React全家桶01-redux

课堂目标

1. 掌握redux、常用中间件的使用及实现

资源

1. 课堂代码github地址： https://github.com/bubucuo/kkb-react
2. 课堂代码码云地址： https://gitee.com/bubucuo/kkb-react
3. vscode相关插件配置详看预习视频（自定义snippets在github）
4. redux
5. redux github

知识点

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， 聚合成一个函数，并把第一个函数的返回值传递给下一个函数，如何处理。

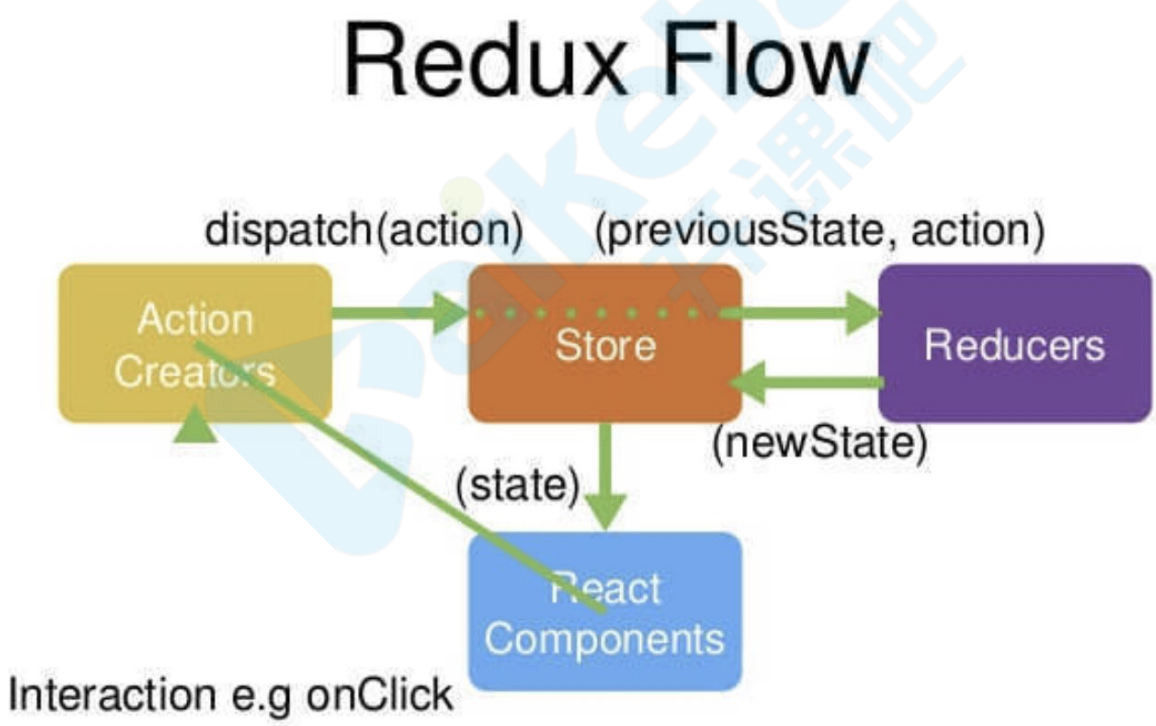
```
function f1(arg) {
  console.log("f1", arg);
  return arg;
}
function f2(arg) {
  console.log("f2", arg);
  return arg;
}
function f3(arg) {
  console.log("f3", arg);
  return arg;
}
```

方法：

```
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)))
}
console.log(compose(f1, f2, f3)("omg"));
```

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

`yarn add redux`

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/index.js

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

function countReducer(state = 0, action) {
  switch (action.type) {
    case "ADD":
      return state + 1;
    case "MINUS":
      return state - 1;
    default:
      return state;
  }
}

const store = createStore(countReducer);

export default store;
```

创建ReduxPage

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

export default class ReduxPage extends Component {
  componentDidMount() {
    store.subscribe(() => {
      this.forceUpdate();
    });
  }
  add = () => {
    store.dispatch({type: "ADD"});
  };
  minus = () => {
    store.dispatch({type: "MINUS"});
  };
  render() {
    console.log("store", store); //sy-log
    return (
      <div>
        <h3>ReduxPage</h3>
        <p>{store.getState()}</p>
```

```
        <button onClick={this.add}>add</button>
        <button onClick={this.minus}>minus</button>
      </div>
    );
  }
}
```

> 如果点击按钮不能更新，查看是否订阅(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/'
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

Redux拓展

核心实现

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

AkRedux.js

```
export default function createStore(reducer, enhancer) {
  if (enhancer) {
    return enhancer(createStore)(reducer);
  }
  let currentState;
  let currentListeners = [];
```

