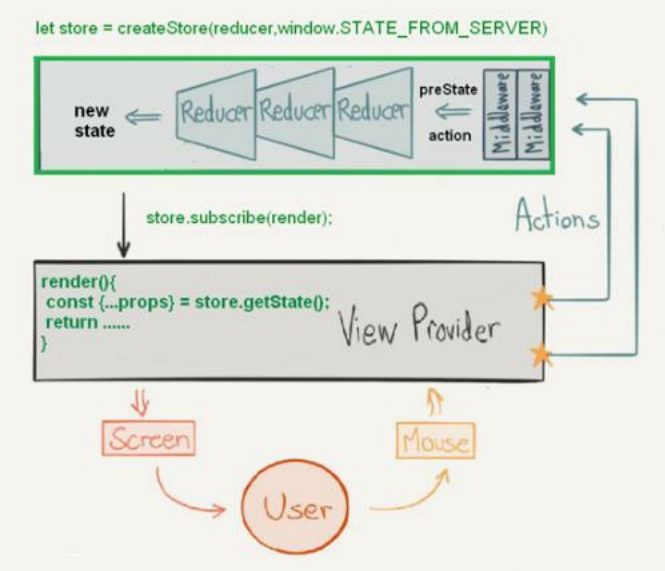
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=68 sentences=194, words=1508

1. Redux中间件

- 如果没有中间件的运用,redux 的工作流程是这样 `action -> reducer`，这是相当于同步操作，由dispatch 触发action后，直接去reducer执行相应的动作
- 但是在某些比较复杂的业务逻辑中，这种同步的实现方式并不能很好的解决我们的问题。比如我们有一个这样的需求，点击按钮 -> 获取服务器数据 -> 渲染视图，因为获取服务器数据是需要异步实现，所以这时候我就需要引入中间件改变redux同步执行的流程，形成异步流程来实现我们所要的逻辑，有了中间件，redux 的工作流程就变成这样 action -> middlewares -> reducer，点击按钮就相当于dispatch 触发action，接下去获取服务器数据 middlewares 的执行，当 middlewares 成功获取到服务器就去触发reducer对应的动作，更新需要渲染视图的数据
- 中间件的机制可以让我们改变数据流，实现如异步 action ，action 过滤，日志输出，异常报告等功能。



2. 日志中间件

- 我们改写了 `dispatch`方法,实现了在更改状态时打印前后的状态
- 但是这种方案并不好。所以我们可以采用中间的方式

2.1 实现日志

src\store\index.tsx

```
import { createStore, Store, AnyAction } from '../redux';
import reducer from './reducers';
import { CombinedState } from './reducers';
const store: Store = createStore(reducer, { counter1: { number: 0 }, counter2: { number: 0 } });
let dispatch = store.dispatch;
store.dispatch = function (action: AnyAction): AnyAction {
    console.log(store.getState());
    dispatch(action);
    console.log(store.getState());
    return action;
};
export default store;
```

2.2 实现异步

src\store\index.tsx

```
import { createStore, Store, AnyAction } from '../redux';
import reducer from './reducers';
import { CombinedState } from './reducers';
const store: Store = createStore(reducer, { counter1: { number: 0 }, counter2: { number: 0 } });
let dispatch = store.dispatch;
store.dispatch = function (action: AnyAction): AnyAction {
    setTimeout(() => {
        dispatch(action);
    }, 1000);
    return action;
};
export default store;
```

3. 日志中间件

3.1 store\index.tsx

src\store\index.tsx

```
+import { createStore, Store, AnyAction, Middleware, MiddlewareAPI, StoreEnhancer, StoreEnhancerStoreCreator, applyMiddleware } from '../redux';
import reducer from './reducers';
import { CombinedState } from './reducers';
+//const store: Store = createStore(reducer, { counter1: { number: 0 }, counter2: { number: 0 } });
+let logger: Middleware = (api: MiddlewareAPI) => (next: any) => (action: any) => {
+    console.log(api.getState());
+    next(action);
+    console.log(api.getState());
+    return action;
+};
+let storeEnhancer: StoreEnhancer = applyMiddleware(logger);
+let storeEnhancerStoreCreator: StoreEnhancerStoreCreator = storeEnhancer(createStore);
+let store: Store = storeEnhancerStoreCreator(reducer);
+export default store;
```

