React全家桶

# 前言：

2020-7-6开始想系统学习下react，当时看了下官网的博客例子，井字游戏；

后面断断续续。

中间在公司学习了一段时间的react官方文档；

2021-2月份学习webpack

2021-4-4 ~ 2021-5-26终于学完了尚硅谷的张天禹的react课程，完结撒花❀❀❀

# React基础

## React概述

React是什么?



前端基本步骤：

1. 发送请求获取数据
2. 处理数据（过滤、整理格式等）
3. 操作DOM呈现页面

React只做了这第三步

为什么要学呢？

1. 原生js操作dom繁琐、效率低。
2. 使用js直接操作dom，浏览器会进行大量的重绘重排。
3. 原生js没有组件化编码方案，代码复用率低。

React的特点：

1. 采用组件化模式、声明式编程，提高开发效率及组件复用率。
2. 在React Native中可以使用React语法进行移动端开发。
3. 使用虚拟Dom+及其优秀的Diff算法，尽量减少与真实DOM的交互。

React的核心内容：

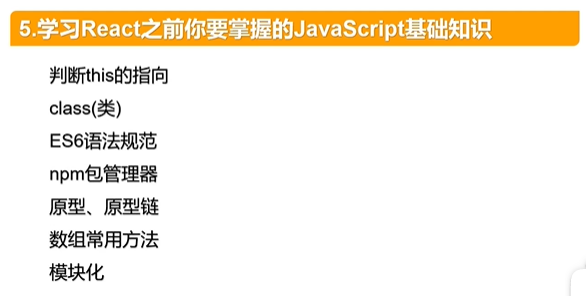
1. 组件化  
   jsx语法，es6 calss得熟悉
2. 核心属性

Props state refs

1. 生命周期

新老生命周期对比

1. Diff算法



## Hello react

虚拟DOM的两种创建方式：

1. React.creatElemnet(type, props, children)
2. JSX

## JSX语法

1. 定义虚拟DOM, 不要用引号，直接赋值就可以了
2. 标签中引入js表达式时要{}
3. 样式问题：用className, 而不是class

style={{key: value}}

1. 只有一个根标签。
2. 标签必须合并。
3. 组件标签首字母要大写，否则就默认为html标签了。

# 面向组件式编程

## 概述与分类

### 函数式组件

适合简单点的功能，没有this

### 类式组件概述

Class MyComponent extends React.Component {

//一定得调用render方法，全局所有的其它方法都是为它的return服务，因为它才是且只呈现于界面上的东东

Render() {

Return <h1>最后的结果</h1>

}

}

## 类式组件的三大核心特性

### State

State是一个对象，{key1:value1, key2:value2, … ,}

自定义方法往往不是被实例对象直接调用的，而是通过赋值语句给onClick, onBlur等，后期点击的时候再触发，这时函数内this会丢失，那么：

1. 通过bind(this)
2. 使用箭头函数来自定义函数，这样在this内找不到this，会往上一层。

This.state.key = value, 不能直接赋值，react不认的，得this.setState({});

### Props

1. Props不可改变
2. 传递值太多时，可以当作对象散开传递 <Demo {…obj} />
3. Props可以做类型、必填等限制；

Demo. propTpyes= {name: PropTpyes.string.isRequired}

Demo.defaultProps = {

sex: '男'

}

1. 函数式组件首字母大写，可以玩props, 但是不能玩State，refs, 这两个需要在实例对象上。

### refs

[refs官方文档](https://zh-hans.reactjs.org/docs/refs-and-the-dom.html#gatsby-focus-wrapper)

