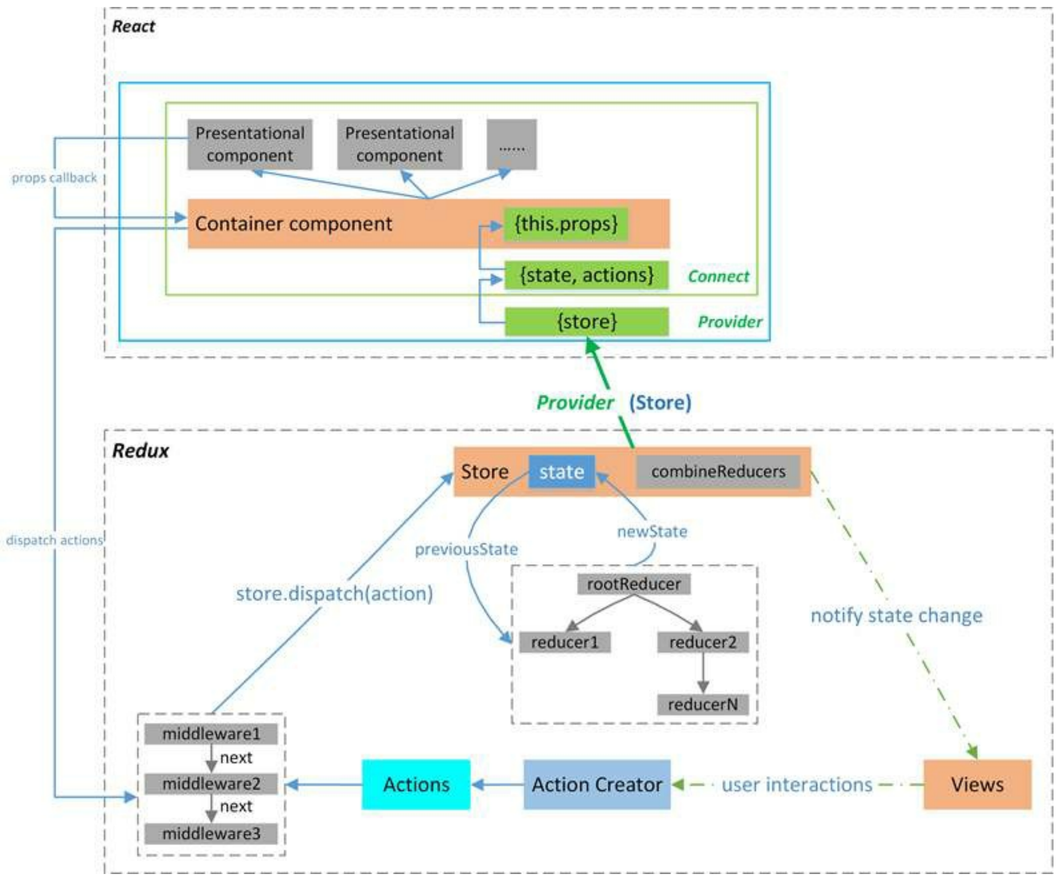
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=71 sentences=184, words=1193

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，所以这时候我们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.js

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

2.2 实现异步

src\store\index.js

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

3. 单个日志中间件

3.1 src\store\logger.js

src\store\logger.js

```
function logger({getState,dispatch}){
   return function(next){
     return function(action){
         console.log('prev state',getState());
         next(action);
         console.log('next state',getState());
         return action;
     }
   }
 }
 export default logger;
```

3.2 redux\applyMiddleware.js

src\redux\applyMiddleware.js

```
function applyMiddleware(logger){
    return function(createStore){
        return function(reducer, preloadedState){
            let store = createStore(reducer, preloadedState);
            dispatch = logger(store)(store.dispatch);
            return {
                ...store,
                dispatch
            };
        }
    }
}
export default applyMiddleware;
```

3.3 redux\index.js

src\redux\index.js

```
export {default as createStore} from './createStore'
export {default as bindActionCreators} from './bindActionCreators';
export {default as combineReducers} from './combineReducers';
+export {default as applyMiddleware} from './applyMiddleware';
```

3.4 store\index.js

src\store\index.js

```
import { createStore,applyMiddleware } from '../redux';
import reducer from './reducers';
import logger from './logger';
let store = applyMiddleware(logger)(createStore)(reducer);
export default store;
```

3.5 createStore.js

src\redux\createStore.js

```
const createStore = (reducer, preloadedState, enhancer) => {
+  if (typeof enhancer !== 'undefined') {
+    return enhancer(createStore)(reducer,preloadedState);
+  }
  let state=preloadedState;
  let listeners = [];
  function getState() {
    return state;
  }
  function dispatch(action) {
    state = reducer(state, action);
    listeners.forEach(l => l());
    return action;
  }
  function subscribe(listener) {
    listeners.push(listener);
    return () => {
      listeners = listeners.filter(l => l !== listener);
    }
  }
  dispatch({ type: '@@REDUX/INIT' });
  return {
    getState,
    dispatch,
    subscribe
  }
}
export default createStore;
```

