Redux使用及原理解析

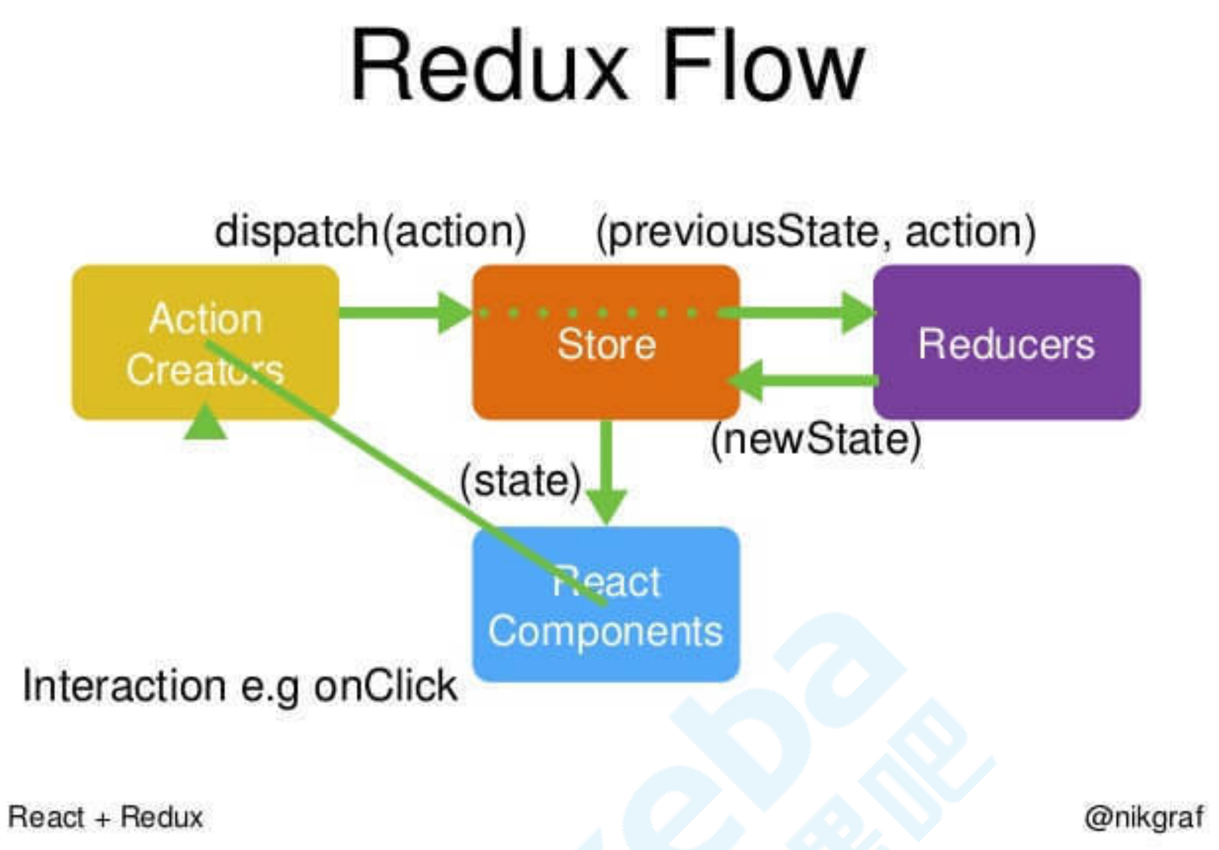
课堂目标

1. 掌握redux
2. 掌握redux中间件
3. 实现redux、react-redux及其中间件原理

资源

1. redux
2. redux github
3. react-redux

起步



Reducer

什么是reducer

reducer 就是一个纯函数，接收旧的 state 和 action，返回新的 state。

```
;(previousState, action) => newState
```

之所以将这样的函数称之为 reducer，是因为这种函数与被传入 `Array.prototype.reduce(reducer, ?initialValue)` 里的回调函数属于相同的类型。保持 reducer 纯净非常重要。**永远不要在** reducer 里做这些操作：