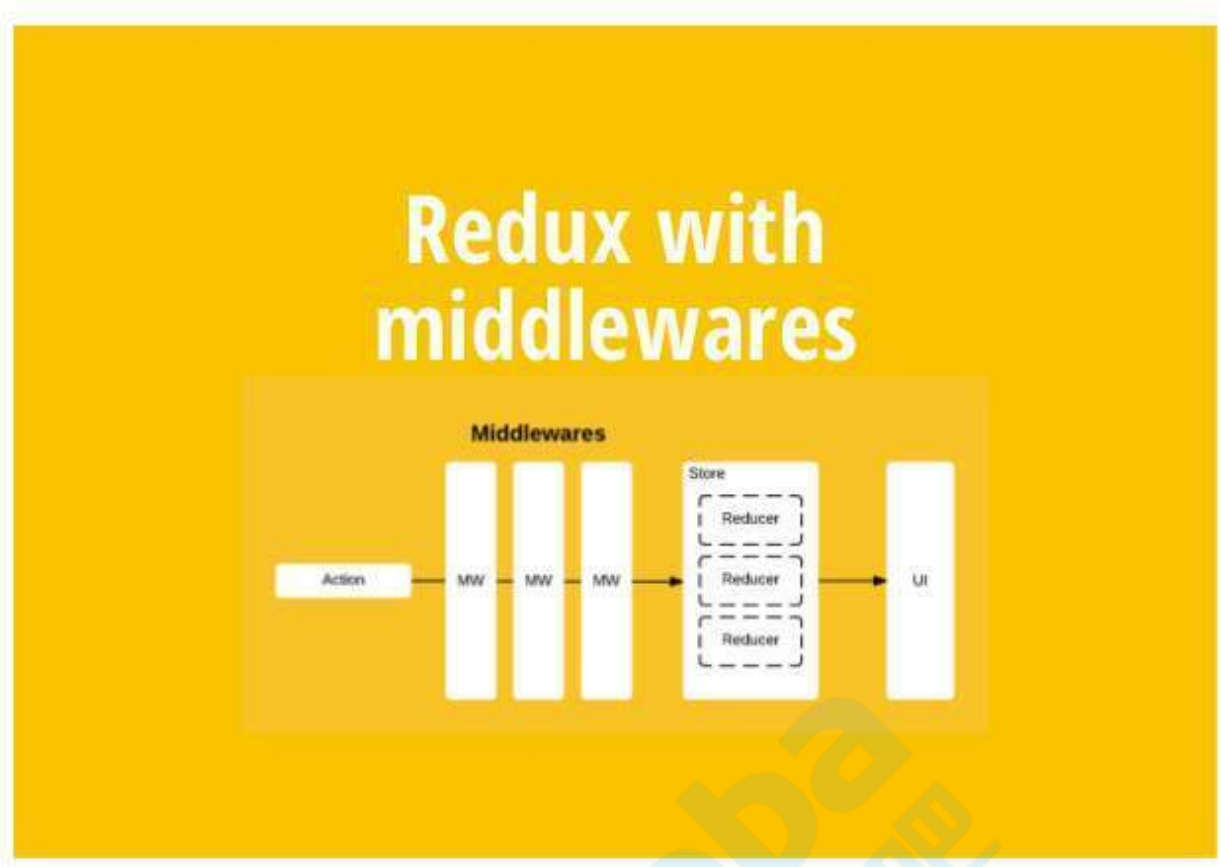
```
  function getState() {
    return currentState;
  }

  function dispatch(action) {
    currentState = reducer(currentState, action);
    currentListeners.forEach(listener => listener());
    return action;
  }

  function subscribe(listener) {
    currentListeners.push(listener);
    return () => {
      currentListeners = [];
    };
  }

  dispatch({type: "KKBREDUX/OOOO"});

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

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，比如我们使用最简单的redux-thunk和redux-logger 。

中间件就是一个函数，对`store.dispatch`方法进行改造，在发出 Action 和执行 Reducer 这两步之间，添加了其他功能。

```
yarn add redux-thunk redux-logger
```



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(thunk, logger));
```

使用异步操作时的变化，ReactReduxPage.js

```
asyAdd = () => {
   store.dispatch((dispatch, getState) => {
     setTimeout(() => {
       // console.log("now ", getState()); //sy-log
       dispatch({type: "ADD", payload: 1});
     }, 1000);
   });
 };
```

