拓展和作业

1. 基于useReducer的方式实现异步action

```
// reducer增加了loading状态
function fruitReducer(state, action) {
  switch (action.type) {
    case "init":
      return { ...state, list: action.payload };
    case "add":
      return { ...state, list: [...state.list, action.payload] };
    case "loading_start":
      return { ...state, loading: true };
    case "loading_end":
      return { ...state, loading: false };
    default:
      return state;
  }
}

// 判断对象是否是Promise
function isPromise(obj) {
  return (
    !!obj &&
    (typeof obj === "object" || typeof obj === "function") &&
    typeof obj.then === "function"
  );
}

// mock一个异步方法
async function asyncFetch(p) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(p);
    }, 1000);
  });
}

// 对dispatch函数进行封装，使其支持处理异步action
function wrapDispatch(dispatch) {
  return function(action) {
    if (isPromise(action.payload)) {
      dispatch({ type: "loading_start" });
      action.payload.then(v => {
        dispatch({ type: action.type, payload: v });
        dispatch({ type: "loading_end" });
      });
    } else {
      dispatch(action);
    }
```

```
  };
}

export default function HookTest() {
  // 修改reducer初始化方式
  const [{ list: fruits, loading }, originDispatch] = useReducer(fruitReducer, {
    list: [],
    loading: false
  });
  // 包装dispatch
  const dispatch = wrapDispatch(originDispatch);

  useEffect(() => {
    console.log("get fruits");

    // setTimeout(() => {
    //   //   setFruits(["草莓", "香蕉"]);
    //   dispatch({ type: "init", payload: ["草莓", "香蕉"] });
    // }, 1000);

    // 派发动作，payload是Promise
    dispatch({ type: "init", payload: asyncFetch(["草莓", "香蕉"]) });
  }, []);

  return (
    <Context.Provider value={{ fruits, dispatch }}>
      <div>
        {/*加载状态处理*/}
        {loading ? (
          <div>数据加载中...</div>
        ) : (
          <FruitList fruits={fruits} setFruit={setFruit} />
        )}
      </div>
    </Context.Provider>
  );
}
```

2. 尝试实现Form（布局、提交）、FormItem（错误信息）、Input（前缀图标）

FormItem

```
class FormItem extends Component {
  render() {
    return (
      <div>
        {this.props.children}
        {this.props.help && (
          <p
            style={{
              color: this.props.validateStatus === "error" ? "red" : "green"
```

```
            }}
          >
            {this.props.help}
          </p>
        )}
      </div>
    );
  }
}
```

使用

```
class KFormTest extends React.Component {

  render() {
    // 获取表单项错误
    const { getFieldDec, isFieldTouched, getFieldError } = this.props;
    const unameError = isFieldTouched("username") && getFieldError("username");
    return (
      <div>
        <FormItem
          validateStatus="error"
          help={unameError || ""}>
          {getFieldDec("username", {
            rules: [{ required: true, message: "Please input your username!" }]
          })(<input type="text" />)}
        </FormItem>
      </div>
    );
  }
}
```

定义isFieldTouched、getFieldError，kFormCreate

```
function kFormCreate(Comp) {
  return class extends React.Component {
    // 焦点处理，错误获取等
    handleFocus = e => {
      const field = e.target.name;
      this.setState({
        [field + "Focus"]: true
      });
    };

    isFieldTouched = field => {
      return !!this.state[field + "Focus"];
    };

    getFieldError = field => {
      return this.state[field + "Message"];
    };
```

```
    getFieldDec = (field, option) => {
      this.options[field] = option;
      return InputComp => (
        <div>
          {React.cloneElement(InputComp, {
            onFocus: this.handleFocus //焦点处理
          })}
        </div>
      );
    };

    render() {
      return (
        <Comp
          validateFields={this.validateFields}
          isFieldTouched={this.isFieldTouched}
          getFieldError={this.getFieldError}
        />
      );
    }
  };
}
```

KInput

```
class KInput extends React.Component {
  render() {
    const {prefix, ...rest} = this.props;
    return (
      <div>
        {prefix}
        <input {...rest}/>
      </div>
    );
  }
}
```

使用

```
import {Icon} from 'antd';

{getFieldDec("username", {
    rules: [{ required: true, message: "Please input your username!" }]
})(<KInput type="text" prefix={<Icon type="user" />} />)}

{getFieldDec("password", {
    rules: [{ required: true, message: "Please input your Password!" }]
})(<KInput type="password" prefix={<Icon type="lock" />}/>)}
```

资源

1. redux
2. react-redux
3. react-router

知识点

使用redux

1. 安装：`npm i redux react-redux -S`
2. 创建store实例

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

function fruitReducer(state = initial, action) {}

const store = createStore(fruitReducer)

export default store;
```

3. 注册该实例 Provider

```
import { Provider } from "react-redux";
import store from "./store";
import ReduxTest from "./ReduxTest";

<Provider store={store}><ReduxTest /></Provider>
```

4. 组件中使用状态 connect

```
import {connect} from 'react-redux';

// connect返回一个高阶组件，可以把redux状态作为属性注入到包装组件
export default connect(state => ({
  loading: state.loading,
  fruits: state.list
}))(function ReduxTest({ loading, fruits }) {
  if(loading) ...
  fruits.map(...)
});