1. 字符串方式  
   简洁好用，奈何官方说有效率问题，不推荐使用了
2. 回调函数形式

个人感觉蛮好的

1. React.createRef();

官网推荐的，并没有第二种好用。

### 疑问

1. 为什么要设计props了，还要设计state呢？

直接让组件内接受到了props，直接改不就可以了吗？

1. 为什么直接修改state不起效果，一定要setState？这就是单向数据流吗？

作用是干吗呢？

## 事件处理

<button onClick={this.func} >

onBlur

1. 对原生的事件进行驼峰封装，  
   原因不明，可能是为了更好的兼容性吧；
2. 和原生js一样，可以通过e.target获取事件发生对象

## 非受控组件&受控组件

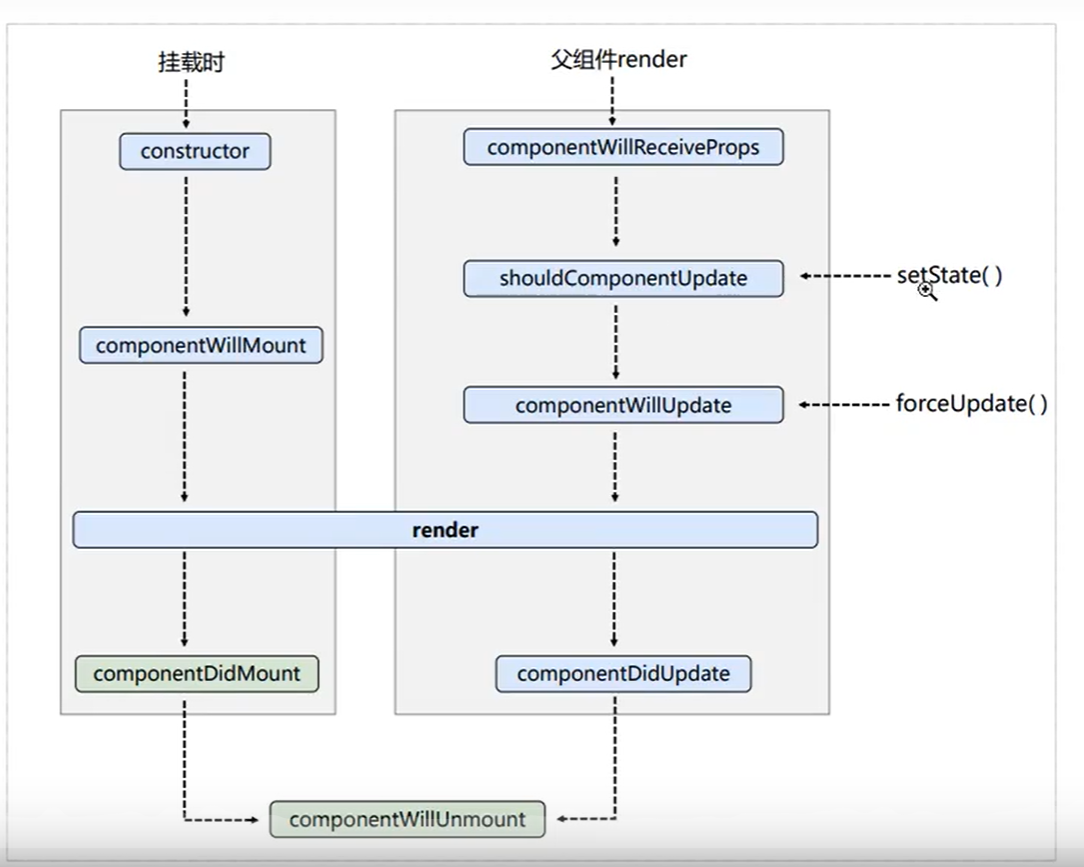
1. 非受控组件：现用现取；可以使用ref来实现

2． 受控组件：被state所管理

不过感觉挺麻烦 要写这么多代码呀

## 生命周期

### 旧版本



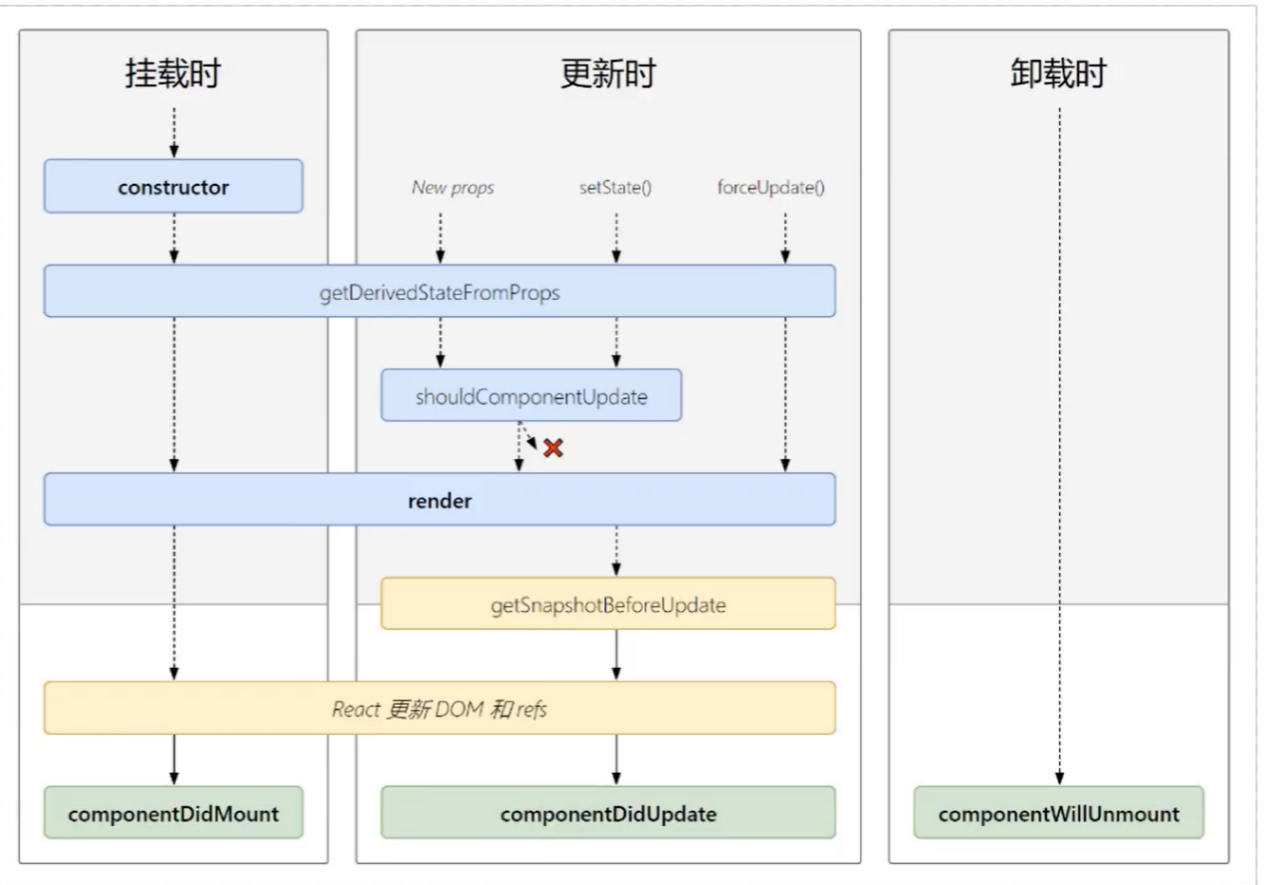
#### 初始化

#### 更新

组件被更新，存在三种情况：

1. 本身setSate了
2. 不常用的forceUpdate()
3. 父组件被重新render了

### 新版本



[官方对三个will类型改名的原因](https://zh-hans.reactjs.org/docs/react-component.html#unsafe_componentwillmount)

日后会推出异步渲染，可能会不安全，提前提醒大家干掉。

### 生命周期使用场景

只讨论新的生命周期：

页面初始化

|  |  |  |
| --- | --- | --- |
| 生命周期 | 使用场景 | 结论 |
| constructor | 1. 通过给 this.state 赋值对象来初始化内部 state。 2. 为事件处理函数绑定实例   [react官网文档-constructor](https://zh-hans.reactjs.org/docs/react-component.html#constructor) | 麻烦，能不写就不写 |
| getDerivedStateFromProps | 初始化那次应该没啥用, 白走一遍 |  |
| render | 必须得有，核心 | 不能setState |
| componentDidMount | 1. 自身发送远程请求并且setState | 掌握componentDidMount && render，一般足矣 |