3.2 redux\types.tsx

src\redux\types.tsx

```
export interface Action {
    type: T
}

export interface AnyAction extends Action {
    [extraProps: string]: any
}

export type Reducer = (
    state: S | undefined,
    action: A
) => S

+export interface Dispatch {
+    (action: T): T
+}

export interface Unsubscribe {
    (): void
}
export interface Store {
    dispatch: Dispatch;
    getState(): S;
    subscribe(listener: () => void): Unsubscribe;
}

+export interface Middleware<
+    DispatchExt = {},
+    S = any,
+    D extends Dispatch = Dispatch
+    > {
+    (api: MiddlewareAPI): (
+        next: Dispatch
+    ) => (action: any) => any
+}

+export interface MiddlewareAPI {
+    dispatch: D
+    getState(): S
+}

+export type StoreEnhancer = (
+    next: StoreEnhancerStoreCreator
+) => StoreEnhancerStoreCreator
+export type StoreEnhancerStoreCreator = <
+    S = any,
+    A extends Action = AnyAction
+    >(
+    reducer: Reducer,
+    preloadedState?: S
+) => Store
```

3.3 redux\index.tsx

src\redux\index.tsx

```
import createStore from './createStore';
import bindActionCreators from './bindActionCreators';
import combineReducers from './combineReducers';
+import applyMiddleware from './applyMiddleware'
export {
    createStore,
    bindActionCreators,
    combineReducers,
+    applyMiddleware
}
export * from './types';
export * from './bindActionCreators';
export * from './combineReducers';
+export * from './applyMiddleware';
```

3.4 redux\compose.tsx

src\redux\compose.tsx

- compose (https://github.com/reduxjs/redux/blob/master/src/compose.ts)

```
function add1(str){
    return '1'+str;
}
function add2(str){
    return '2'+str;
}
function add3(str){
    return '3'+str;
}

function compose(...funcs){
    return funcs.reduce((a,b)=>(...args)=>a(b(...args)));
}

let result = compose(add3,add2,add1)('zhufeng');
console.log(result);
```

```
export default function compose(...funcs: Function[]) {
    return funcs.reduce((a, b) => (...args: any) => a(b(...args)))
}
```

3.5 applyMiddleware.tsx

src\redux\applyMiddleware.tsx

```
import compose from './compose';
import { Middleware, Store, StoreEnhancer, Dispatch, MiddlewareAPI, StoreCreator, Action, AnyAction, Reducer } from './'
export function applyMiddleware(
    ...middlewares: Middleware[]
): StoreEnhancer
export default function applyMiddleware(
    ...middlewares: Middleware[]
): StoreEnhancer {
    return (createStore: StoreCreator) => (
        reducer: Reducer
    ): Store => {
        const store = createStore(reducer)
        let dispatch: Dispatch;

        const middlewareAPI: MiddlewareAPI = {
            getState: store.getState,
            dispatch: (action) => dispatch(action)
        }
        const chain = middlewares.map(middleware => middleware(middlewareAPI))
        dispatch = compose(...chain)(store.dispatch)
        return {
            ...store,
            dispatch
        }
    }
}
```

函数兼容性判断

```

let x: string | number;
let y: string;
y = x;
x = y;

==========================

export interface Action {
    type: string
}
export interface NameAction extends Action {
    name: string;
}
export interface AgeAction extends Action {
    age: number;
}
export interface AllAction extends Action {
    name: string;
    age: number;
}
export interface Dispatch {
    (action: T): T
}
let dispatchNameAction: Dispatch = null;
let dispatchAgeAction: Dispatch = null;
let dispatchAllAction: Dispatch = null;
dispatchNameAction = dispatchAgeAction;
dispatchAgeAction = dispatchNameAction;
dispatchAllAction = dispatchNameAction;
dispatchAllAction = dispatchAgeAction;
```

4. 级联中间件

4.1 store\index.tsx

src\store\index.tsx

```
import { createStore, Store, AnyAction, Middleware, MiddlewareAPI, StoreEnhancer, StoreEnhancerStoreCreator, applyMiddleware } from '../redux';
import reducer from './reducers';
import { CombinedState } from './reducers';
//const store: Store = createStore(reducer, { counter1: { number: 0 }, counter2: { number: 0 } });
let logger: Middleware = (api: MiddlewareAPI) => (next: any) => (action: any) => {
    console.log(api.getState());
    next(action);
    console.log(api.getState());
    return action;
};
+let thunk: Middleware = (api: MiddlewareAPI) => (next: any) => (action: any) => {
+    if (typeof action == 'function') {
+        return action(api.dispatch, api.getState);
+    }
+    return next(action);
+};
+function isPromise(obj: any) {
+    return !!obj && (typeof obj === 'object' || typeof obj === 'function') && typeof obj.then === 'function';
+}
+let promise: Middleware = (api: MiddlewareAPI) => (next: any) => (action: any) => {
+    return isPromise(action.payload)
+        ? action.payload
+            .then((result: any) => api.dispatch({ ...action, payload: result }))
+            .catch((error: any) => {
+                api.dispatch({ ...action, payload: error, error: true });
+                return Promise.reject(error);
+            })
+        : next(action);
+};
+let storeEnhancer: StoreEnhancer = applyMiddleware(thunk, promise, logger);
let storeEnhancerStoreCreator: StoreEnhancerStoreCreator = storeEnhancer(createStore);
let store: Store = storeEnhancerStoreCreator(reducer);
export default store;
```