- 修改传入参数；

- 执行有副作用的操作，如 API 请求和路由跳转；
- 调用非纯函数，如 `Date.now()` 或 `Math.random()`。

什么是reduce

此例来自https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) =>
accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer));
// expected output: 10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5));
// expected output: 15
```

思考：有如下函数， 想要顺序输出1 2 3，如何处理。

```
function f1() {
  console.log("f1");
}
function f2() {
  console.log("f2");
}
function f3() {
  console.log("f3");
}
```

方法1：

```
f1();f2();f3();
```

方法2：

```
f3(f2(f1()));
```

方法3：

```
function compose(...funcs) {
  if (funcs.length === 0) {
    return arg => arg
  }
  if (funcs.length === 1) {
    return funcs[0]
  }
  return funcs.reduce((a, b) => (...args) =>
a(b(...args)))
}
compose(
  f1,
  f2,
  f3,
)();
```

Redux 上手

Redux是JavaScript应用的状态容器。它保证程序行为一致性且易于测试。

安装redux

`npm install redux --save`

redux上手

redux~~较难上手，是因为上来就有太多的概念需要学习~~，用一个累加器举例

1. 需要一个store来存储数据
2. store里的reducer初始化state并定义state修改规则
3. 通过dispatch一个action来提交对数据的修改
4. action提交到reducer函数里，根据传入的action的type，返回新的state

创建store，src/store/ReduxStore.js

```
import {createStore} from 'redux'

const counterReducer = (state = 0, action) => {
    switch (action.type) {
      case 'add':
        return state + 1
      case 'minus':
        return state - 1
      default:
        return state
    }
  }
const store = createStore(counterReducer)

export default store
```

创建ReduxPage

```
import React, { Component } from "react";
import store from "../store/ReduxStore";

export default class ReduxPage extends Component {
  componentDidMount() {
    store.subscribe(() => {
      console.log("subscribe");
      this.forceUpdate();
      //this.setState({});
    });
  }
  add = () => {
    store.dispatch({ type: "add" });
  };
  minus = () => {
    store.dispatch({ type: "minus" });
  };
  stayStatic = () => {
    store.dispatch({ type: "others" });
  };
  render() {
    console.log("store", store);
    return (
      <div>
        <h1>ReduxPage</h1>
        <p>{store.getState()}</p>
        <button onClick={this.add}>add</button>
        <button onClick={this.minus}>minus</button>
        <button onClick=
{this.stayStatic}>static</button>
      </div>
    );
  }
}
```

> 如果点击按钮不能更新，因为没有订阅(subscribe)状态变更

还可以在src/index.js的render里订阅状态变更

```
import store from './store/ReduxStore'
const render = ()=>{

  ReactDom.render(
    <App/>,
    document.querySelector('#root')
  )
}
render()

store.subscribe(render)
```

检查点

1. createStore 创建store
2. reducer 初始化、修改状态函数
3. getState 获取状态值
4. dispatch 提交更新
5. subscribe 变更订阅

react-redux

每次都重新调用render和getState太low了，想用更react的方式来写，需要react-redux的支持

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

提供了两个api

1. Provider 为后代组件提供store
2. connect 为组件提供数据和变更方法

全局提供store，index.js

```
import React from 'react'
import ReactDom from 'react-dom'
import App from './App'
import store from './store/ReactReduxStore'

import { Provider } from 'react-redux'
ReactDom.render(
  <Provider store={store}>
    <App/>
  </Provider>,
  document.querySelector('#root')
)
```

获取状态数据，ReactReduxPage.js

```
import React, { Component } from "react";
import { connect } from "react-redux";
```