// dispatch方法通过connect方式注入
const FruitAdd = connect()(function({ dispatch }) {
  dispatch({type:'add',payload:input})
});
```

5. 代码优化

1. 提取action creator

```
export const init = (payload) => ({
    type: 'init',
    payload
})
export const add = (payload) => ({
    type: 'add',
    payload
})
export const loadingStart = () => ({
    type: 'loading_start',
})
export const loadingEnd = () => ({
    type: 'loading_end',
})
```

2. 映射为dispatch函数到属性

```
import { init, loadingStart, loadingEnd } from './store';

const mapStateToProps = state => ({ loading: state.loading, fruits: state.list
});
const mapDispatchToProps = { init, loadingStart, loadingEnd };
function ReduxTestContainer({ loading, fruits, loadingStart, loadingEnd, init
}) {...}
const ReduxTest = connect(mapStateToProps, mapDispatchToProps)
(ReduxTestContainer);
export default ReduxTest;
```

6. 异步

1. 安装redux-thunk：`npm i redux-thunk -S`
2. 应用中间件，store.js

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

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

3. 定义异步动作

```
export const asyncFetch = (payload) => {
  return dispatch => {
    dispatch({type:'loading_start'});
    setTimeout(() => {
        dispatch({type:'loading_end'});
        dispatch({type:'init', payload: ["草莓", "香蕉"]});
    }, 1000);
  };
};
```

4. 使用

```
import { asyncFetch } from "./store";

const mapDispatchToProps = {
    asyncFetch
//   init,
//   loadingStart,
//   loadingEnd
};
function ReduxTestContainer({
  asyncFetch
//   loadingStart,
//   loadingEnd,
//   init
}) {}
```

7. 模块化

```
// 把action和reducer移至fruit.redux.js

// store/index.js
import { combineReducers } from "redux";
import fruitReducer from './fruit.redux';

const store = createStore(
  combineReducers({ fruit: fruitReducer }),
  applyMiddleware(logger, thunk)
);

// ReduxTest.js
import { asyncFetch } from "./store/fruit.redux";

const mapStateToProps = state => ({
  loading: state.fruit.loading,
  fruits: state.fruit.list
});
```

react-router

1. 安装：`npm install --save react-router-dom`
2. 设定路由器

```
<BrowserRouter>content...</BrowserRouter>
```

3. 导航

```
<Link to="/">水果列表</Link>|
<Link to="/add">添加水果</Link>
```

4. 路由

```
{/* 根路由要添加exact，render可以实现条件渲染 */}
        <Route
          exact
          path="/"
          render={props =>
            loading ? <div>数据加载中...</div> : <FruitList fruits={fruits} />
          }
        />
        <Route path="/add" component={FruitAdd} />
```

5. 传参

```
<Link to={`/detail/${f}`}>{f}</Link>

<Route path="/detail/:fruit" component={Detail} />
```

6. 嵌套

   Route组件嵌套在其他页面组件中就产生了嵌套关系

7. 404

```
{/* 添加Switch表示仅匹配一个 */}
<Switch>
    <Route exact path="/" render={props => <Redirect to="/list" />} />
    <Route component={() => <h3>页面不存在</h3>}></Route>
</Switch>
```

8. 路由守卫

创建高阶组件包装Route使其具有权限判断功能

```
function PrivateRoute({ component: Component, isLogin, ...rest }) {
  // 结构props为component和rest
  // rest为传递给Route的属性
  return (
    <Route
      {...rest}
      render={
        // 执行登录判断逻辑从而动态生成组件
        props =>
          isLogin ? (
            <Component {...props} />
          ) : (
            <Redirect
              to={{
                pathname: "/login",
                state: { redirect: props.location.pathname } // 重定向地址
              }}
            />
          )
      }
    />
  );
}
```

使用

```
<PrivateRoute path="/add" component={FruitAdd} />
```

redux原理

```
export function createStore(reducer, enhancer){
    if (enhancer) {
        return enhancer(createStore)(reducer)
    }
    let currentState = {}
    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}
}

export function applyMiddleware(...middlewares){
    return createStore=>(...args)=>{
        const store = createStore(...args)
        let dispatch = store.dispatch

        const midApi = {
            getState:store.getState,
            dispatch:(...args)=>dispatch(...args)
        }
        const middlewareChain = middlewares.map(middleware=>middleware(midApi))
        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((left,right) => (...args)=>right(left(...args)))
}
function bindActionCreator(creator, dispatch){
    return (...args) => dispatch(creator(...args))
}
export function bindActionCreators(creators,dispatch){
    return Object.keys(creators).reduce((ret,item)=>{
        ret[item] = bindActionCreator(creators[item],dispatch)
        return ret
    },{})
}
```

react-redux原理

```
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{
        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
            const stateProps = mapStateToProps(store.getState())
            const dispatchProps = bindActionCreators(mapDispatchToProps,
store.dispatch)
            this.setState({
                props:{
                    ...this.state.props,
                    ...stateProps,
                    ...dispatchProps
                }
            })
        }
        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
    }
}
```

redux-thunk原理

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