页面更新

|  |  |  |
| --- | --- | --- |
| 生命周期 | 使用场景 | 结论 |
| getDerivedStateFromProps | 从父组件中获取props，初始化自身state  父组件发送远程请求，得到的结果作为props传递下来，必然会有异步延时，所以此场景在componentDidMount中setState行不通 | 1. 因为是static，所以不能this.setState 2. 得限制只是父组件props变化时候才return state 3. 非常难用 |
| shouldComponentUpdate | 判断是否要render |  |
| render | 必须得有，核心 | 不能setState |
| getSnapshotBeforeUpdate | 更新dom之前，获取dom的展示信息，不如滚动条高度之类的 |  |
| componentDidUpdate | 数据看板里面，父组件setTimeInterval调用接口，如果需要重绘echart之类的 |  |

## Diff算法

# 3. React脚手架

## Create-react-app

### 脚手架介绍

### 配置代理

#### Webpack

如何解决跨域，详见：

D:\code\js\_workpalace\JS\_Frame\1\_webpack\_study\5\_webpack详细配置\36\_devserver

 \* 方法1. 自己启动server, 服务端允许跨域

 \* 方法2：配置proxy

配置方法又分为两种：

1. Package.json中增加一行： “proxy”:”http://localhost:5000”
2. Webpack.config.js中增加:
3. devServer: {
4. port: 5000,
5. open: true,
6. host: 'localhost',
7. // 服务器代理，解决开发环境下的跨域问题
8. proxy: {
9. // 1. 先去掉app1的域名，让/app1/service开头的服务直接发给webpack-dev-server
10. // 2. 再由这里转给targe
11. '/app1/service': {
12. target: 'http://localhost:3006',
13. // changeOrigin: true,
14. // 发送请求时，请求路径重写，将/api/xx -> /xx
15. // patchRewrite: {
16. //     '^/service': ''
17. // }
18. },
19. }
20. }

#### Create-react-app

用的完全就算照搬 webpack-dev-server

方法二：新增一个文件src/setupProxy.js

## Umi

## Ant-design-pro

# Ajax异步请求

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）

AJAX 最大的优点是在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容。

## 原生js

1.创建一个XMLHttpRequest异步对象

2.设置请求方式和请求地址

3.接着，用send发送请求

4.监听状态变化

5.最后，接收返回的数据

## Jquery

[jQuery发送Ajax请求](https://blog.csdn.net/jinixin/article/details/80042763)

五花八门，通用的：

// $.ajax({

            //     type: "get",

            //     url: "https://so.csdn.net/api/v2/search?q=王力宏&t=userinfo",

            //     dataType: "json",

            //     success: function (data) {

            //         console.log(data);

            //         document.getElementById("myDiv").innerHTML = JSON.stringify(data);

            //     }

            // });

            // 2. 指定调用方法类型

            $.get(

                "https://so.csdn.net/api/v2/search?q=王力宏&t=userinfo",

                function (data) {

                    console.log(data);

                    document.getElementById("myDiv").innerHTML = JSON.stringify(data);

                }

            );

            // 3. 指定返回值

            // $.getJSON(

            //     "https://so.csdn.net/api/v2/search?q=王力宏&t=userinfo",

            //     function (data) {

            //         console.log(data);

            //         document.getElementById("myDiv").innerHTML = JSON.stringify(data);

            //     }

            // );

## Axios

[axios官方文档](http://www.axios-js.com/)

## Fetch

# 路由

## 如何制作菜单导航

### 多个页面

实现菜单导航，可以做多个html, 选中页面，对应菜单添加高亮样式：class=”active”

优点：原理简单，实现起来方便

缺点：页面会闪烁

但是如果菜单渲染是动态的，并且代码只有一份，其实也没什么关系

### Iframe

#### 基本用法

说起iframe，大家都会觉得很嫌弃或者很不愿接近，原因大概都是：

1. 能耗高，安全问题，
2. spider不喜欢它...
3. 也有（不少）同学内心独白就是：不！这很low很dirty，我才不想用惹！

但是我们必须的承认iframe之强大，很多时候我们都会（不得不）使用它，真的素又爱又恨呐-8-

[深入浅出iframe - rocky wong - 简书](https://www.jianshu.com/p/7ec986aa28a7)