```
class ReactReduxPage extends Component {
  render() {
    const { num, add, minus, asyAdd } = this.props;
    return (
      <div>
        <h1>ReactReduxPage</h1>
        <p>{num}</p>
        <button onClick={add}>add</button>
        <button onClick={minus}>minus</button>
        {/* <button onClick={asyAdd}>asyAdd</button>
*/}
      </div>
    );
  }
}

const mapStateToProps = state => {
  return {
    num: state,
  };
};
const mapDispatchToProps = {
  add: () => {
    return { type: "add" };
  },
  minus: () => {
    return { type: "minus" };
  },
  //Actions must be plain objects. Use custom
middleware for async actions.
  // asyAdd: () => {
    //   //console.log("omh", dispatch);
    //   setTimeout(() => {
```

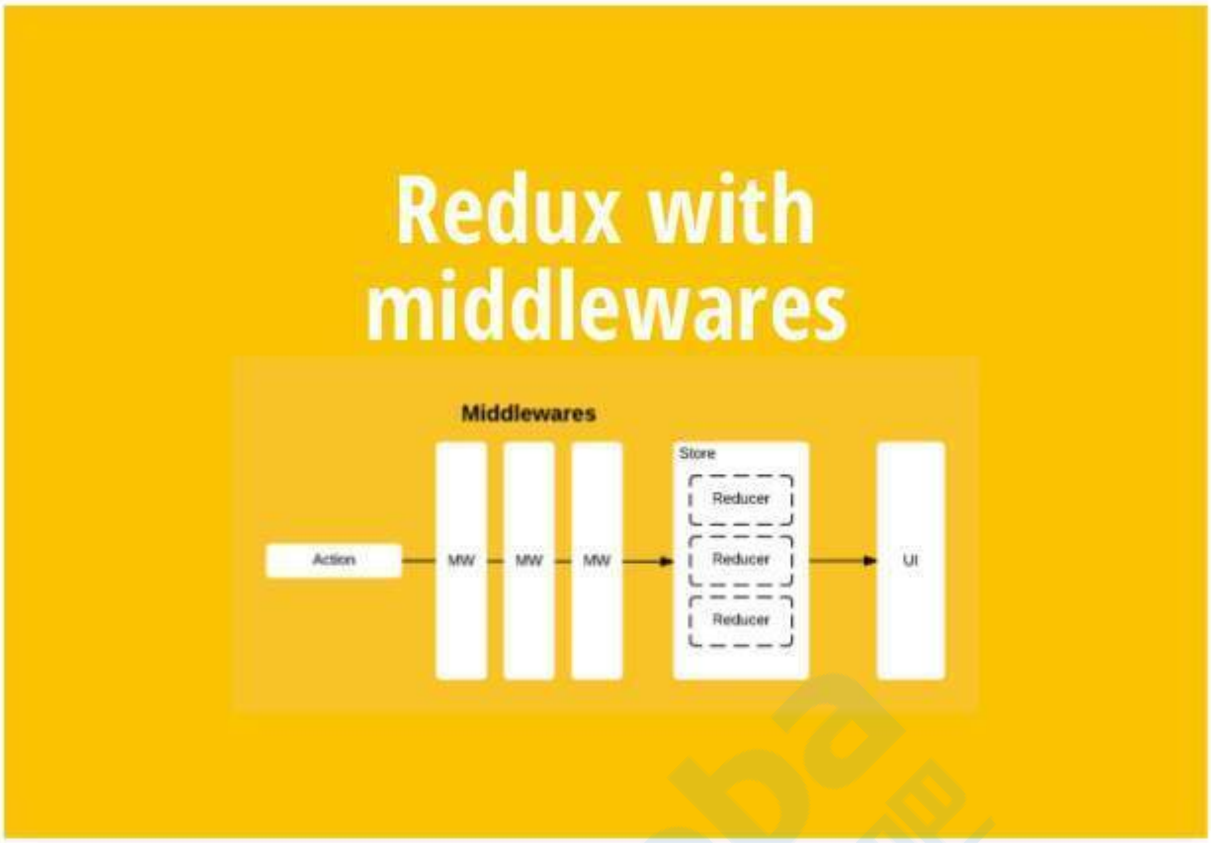
```
    //      return { type: "add" };
    //    }, 1000);
    // },
};

export default connect(
  mapStateToProps, //状态映射 mapStateToProps
  mapDispatchToProps, //派发事件映射
)(ReactReduxPage);
```

> connect中的参数：state映射和事件映射

异步

Redux只是个纯粹的状态管理器，默认只支持同步，实现异步任务比如延迟，网络请求，需要中间件的支持，比如我们试用最简单的redux-thunk和redux-logger

