　前提是我们需要下载 nodejs

　　使用 npm 下载 react 的脚手架，react-router-dom，redux

　　　　全局下载 react 的脚手架：npm i create-react-app -g

　　　　使用 react 脚手架创建一个 app 的项目：create-react-app app

　　　　根据指令进入 app 项目： cd app

　　　　下载 react 的路由插件：npm i react-router-dom -S

　　　　组件通信的优化： npm i redux -S

　　以上便是把我们的 react 的脚手架配置完了

　前提是我们需要下载 nodejs

　　使用 npm 下载 react 的脚手架，react-router-dom，redux

　　　　全局下载 react 的脚手架：npm i create-react-app -g

　　　　使用 react 脚手架创建一个 app 的项目：create-react-app app

　　　　根据指令进入 app 项目： cd app

　　　　下载 react 的路由插件：npm i react-router-dom -S

　　　　组件通信的优化： npm i redux -S

　　以上便是把我们的 react 的脚手架配置完了

**组件的生命周期克分为三个状态**

Mounting：已插入真实 DOM  
 Updating：正在被重新渲染

Unmounting：已移出真实 DOM

**生命周期的方法有：**

**componentWillMount** : 在渲染前调用,在客户端也在服务端。

**componentDidMount** : 在第一次渲染后调用，只在客户端。之后组件已经生成了对应的DOM结构，可以通过this.getDOMNode()来进行访问。 如果你想和其他JavaScript框架一起使用，可以在这个方法中调用setTimeout, setInterval或者发送AJAX请求等操作(防止异步操作阻塞UI)。

**componentWillReceiveProps** : 在组件接收到一个新的 prop (更新后)时被调用。这个方法在初始化render时不会被调用。

**shouldComponentUpdate :** 返回一个布尔值。在组件接收到新的props或者state时被调用。在初始化时或者使用forceUpdate时不被调用。   
可以在你确认不需要更新组件时使用。

**componentWillUpdate :** 在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用。

**componentDidUpdate :**  在组件完成更新后立即调用。在初始化时不会被调用。

**componentWillUnmount :** 在组件从 DOM 中移除之前立刻被调用。

react讲解：

MVVM模式的前端框架，也是偏向于view(UI)层

刚开始react是UI框架 bs ele iview easy-UI ionic

redux 管理数据 vuex 一样

react 才是个MVVM框架

优点：

1、声明式设计

2、虚拟DOM

记住：虚拟DOM加载比真实DOM更费时

3、jsx语法

home.jsx home.js

js的扩展 执行更快 效率高 不易出现错误

4、灵活

可以和任何的框架配合使用

vue bs ele ionic jq

axios

5、组件化 一切皆组件

6、单向数据流

数据永远从上到下

缺点：

1、上手难，开发成本高 all in js

2、用了数据管理redux之后，页面逻辑变得复杂，不易阅读

npm install vuex -S --save最终要发布到线上的插件

-S或者--save会把插件下载到 dependencies里面，是代表要发布到线上

npm install less --save-dev 将插件下载到本地 辅助你开发项目，但是此插件不上线

-D或者--save-dev 会将插件下载到 devDependencies里面，不上线，辅助你开发

install 简称i

react:

1、会写页面就可以

Redux：

store -> state

reducer -> mutations

action -> actions

react-redux:

privider

connect

react-router-dom:

路由，嵌套路由

和react配合使用的UI框架：

react + antd 配合使用比较好

react-router: 2.0

是react2.0路由的路由库，只能将路由进行统一管理，API是2.0路由的

react-router-dom: 4.0之后 常用

是react4.0路由的路由库，你可以把路由分别写在组件内，也可以写在统一管理的文件内

react-router: 4.0

是react4.0路由的路由库，可以写在统一管理的文件内,API是4.0路由的

4.0路由：

BrowserRouter www.zimogod.com/index/income 浏览器正常路由跳转

HashRouter www.zimogod.com/#/index/income 浏览器hash跳转 锚点跳转

Route 路由容器组件

Link 路由导航组件

Switch 单一路由容器组件 显示哪个组件，就加载哪个组件，其他组件一律不加载

Redirect 路由重定向 有两个属性，from:从哪里来 to：到哪里去

react中，路由传参：

query，hash值不参与路由跳转，只是跳转时携带的数据信息而已，也就是说 ？ #之后的路径不参与跳转

react中query是默认隐藏的，url地址栏不可见

match:是路由匹配的对象，是路由的匹配规则

exact:是路由的精确匹配

localhost:8080/

localhost:8080/index

Route:

