**Redux于navigation 的组合使用**

**需要引入的文件:**

**1. redux**

**2. react-redux**

**3. react-navigation-redux-helpers**

**配置navigation + redux:**

1. **配置navigation导航:**低版本将reduxifyNavigator改为createReduxContainer,切调换middleware中参数的位置

**import** React **from** 'react';

**import** {createStackNavigator, createSwitchNavigator} **from** 'react-navigation';

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

**import** {createReactNavigationReduxMiddleware, createReduxContainer} **from** 'react-navigation-redux-helpers';

**export** **const** rootCom **=** 'Init';*//设置根路由*

**export** **const** RootNavigator **=** createSwitchNavigator({

...

});

*/\*\* \* 1.初始化react-navigation与redux的中间件， \* 该方法的一个很大的作用就是为createReduxContainer的key设置actionSubscribers(行为订阅者) \* 设置订阅者@https://github.com/react-navigation/react-navigation-redux-helpers/blob/master/src/middleware.js#L29 \* 检测订阅者是否存在@https://github.com/react-navigation/react-navigation-redux-helpers/blob/master/src/middleware.js#L97 \* @type {Middleware} \*/*

**export** **const** middleware **=** createReactNavigationReduxMiddleware(

state **=>** state.nav,

'root'

);

*/\*\* \* 2.将根导航器组件传递给 createReduxContainer 函数, \* 并返回一个将navigation state 和 dispatch 函数作为 props的新组件； \* 注意：要在createReactNavigationReduxMiddleware之后执行 \*/*

**const** AppWithNavigationState **=** createReduxContainer(RootNavigator, 'root');

*/\*\* \* State到Props的映射关系 \* @param state \*/*

**const** mapStateToProps **=** state **=>** ({

state: state.nav,*//v2*

});

*/\*\* \* 3.连接 React 组件与 Redux store \*/*

**export** **default** connect(mapStateToProps)(AppWithNavigationState);

1. **配置Reducer**

**import** {combineReducers} **from** 'redux'

**import** theme **from** './theme'

**import** {rootCom, RootNavigator} **from** '../navigator/AppNavigators';

*//1.指定默认state*

**const** navState **=** RootNavigator.router.getStateForAction(RootNavigator.router.getActionForPathAndParams(rootCom));

*/\*\* \* 2.创建自己的 navigation reducer， \*/*

**const** navReducer **=** (state **=** navState, action) **=>** {

**const** nextState **=** RootNavigator.router.getStateForAction(action, state);

*// 如果`nextState`为null或未定义，只需返回原始`state`*

**return** nextState **||** state;

};

*/\*\* \* 3.合并reducer \* @type {Reducer<any> | Reducer<any, AnyAction>} \*/*

**const** index **=** combineReducers({

nav: navReducer,

theme: theme,

});

**export** **default** index;

1. **配置store**

**import** {applyMiddleware, createStore} **from** 'redux'

**import** thunk **from** 'redux-thunk'

**import** reducers **from** '../reducer'

**import** {middleware} **from** '../navigator/AppNavigators'

**const** middlewares **=** [

middleware,

];

*/\*\* \* 创建store \*/*

**export** **default** createStore(reducers, applyMiddleware(...middlewares));

1. **在组件中的使用**

**import** React, {Component} **from** 'react';

**import** {Provider} **from** 'react-redux';

**import** AppNavigator **from** './navigator/AppNavigators';

**import** store **from** './store'

type Props **=** {};

**export** **default** **class** App **extends** Component**<**Props**>** {

render() {

*/\*\* \* 将store传递给App框架 \*/*

**return** **<**Provider store**=**{store}**>**

**<**AppNavigator**/>**

**<**/Provider>  }

}

**5.** [**使用react-navigaton+redux**](http://www.devio.org/2019/03/30/react-native-redux-react-navigation/#%E4%BD%BF%E7%94%A8react-navigatonredux) **🡨🡨** [**点击跳转**](http://www.devio.org/2019/03/30/react-native-redux-react-navigation/#%E4%BD%BF%E7%94%A8react-navigatonredux)

**6. 中间件的编写以及使用:**

*/\*\*  
 \** ***@Author:Training*** *\** ***@param*** *store  
 \** ***@returns*** *{function(\*): Function}  
 \** ***@desc:编写redux中间件*** *\*/*const logger = store=>next=>action=>{  
 if (typeof action ==='function'){ //判断 action是否是一个function  
 console.log('action is a function !')  
 }else{  
 console.log('dispatching:' ,action ) //打印redux状态改变前的action  
 }  
 const result = next(action); //继续执行action  
 console.log('nextState',store.getState()); //打印状态改变后的值  
};  
  
const middlewares = [  
 middleware,  
 logger,  
 thunk  
];

**7. 安卓物理返回键的配置,--- home页面:**

*/\*\*  
 \** ***@Author:Training*** *\** ***@format*** *\** ***@flow*** *\*/*import React, {*Component*} from 'react';  
import {Platform, StyleSheet, Text, View} from 'react-native';  
import navigatorUtil from "../Util/navigatorUtil"  
import DynamicTabNavigation from '../navigator/DynamicTabNavigation'  
import {connect} from "react-redux";  
import {BackHandler} from "react-native";  
import {NavigationActions} from "react-navigation";  
  
type Props = {};  
class App extends *Component*<Props> {  
 */\*\*  
 \** ***@Author:Training*** *\** ***@Desc:对于Android物理返回键的适配支持*** *\** ***@Params:*** *\*/* componentDidMount() {  
 BackHandler.addEventListener("hardwareBackPress", this.onBackPress);  
 }  
 componentWillUnmount() {  
 BackHandler.removeEventListener("hardwareBackPress", this.onBackPress);  
 }  
 */\*\* \* 处理 Android 中的物理返回键 \* https://reactnavigation.org/docs/en/redux-integration.html#handling-the-hardware-back-button-in-android \* @returns {boolean} \*/* onBackPress = () => {  
 const {dispatch, nav} = this.props;  
 //if (nav.index === 0) {  
 if (nav.routes[1].index === 0) {//如果RootNavigator中的MainNavigator的index为0，则不处理返回事件  
 return false;  
 }  
 dispatch(NavigationActions.back());  
 return true;  
 };  
 render() {  
 navigatorUtil.navigation = this.props.navigation;  
 return <DynamicTabNavigation />;  
 }  
}  
const mapStateToProps = state=>(  
 {  
 nav:state.nav  
 }  
)  
export default connect(mapStateToProps)(App);