中间件实现

核心任务是实现函数序列执行。

applyMiddleware

```
export default function applyMiddleware(...middlewares) {
  return createStore => reducer => {
    const store = createStore(reducer);
    let dispatch = store.dispatch;

    const midApi = {
      getState: store.getState,
      dispatch: (action, ...args) => dispatch(action, ...args)
    };
    const middlewareChain = middlewares.map(middleware => middleware(midApi));

    dispatch = compose(...middlewareChain)(store.dispatch);
    return {
      ...store,
      // 加强版的dispatch
      dispatch
    };
  };
}

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)));
}
```

redux-logger原理

logger可打印redux state变更日志。

```
function logger({getState}) {
  return next => action => {
    console.log("====================================");
    console.log(action.type + "执行了! "); //sy-log

    const prevState = getState();
    console.log("prev state", prevState); //sy-log

    const returnValue = next(action);
    const nextState = getState();
    console.log("next state", nextState); //sy-log
```

```
    console.log("====================================");
    return returnValue;
  };
}
```

redux-thunk原理

thunk增加了处理函数型action的能力。

```
function thunk({dispatch, getState}) {
  return next => action => {
    if (typeof action === "function") {
      return action(dispatch, getState);
    }
    return next(action);
  };
}
```

redux-promise

简版：

```
function promise({dispatch}) {
  return next => action => {
    return isPromise(action) ? action.then(dispatch) : next(action);
  };
}
```

完整版：

```
import isPromise from 'is-promise';
import { isFSA } from 'flux-standard-action';

export default function promiseMiddleware({ dispatch }) {
  return next => action => {
    if (!isFSA(action)) {
      return isPromise(action) ? action.then(dispatch) : next(action);
    }

    return isPromise(action.payload)
      ? action.payload
          .then(result => dispatch({ ...action, payload: result }))
          .catch(error => {
            dispatch({ ...action, payload: error, error: true });
            return Promise.reject(error);
```

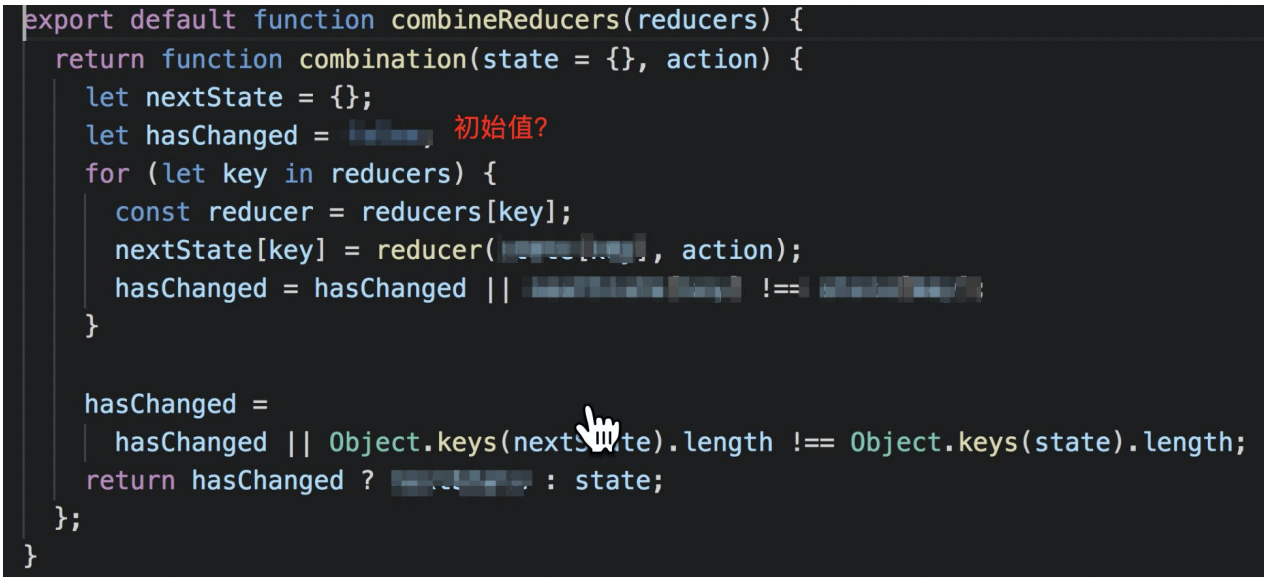
```
          })
        : next(action);
  };
}
```

```
  promiseMinus = () => {
    store.dispatch(
      Promise.resolve({
        type: "MINUS",
        payload: 100
      })
    );
  };
```

回顾

作业

1. 实现combineReducers，阅读源码，补充以下代码，调试并运行，**提交自己运行之后的代码截图!**



下节课内容

React全家桶02：react-redux使用及实现。