[ifram 实现左侧菜单，右侧显示内容 - cnblog](https://www.cnblogs.com/wangzh1guo/p/10182558.html)

#### 应用在菜单上

优点：

1. 页面整体不会刷新，
2. 整洁，代码不重复
3. 简单好理解

缺点:

1. 地址栏url不变化，只能看菜单或者面包屑区分当前位置
2. 页面不好刷新

### SPA

Single page application

这里指的是：自己写状态控制页面组件的显示or隐藏

优点：

页面刷新方面，已经很完美了

缺点：

1. 如果写菜单，并不是那么简洁，还需要封装
2. 地址栏没有发生变化，也就会造成iframe同样的问题：没法刷新浏览器

### 什么是路由

路由就算一个key value映射

Key为path, value是组件或者处理函数

#### 后端路由

Value是一个fuction，用来处理客户端发送过来的请求

Nodejs代码为例：

Router.get(“/getStudents”, function(req, resp){…} )

#### 前端路由

Value是一个component, 用来渲染页面

<Router path=“/test” component={Test} />

工作过程：

当浏览器内url发生变化（可以是用户输入的，也可以是点击路由菜单，比如Link这种），就会触发组件更改并重新渲染；

## React-router

### 基本概念

Link && NavLink

BrowserRouter && HashRouter

原理: Link改变地址栏url，但是不发送网络请求，页面不跳转，是操作了window.history对象；地址栏一变，BrowserRouter就根据注册的路由变换组件。

不同点大比拼：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 写法不同 | 位置不同 | 传递参数不同 |
| 普通组件 | <Hello name=”sj” /> | 随意，哪儿都可以 |  |
| 路由组件 | <Router path=”/home” compotent={Hello} /> | BrowserRouter HashRouter | History  Location  match |

如何让 普通组件->路由组件？

使用高阶函数用法：withRouter(普通组件)

NavLink带了active样式，或者activeClassName="active"

### 路由组件如何传递参数

\* 1. params方式

\* 2. search

\* 3. state传递: 隐藏传递参数，不想体现下地址栏内

例子：Xxx/5react\_router/3router\_parameter.js

小知识：怎么实现无痕模式呢?

其实很简单的，操作window.history即可

或者开启replace，默认是push： <Link to=”” replace />

### 编程式路由

如果不写<Link />, 而是代码实现路由跳转：

则需要研究：

history:

push: ƒ push(path, state)

replace: ƒ replace(path, state)

## BroserRouter VS HashRouter

|  |  |  |
| --- | --- | --- |
|  | BroserRouter | HashRouter |
| 实现原理 | H5的history API, 不兼容IE9及以下版本 | URL的哈希值 |
| url表现形式不一样 |  | 路径中带#  Localhost:3000#/demo/test |
| 刷新后对state参数的影响 | 不丢失，因为state保存在history里面 | 丢了 |

## 使用脚手架实现路由

[Menu导航菜单\_antd](https://ant.design/components/menu-cn/)

# 6 UI组件库

## Antd

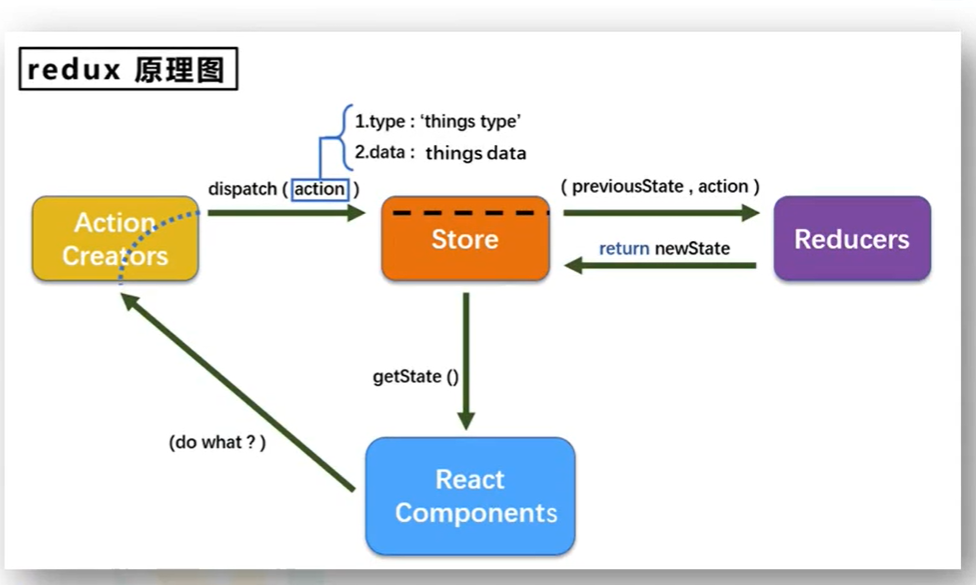
[ant design](https://ant.design/index-cn)