4. 级联中间件

4.1 compose

- 如果一个函数需要经过多个函数处理才能得到最终值，这个时候可以把中间过程的函数合并一个函数**4.1.1 compose.js #** src\redux\compose.js
- compose (https://gitee.com/zhufengpeixun/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 function(args){
        for(let i=funcs.length-1;i>=0;i--){
            args=funcs[i](args);
        }
        return args;
    }
}

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

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

** 4.1.2 链式调用 #**

```
function compose(...funcs){
    return funcs.reduce((a,b)=>(...args)=>a(b(...args)));
}
let promise = (next)=>action=>{
    console.log('promise');
    next(action);
};
let thunk = (next)=>action=>{
    console.log('thunk');
    next(action);
};
let logger = (next)=>action=>{
    console.log('logger');
    next(action);
};

let chain = [promise,thunk,logger];
let composed = compose(...chain)
let dispatch = ()=>{
    console.log('原始的dispatch');
}
let newDispatch = composed(dispatch);
newDispatch({type:"add"});
```

4.2 applyMiddleware

src\redux\applyMiddleware.js

```
import compose from './compose';
function applyMiddleware(...middlewares) {
    return function (createStore) {
        return function (reducer,preloadedState) {
            let store = createStore(reducer,preloadedState);
            let dispatch;
            let middlewareAPI = {
                getState: store.getState,
                dispatch: (action) => dispatch(action)
            }
            let chain = middlewares.map(middleware => middleware(middlewareAPI));
            dispatch = compose(...chain)(store.dispatch);
            return {
                ...store,
                dispatch
            }
        }
    }
}
export default applyMiddleware;
```

```
let dispatch;
let middlewareAPI = {
    dispatch:(action)=>dispatch(action)
}
dispatch = (action)=>{console.log('action',action);}
middlewareAPI.dispatch({type:'ADD'});

let a;
let b=a;
a = 1;
console.log(b);
```

4.3 redux\index.js

src\redux\index.js

```
export {default as createStore} from './createStore';
export {default as bindActionCreators} from './bindActionCreators';
export {default as combineReducers} from './combineReducers';
export {default as applyMiddleware} from './applyMiddleware';
+export {default as compose} from './compose';
```

4.4 redux-logger\index.js

src\redux-logger\index.js

- redux-logger (https://gitee.com/zhufengpeixun/redux-logger/blob/master/src/index.js)

```
export default  (api) => (next) => (action) => {
  console.log(api.getState());
  next(action);
  console.log(api.getState());
  return action;
};
```

4.5 redux-promise\index.js

src\redux-promise\index.js

- redux-promise (https://gitee.com/zhufengpeixun/redux-promise/blob/master/src/index.js)

```
 function promise({getState,dispatch}){
    return function(next){
      return function(action){
          if(action.then&& typeof action.then==='function'){
            action.then(dispatch).catch(dispatch);
          }else if(action.payload && typeof action.payload.then==='function'){
            action.payload
            .then(result => dispatch({ ...action, payload: result }))
            .catch(error => {
               dispatch({ ...action, payload: error, error: true });
               return Promise.reject(error);
             })
          }else{
             next(action);
          }
      }
    }
  }
  export default promise;
```

4.6 redux-thunk\index.js

src\redux-thunk\index.js

- redux-thunk (https://gitee.com/zhufengpeixun/redux-thunk/blob/master/src/index.js)

```
export default ({ dispatch, getState }) => (next) => (action) => {
  if (typeof action === 'function') {
      return action(dispatch, getState);
  }
  return next(action);
};
```

4.7 actions\counter1.js

src\store\actions\counter1.js

```
import * as types from '../action-types';

const actions = {
    add1() {
        return { type: types.ADD1 };
    },
    minus1() {
        return { type: types.MINUS1 };
    },
+    thunkAdd1() {
+        return function (dispatch, getState) {
+            setTimeout(function () {
+                dispatch({ type: types.ADD1 });
+            }, 2000);
+        }
+    },
+    promiseAdd1() {
+        return {
+            type: types.ADD1,
+            payload: new Promise((resolve, reject) => {
+                setTimeout(() => {
+                    let result = Math.random();
+                    if (result > .5) {
+                        resolve(result);
+                    } else {
+                       reject(result);
+                    }
+                }, 1000);
+            })
+        }
+    },
+   promiseAdd2() {
+       return new Promise((resolve, reject) => {
+           setTimeout(() => {
+               resolve({ type: types.ADD1});
+           }, 1000);
+       });
    }
}
export default actions;
```

4.8 store\index.js

src\store\index.js

```
import { createStore, applyMiddleware } from '../redux';
import reducer from './reducers';
+import logger from '../redux-logger';
+import promise from '../redux-promise';
+import thunk from '../redux-thunk';
+let store = applyMiddleware(promise,thunk,logger)(createStore)(combinedReducer);
export default store;
```

4.9 Counter1.js

src\components\Counter1.js

```
import React, { Component } from 'react';
import actions from '../store/actions/counter1';
import { connect } from '../react-redux';
class Counter1 extends Component {
    render() {
        let { number, add1,addThunk1,addPromise1,addPromise2 } = this.props;
        return (

                {number}
                +
+               thunk+1
+               promise+1
+               promise+2

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