```
npm install redux-thunk redux-logger --save
```



应用中间件，store.js

```
import { createStore, applyMiddleware } from
"redux";
import logger from "redux-logger";
import thunk from "redux-thunk";
import counterReducer from './counterReducer'

const store = createStore(counterReducer,
applyMiddleware(logger, thunk));
```

使用异步操作时的变化，ReactReduxPage.js

```
const mapDispatchToProps = {
  add: () => {
    return { type: "add" };
  },
  minus: () => {
    return { type: "minus" };
  },
  asyAdd: () => dispatch => {
    setTimeout(() => {
      // 异步结束后，手动执行dispatch
        dispatch({ type: "add" });
    }, 1000);
  },
};
```

*如果出现错误类似/babel-preset-react-app/node_modules/@babel/runtime/helpers/slicedToArray' at webpackMissingModule '，就**npm add @babel/runtime** *

代码抽取

抽离reducer和action:

1. 抽离action

action/reactReduxPage.js

```
export const add = () => {
  return { type: "add" };
};
export const minus = () => {
  return { type: "minus" };
};
export const asyAdd = () => dispatch => {
  setTimeout(() => {
    dispatch({ type: "add" });
  }, 1000);
};
```

//对应的ReactReduxPage文件直接引用

```
import React, { Component } from "react";
import { connect } from "react-redux";
import { add, minus, asyAdd } from
"../action/reactReduxPage";//此处直接引用

class ReactReduxPage extends Component {
  render() {
    const { num, add, minus, asyAdd } = this.props;
    return (
      <div>
        <h1>ReactReduxPage</h1>
        <p>{num}</p>
        <button onClick={add}>add</button>
        <button onClick={minus}>minus</button>
        <button onClick={asyAdd}>asyAdd</button>
      </div>
    );
  }
}
```

```
const mapStateToProps = state => {
  return {
    num: state,
  };
};
const mapDispatchToProps = {//此处
  add,
  minus,
  asyAdd,
};

export default connect(
  mapStateToProps, //状态映射 mapStateToProps
  mapDispatchToProps, //派发事件映射
)(ReactReduxPage);
```

2. 抽离reducer

store/counterReducer.js

```
export const counterReducer = (state = 0, action) =>
{
  switch (action.type) {
    case "add":
      console.log("stat", state);
      return state + 1;
    case "minus":
      return state - 1;
    default:
      return state;
  }
};
```

store/index.js也是直接引用counterReducer即可

```
import { createStore, applyMiddleware } from
"redux";
import logger from "redux-logger";
import thunk from "redux-thunk";
import { counterReducer } from "./counterReducer";

const store = createStore(counterReducer,
applyMiddleware(logger, thunk));
```

模块化

combineReducers， store.js

```
import { combineReducers } from "redux";

const store = createStore(
combineReducers({counter: counterReducer}),
applyMiddleware(logger, thunk)
);
```

ReactReduxPage.js

```
import React, { Component } from "react";
import { connect } from "react-redux";
import { add, minus, asyAdd } from
"../action/reactReduxPage";

class ReactReduxPage extends Component {
render() {
 console.log("props", this.props);
 const { counter, add, minus, asyAdd } =
this.props;
 const { num } = counter;

 return (
   <div>
     <h1>ReactReduxPage</h1>
     <p>{num}</p>
     <button onClick={add}>add</button>
     <button onClick={minus}>minus</button>
     <button onClick={asyAdd}>asyAdd</button>
   </div>
 );
}
}

const mapStateToProps = state => {
return {
 counter: state,
};
};
const mapDispatchToProps = {
add,
minus,
asyAdd,
};

export default connect(
mapStateToProps, //状态映射 mapStateToProps
mapDispatchToProps, //派发事件映射
)(ReactReduxPage);
```

Redux拓展

redux原理

核心实现

- 存储状态state
- 获取状态getState
- 更新状态dispatch
- 变更订阅subscribe

kRedux.js