# 7 Redux

94小节 – 114小节 内容可真够多的

## 原生redux

* npm i redux



组件自身不在维护状态了，把state交给store去维护；

Store只是个转手掌柜，干活的是打工人reducer；

Action可以不写，直接放在组件内。

Action分为两种：

1. 同步action

Return 对象：  
export const plusAction = data => ({type:"plus", data:data\*1});

1. 异步action  
   return function:

export const plusAsyncAction1 = (data, time) => {

setTimeout(()=>{

store.dispatch(plusAction(data))

}, time);

};

### 疑问

如果我来设计redux，只有两个api:

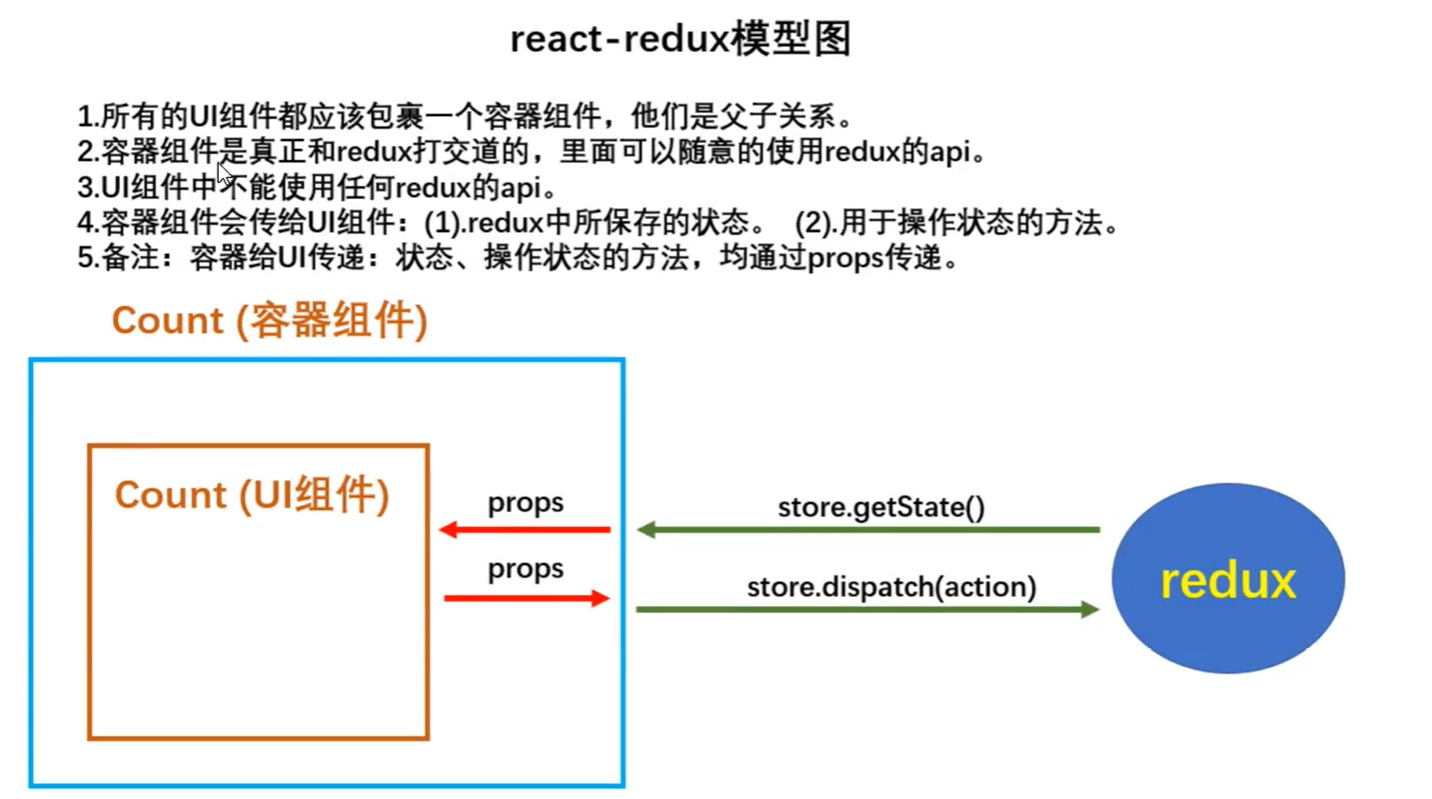
Store.getState(), Store.setState() ; 不久可以了吗？