4.2 actions\counter1.tsx

src\store\actions\counter1.tsx

```
import * as types from '../action-types';
+import { AnyAction, Dispatch } from '../../redux';
const actions = {
    increment1(): AnyAction {
        return { type: types.INCREMENT1 };
    },
+    increment1Async() {
+        return function (dispatch: Dispatch) {
+            setTimeout(() => {
+                dispatch({ type: types.INCREMENT1 });
+            }, 1000);
+        }
+    },
+    increment1Promise() {
+        return {
+            type: types.INCREMENT1,
+            payload: new Promise((resolve: any, reject: any) => {
+                setTimeout(() => {
+                    let result = Math.random();
+                    if (result > .5) {
+                        resolve(result);
+                    } else {
+                        reject(result);
+                    }
+                }, 1000);
+            })
+        }
+    },
    decrement1(): AnyAction {
        return { type: types.DECREMENT1 };
    }
}
export default actions;
```

4.3 Counter1.tsx

src\components\Counter1.tsx

```
import React, { Component } from 'react';
import actions from '../store/actions/counter1';
import { CombinedState } from '../store/reducers';
import { Counter1State } from '../store/reducers/counter1';
import { connect } from '../react-redux';
+import { ActionCreatorsMapObject, AnyAction } from '../redux';
type Props = Counter1State & typeof actions;
class Counter1 extends Component {
    render() {
        let { number, increment1, decrement1 } = this.props;
        return (

                {number}
                +
+                异步+1
+                promise异步+1

        )
    }
}
let mapStateToProps = (state: CombinedState): Counter1State => state.counter1;
+export default connect(
    mapStateToProps,
    actions
)(Counter1)
```

4.4 connect.tsx

src\react-redux\connect.tsx

```
import React from "react";
import { bindActionCreators, Unsubscribe, AnyAction, ActionCreatorsMapObject } from "../redux";
import ReactReduxContext from "./context";
import { CombinedState } from '../store/reducers';
import { ContextValue } from './types';
interface MapStateToProps {
    (state: CombinedState): any;
}
interface MapDispatchToProps {
    [method: string]: void
}
+export default function (mapStateToProps: MapStateToProps, mapDispatchToProps: ActionCreatorsMapObject) {
    return function wrapWithConnect(WrappedComponent: React.ComponentType) {
        return class extends React.Component {
            static contextType = ReactReduxContext;
            unsubscribe: Unsubscribe;
            constructor(props: any, context: ContextValue) {
                super(props);
                this.state = mapStateToProps(context.store.getState());
            }
            componentDidMount() {
                this.unsubscribe = this.context.store.subscribe(() =>
                    this.setState(mapStateToProps(this.context.store.getState()))
                );
            }
            shouldComponentUpdate() {
                if (this.state
                    return false;
                }
                return true;
            }
            componentWillUnmount() {
                this.unsubscribe();
            }
            render() {
+                let actions = bindActionCreators>(
                    mapDispatchToProps,
                    this.context.store.dispatch
                );
                return ;
            }
        };
    };
}
```

4.5 bindActionCreators.tsx

src\redux\bindActionCreators.tsx

```
import { Dispatch, AnyAction } from './';
+export interface ActionCreator {
    (...args: any[]): A
}
+export interface ActionCreatorsMapObject {
    [key: string]: ActionCreator
}
+export default function bindActionCreators>(
    actionCreators: M,
    dispatch: Dispatch
): M
+export default function bindActionCreators>(
    actionCreators: M,
+    dispatch: Dispatch
): M {
+    const boundActionCreators: ActionCreatorsMapObject = {};
    for (const key in actionCreators) {
        const actionCreator = actionCreators[key]
        if (typeof actionCreator
            boundActionCreators[key] = bindActionCreator(actionCreator, dispatch)
        }
    }
    return boundActionCreators as M;
}
+function bindActionCreator(
    actionCreator: ActionCreator,
+    dispatch: Dispatch
) {
    return function (this: any, ...args: any[]) {
        return dispatch(actionCreator.apply(this, args));
    }
}
```

4.6 redux\types.tsx

src\redux\types.tsx

```
+export interface Dispatch {
    (action: T): T
}
```