## **安装 React Redux**

Redux 默认并不包含 [React 绑定库](https://github.com/reactjs/react-redux" \t "https://www.redux.org.cn/docs/basics/_blank)，需要单独安装。

## npm install --save react-redux

## **容器组件和展示组件**

|  | **展示组件** | **容器组件** |
| --- | --- | --- |
| **作用** | 描述如何展现（骨架、样式） | 描述如何运行（数据获取、状态更新） |
| **直接使用 Redux** | 否 | 是 |
| **数据来源** | props | 监听 Redux state |
| **数据修改** | 从 props 调用回调函数 | 向 Redux 派发 actions |
| **调用方式** | 手动 | 通常由 React Redux 生成 |

技术上讲你可以直接使用 store.subscribe() 来编写容器组件。但不建议这么做，而是使用 React Redux 的 connect() 方法来生成

容器组件和展示组件在职责上面有区别，代码上面无区别，如果不了解可以问百度

**React Redux完整示例**

**动作类型**

export const HelloActionType = 'HelloActionType'

**创建动作函数**

export function createHelloAction(isShow){

return { type : HelloActionType , isShow }

}

**根据action创建Reducer**

function helloReducer(state = true, action){

switch (action.type){

case HelloActionType:

return action.isShow;

default:

return state;

}

}

**生成根Reducer**

const todoApp = combineReducers({

showHello: helloReducer

})

**使用根Reducer创建store**

let store = createStore(todoApp)

**创建应用程序入口文件index.js**

import { Provider } from 'react-redux'

render(

<Provider store={store}>

<HelloContain />

</Provider>,

document.body

)

其中Provider作为应用的根，接收store，这样在子组件中便可访问store

**创建Hello的容器组件HelloContain**

const mapStateToProps = (state, ownProps) => { // ownProps为当前组件的props

    return {

        isShow: state.showHello

    }

}

const mapDispatchToProps = (dispatch, ownProps) => {

    return {

        setShowHello: (isShow) => { dispatch(createHelloAction(isShow)) }

    }

}

const HelloContain = connect(

    mapStateToProps, // 关于state

    mapDispatchToProps // 关于dispatch

)(Hello)

export default HelloContain;

这里使用connect来创建容器组件，connect第一个参数为state的委托，第二个参数为dispatch的委托，两个个委托均返回展示组件Hello的props

**创建hello展示组件**

export default class Hello extends React.Component {

    render() {

        return (<div>

            <h1>{this.props.isShow && "Hello"}</h1>

            <button onClick={() => this.props.setShowHello(!this.props.isShow)}>

                {this.props.isShow ? "隐藏" : "显示"}

            </button>

        </div>)

    }

}

**上述应用数据流为：**

1. let store = createStore(todoApp)创建store，store生成初始化state
2. <Provider store={store}>将store放入上下文中，每个组件都可以访问
3. 使用connect创建容器组件HelloContain，connect调用[store.subscribe(listener)](https://www.redux.org.cn/docs/api/Store.html" \l "subscribe)为HelloContain安装监听器
4. 容器组件HelloContain监听store获取state，调用其mapStateToProps 和mapDispatchToProps 委托，生成展示组件Hello的props
5. 展示组件利用props展示数据，展示组件点击隐藏按钮，触发props的setShowHello委托，setShowHello由HelloContain定义
6. HelloContain的setShowHello触发dispatch，传入一个动作
7. helloReducer处理的这个动作，并返回新的state个store
8. 所有监听器被触发，返回步骤4

**connect() 参数**

connect([mapStateToProps], [mapDispatchToProps], [mergeProps], [options])(MyComponent)

前面两个参数在示例中已介绍，这里介绍后面两个参数

**mergeProps**

mergeProps(stateProps, dispatchProps, ownProps):props

该参数将前面两个参数的结构进行封装，生成新的props

**Options**

配置connect的行为，为一个js对象，其属性有

pure ：如果为 true，connect 将执行 shouldComponentUpdate 并且浅对比 mergeProps 的结果，避免不必要的更新默认值为 true

withRef：如果为 true，connect会保存一个对被包装组件实例的引用，该引用通过 getWrappedInstance() 方法获得。默认值为 false