为啥要把原始数据封装成action，可能还会在action里面发送请求之类的，

为啥要把数据操作过程封装成reducer呢？

## React-reduct

# npm i react-reduct



1. react-redux并不是完全重写redux，就好比react-app-rewired只是封装了webpack一样；react-redux是封装了redux，还是离不开redux本身的使用的
2. 它的目的应该是不想让redux散落在react的各处代码处，所以整出了一个容器组件概念：作为中间人把redux的状态，action方法作为属性传递下去

核心得理解容器组件的写法：

export default connect(

    state => ({

        persons: state.person, // 传递redux的state

        count: state.counter

    }),

    {plus, minus, plusAsync} // 传递redux的action方法s

)(Counter) // UI组件

对于redux的Reducer，容器组件也是接触不到的，reducer只和store打交道；

Redux的store怎么传递给容器组件呢，有点奇葩了，得在容器组件声明的时候传递, 可能是为了能够让react-redux在容器组件中支持

<Provider store={store}>

     <CounterContainer />

      <hr />

      <PersonContainer />

</Provider>

## Redux Vs React-redux

为什么不用redux，而要用react-redux呢？我实在是没想到为什么？

甚至不明白为什么要用redux？不用PubSub.subscribe() PubSub.publish()

|  |  |  |  |
| --- | --- | --- | --- |
|  | PubSub | Redux | React-redux |
| 基本用法 | PubSub.subscribe("users\_topic", (topic, data)=>{  this.setState({data});  });  PubSub.publish() | 1. Store中createStore(counter\_reducers) 2. 组件里面调用store.dispatch， action都可以省略 3. 当然也可以写action | 1. Redux该怎么写还是怎么写 2. Store和ui组件之间隔了一个容器组件； |
|  |  | componentDidMount() {  store.subscribe(() => {  this.setState({})  })  } | 不需要手动监听store变化 |
|  |  |  | 剩下的，没有感觉到任何有点，有的只是麻烦和复杂 |

## Redux开发者工具

如果组件特别多，而且大家协调的，redux可能会变的很难看，

那需要个开发者工具在浏览器内观察吗？

也挺好哦

<https://www.bilibili.com/video/BV1wy4y1D7JT?p=113>

# 8 Hooks

## 设计hook的动机

官网给出的三大理由

[Hooks简介 - 官网](https://react.docschina.org/docs/hooks-intro.html)

**在组件之间复用状态逻辑很难**

Hooks这么强悍吗？那真是太酷了….

*Hook 使你在无需修改组件结构的情况下复用状态逻辑。 这使得在组件间或社区内共享 Hook 变得更便捷。*

**复杂组件变得难以理解**

**难以理解的 class**

Class难以理解？不至于….