```
export function createStore(reducer, enhancer){
  if (enhancer) {
    return enhancer(createStore)(reducer)
  }
    // 保存状态
  let currentState = undefined;
    // 回调函数
  let currentListeners = [];

  function getState(){
    return currentState
  }
  function subscribe(listener){
    currentListeners.push(listener)
  }
  function dispatch(action){
    currentState = reducer(currentState, action)
    currentListeners.forEach(v=>v())
    return action
  }
  dispatch({type:'@IMOOC/KKB-REDUX'})
  return { getState, subscribe, dispatch}
}
```

store/MyReduxStore.js

```
const counterReducer = (state = 0, action) => {
  switch (action.type) {
    case "add":
      return state + 1;
    case "minus":
      return state - 1;
    default:
      return state;
  }
};
const store = createStore(counterReducer);

export default store;
```

页面可以用原来的ReduxPage.js测试以上代码

中间件实现

核心任务是实现函数序列执行

//把下面加入kRedux.js

```
export function applyMiddleware(...middlewares){
   // 返回强化以后函数
  return createStore => (...args) => {
    const store = createStore(...args)
    let dispatch = store.dispatch

    const midApi = {
      getState:store.getState,
      dispatch:(...args)=>dispatch(...args)
    }
    // 使中间件可以获取状态值、派发action
    const middlewareChain =
middlewares.map(middleware => middleware(midApi))
    // compose可以middlewareChain函数数组合并成一个函数
    dispatch = compose(...middlewareChain)
(store.dispatch)
    return {
      ...store,
      dispatch
    }
  }
}
export function compose(...funcs) {
  if (funcs.length === 0) {
    return arg => arg
  }
  if (funcs.length === 1) {
    return funcs[0]
  }
  return funcs.reduce((a, b) => (...args) =>
a(b(...args)))
}
```

把下面加入MyReduxStore.js

```
function logger() {
  return dispatch => action => {
    // 中间件任务
    action.type && console.log(action.type + "执行
了！");
    return dispatch(action);
  };
}

const store = createStore(counterReducer,
applyMiddleware(logger));
```

redux-thunk原理

thunk增加了处理函数型action的能力，把下面加入MyReduxStore.js

```
function thunk({ getState }) {
  return dispatch => action => {
    if (typeof action === "function") {
      return action(dispatch, getState);
    } else {
      return dispatch(action);
    }
  };
}
const store = createStore(counterReducer,
applyMiddleware(logger,thunk));
```

测试代码

```
import React, { Component } from "react";
import store from "../store/MyReduxStore";

export default class MyReduxPage extends Component {
  componentDidMount() {
    store.subscribe(() => {
      this.forceUpdate();
      // this.setState({});
    });
  }
  asyAddHandle = () => {
    store.dispatch(dispatch => {
      setTimeout(() => {
        dispatch({ type: "add" });
      }, 1000);
    });
  };
  render() {
    return (
      <div>
        <h1>MyReduxPage</h1>
        <p>counter:{store.getState()}</p>
        <button onClick={() => store.dispatch({
type: "add" })}>add</button>
        <button onClick={() => store.dispatch({
type: "minus" })}>minus</button>
        <button onClick=
{this.asyAddHandle}>asyAdd</button>
      </div>
    );
  }
}
```

react-redux原理

核心任务：

- 实现一个高阶函数工厂connect，可以根据传入状态映射规则函数和派发器映射规则函数映射需要的属性，可以处理变更检测和刷新任务
- 实现一个Provider组件可以传递store

//可用下面代码测试

Src/index.js

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import { Provider } from "./kReact-redux";
import store from "./store/myReactReduxStore";

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

MyReactReduxPage.js

```
import React, { Component } from "react";
import { connect } from "../kReact-redux";
```