这理由比较牵强。

Hook 使你在非 class 的情况下可以使用更多的 React 特性。 从概念上讲，React 组件一直更像是函数。而 Hook 则拥抱了函数，

## 使用方法

三大钩子：

useState： 替代state

useEffect: 替代三个常用生命周期：componentDidMount componentWillUnmount componentDidUpdate

useRef: 替代React.createRef()

还能reducer？

当然为了不重复创建函数，对象；还有

Memo useCallback useMemo

# 9 专题

## Setstate

setState有两种方式

1. 对象式setState({}, [callback])
2. 函数式setState(function, [callback])

## 懒加载

懒的总结了。。。

## Context

React.Context可以实现祖孙组件之间数据通讯，

但是呢，我暂时不晓得跨文件怎么传递MyContext

步骤:

1. 创建  
   export const MyContext = React.createContext()

const {Provider, Consumer} = MyContext

1. 第一级组件声明要传递的区域 & 传递的值

<Provider value={name}>

<B />

</Provider>

1. 接收方式有两种：

方式一：

static contextType = MyContext

this.contxt

方式二:

再搞出一个方式二，应该是为了让函数组件也能用，因为函数组件用不了方式一的this

<Consumer>

{value => value}

</Consumer>

## RenderProps

可以 this.props.children渲染子组件，但是呢，这样不好传递参数吧？

所以呢，我们还可以使用：

<A render={value => <B name={value}>} />

在A里面渲染处B  
this.render(name)

## ErrorBoundary

为什么要定位为static函数呢？

Static getDerivedStateFromError(error) {

Return {error}

}

## 组件通信

1. 父传子Props
2. PubJs
3. Redux 或者 dva
4. React.createContext

## Form表单

## Drag and drop

这东东太有用了，工作中做BE-UI必须得玩的好

### Html positon

1. 定义div样式为absolute, left, top
2. 定义方法: onmousedown onmouseup onmousemove
3. move\_div.style.left = ndx + "px";

move\_div.style.top = ndy + "px";

[html中鼠标拖拽的实现-麦田里的守望者-csdn](https://blog.csdn.net/qq_40281230/article/details/78119098?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_title~default-1.control&spm=1001.2101.3001.4242)

最原始，但是很累呀，麻烦的不得了

### Html5 drag and drop

[HTML5 拖放（Drag 和 Drop）- 菜鸟教程](https://www.runoob.com/html/html5-draganddrop)

给source添加属性：draggable="true"

给target添加对应方法：ondragover, ondrop

<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">

</div>

<img src="tangsan.jpg" id="drag1" width="336" draggable="true"

>

非常好，可以实现很不错的效果了：

[React 实现拖拽功能 - 我不吃饼干呀 - cnblog](https://www.cnblogs.com/wenruo/p/10225377.html)

### React-dnd

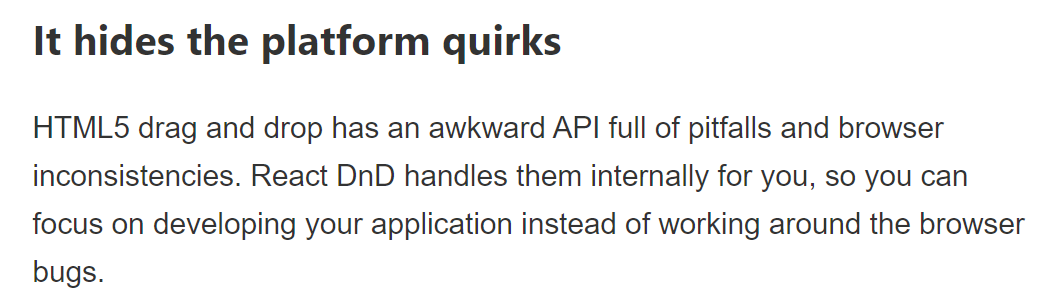
[react-dnd官方文档](https://react-dnd.github.io/react-dnd/about)

个人表示很难用！！！

虽然熟悉了之后，特别是做大点的项目，确实香，但是感觉Dan Abramov写的东西对初学者都不友好，太难用啦，例如：react, react-redux, react-dnd

希望作者能进一步优化简单点呀~

但是王婆卖瓜，总是喜欢自卖自夸，react-dnd官网对html api的评价是：



通过具体的例子，循序渐进的感受react-dnd的概念和api

## React嵌入到其它工程

# 待研究

# 参考链接

[尚硅谷2021版React技术全家桶 - B站视频](https://www.bilibili.com/video/BV1wy4y1D7JT?from=search&seid=6418214089722823230)