```
class MyReactReduxPage extends Component {
  render() {
    const { counter, add } = this.props;
    return (
      <div>
        <h1>MyReactReduxPage</h1>
        <p>counter:{counter}</p>
        <button onClick={add}>add</button>
      </div>
    );
  }
}

export default connect(
  state => {
    return { counter: state };
  },
  {
    add: () => {
      return { type: "add" };
    },
  },
)(MyReactReduxPage);
```

实现react-redux

实现kReact-redux.js

```
import React from 'react'
import PropTypes from 'prop-types'
import {bindActionCreators} from './kkb-redux'

export const connect = (mapStateToProps =
state=>state, mapDispatchToProps = {}) =>
(WrapComponent) => {
  return class ConnectComponent extends
React.Component{
        // class组件中声明静态contextTypes可以获取上下文
Context
    static contextTypes = {
      store: PropTypes.object
    }
    constructor(props, context){
      super(props, context)
      this.state = {
        props:{}
      }
    }
    componentDidMount(){
      const {store} = this.context
      store.subscribe(()=>this.update())
      this.update()
    }
    update(){
      const {store} = this.context
            // state => ({num: state.counter})
      const stateProps =
mapStateToProps(store.getState())
            // {add:()=>({type:'add'})}
            // {add:(...args) =>
dispatch(creator(...args))}
```

```
      const dispatchProps =
bindActionCreators(mapDispatchToProps,
store.dispatch)
      this.setState({
        props:{
          ...this.state.props, // 之前的值
          ...stateProps, // num: state.counter
          ...dispatchProps // add:(...args) =>
dispatch(creator(...args))
        }
      })
    }
    render(){
      return <WrapComponent {...this.state.props}>
</WrapComponent>
    }
  }
}

export class Provider extends React.Component{
  static childContextTypes = {
    store: PropTypes.object
  }
  getChildContext() {
    return { store: this.store }
  }
  constructor(props, context) {
    super(props, context)
    this.store = props.store
  }
  render() {
    return this.props.children
  }
```

```
}
//实现bindActionCreators
function bindActionCreator(creator, dispatch){
  return (...args) => dispatch(creator(...args))
}
export function
bindActionCreators(creators,dispatch){
    // {add:()=>({type:'add'})}
    // {add:(...args) => dispatch(creator(...args))}
  return Object.keys(creators).reduce((ret,item)=>{
    ret[item] =
bindActionCreator(creators[item],dispatch)
    return ret
  },{})
}
```

用hooks实现:

```
import React, { useContext, useState, useEffect }
from "react";

const Context = React.createContext();

export function Provider({ store, children }) {
  return <Context.Provider value={store}>{children}
</Context.Provider>;
}

export const connect = (
  mapStateToProps = state => state,
  mapDispatchToProps = {},
) => Cmp => props => {
  const store = useContext(Context);
  const getMoreProps = () => {
    const stateProps =
mapStateToProps(store.getState());
    const dispatchProps = bindActionCreators(
      mapDispatchToProps,
      store.dispatch,
    );
    return { ...stateProps, ...dispatchProps };
  };
  const [moreProps, setMoreProps] =
useState(getMoreProps());
  useEffect(() => {
    store.subscribe(() => {
      setMoreProps({ ...moreProps, ...getMoreProps()
});
    });
  }, []);
  return <Cmp {...props} {...moreProps} />;
};

//这两个函数可在redux源码中查看，不懂的地方课console.log打
印查看，建议打印下actionCreators和key
function bindActionCreator(creator, dispatch) {
  return (...args) => dispatch(creator(...args));
}
function bindActionCreators(actionCreators,
dispatch) {
  const boundActionCreators = {};
  for (const key in actionCreators) {
    boundActionCreators[key] =
bindActionCreator(actionCreators[key], dispatch);
  }
  return boundActionCreators;
```

```
}
```

回顾

作业

1. 理解这节课内容
2. 整合自己目前在学习项目，加入redux

下节课内容

react-router基本使用、动态路由、路由嵌套、路由守卫，源码实现BrowserRouter、Route、Link。