path: 可选属性，路径属性，用于你跳转，url地址栏显示路径的作用

component：必须属性，才是正真的将你跳转的路径导向到你跳转的那个组件

url地址栏里面，#，？后面不是真正的路径，不参与路由的跳转，只是页面之间传递参数

props：react中，所有的数据，属性都会保存在props里面，

要想得到数据，必须去props里面找

params：获取 this.props.match.params.id

query: 获取 this.props.location.query.name

state: 获取 this.props.location.state.name

2.0路由：

<Router history={history}>

<Admin></Admin>

</Router>

Router 路由包裹容器

IndexRoute 默认显示内容认路由

Link 路由导航组件

NavLink 路由导航组件

HashHistory Router组件的必传属性，以hash跳转

Redirect 重定向，可以导向到指定组件

Route 路由组件 可嵌套

<Router>

<IndexRoute component={Home} />

<Route path="/index" component={Index} />

<Route path="/come" component={Come} />

<Route path="/index" component={Index}>

<IndexRoute component={page1} />

<Route path="/page1" component={page1} />

<Route path="/page2" component={page2}>

<Route path="/page1" component={page1} />

</Route>

</Route>

</Router>

redux:讲解：

项目中有没有redux都可以，都能正常开发项目

1、store

是redux的核心，是数据仓库，存放公共数据的仓库，store将数据分发到视图层

store接收三个参数

第一个是reducer，处理之后的reducer，或者默认的reducer

第二个参数是initialState,视图加载的时候，默认展示在页面上的初始化数据 可选参数

第三个参数是中间件，为了处理异步代码而来的第三方插件

2、reducer

是逻辑处理中心，是纯函数，必须有返回值，没有处理时，必须返回原有的数据，

是同步操作，不处理其他业务，可以有多个reducer，最后可以合并为一个reducer

接收两个参数

第一个参数：接收的是默认数据

第二个参数：接收的是action派发过来的事件类型 ADD\_NUM DELETE\_USER ADD\_USER

函数：输入什么，就输出什么如：输入的是数组，返回的必须是数组，必须有返回值，必须是同步操作，

只处理数据，不适合其他任何程序

3、action

是异步操作，从在于视图层，进行派发事件类型 dispatch派发的意思

Object.assign():

此方法是将另外的对象或者数据，拷贝到一个新的对象内，新的对象必须是空的

组件内： vuex：

data(){

return { state

}

}

computed getters

生命周期 mutations

methods action

嵌套组件 modules

new Vuex.Store({

state,

mutations,

action,

modules

})

applyMiddleware:叫做中间件，处理异步数据的中间件，属于redux的API；必须放在store的最后面

redux-thunk:是真正的中间件，可以处理异步请求，异步操作，必须放在applyMiddleware的最后一个参数

redux-devtools-extension:开发者工具，没什么用，但是可以在打印台进行调试redux代码

组件的生命周期

1、自动执行

2、按顺序执行

常用生命周期：

shouldComponentUpdate

componentDidMount

render

getDefaultProps：

设置默认属性

getInitialState：

设置默认数据，在使用es6的class语法时是没有这个钩子函数的，可以直接在constructor

中定义this.state。此时可以访问this.props

componentWillMount：

组件初始化时只调用，以后组件更新不调用，整个生命周期只调用一次，此时可以修改state。

componentDidMount：

组件渲染之后调用，只调用一次。

shouldComponentUpdate：

react性能优化非常重要的一环。组件接受新的state或者props时调用，

我们可以设置在此对比前后两个props和state是否相同，如果相同则返回false阻止更新，因为相同的

属性状态一定会生成相同的dom树，这样就不需要创造新的dom树和旧的dom树进行diff算法对比，

节省大量性能，尤其是在dom结构复杂的时候

render：

react最重要的步骤，创建虚拟dom，进行diff算法，更新dom树都在此进行。此时就不能更改state了。

componentWillReceiveProps：

组件初始化时不调用，组件接受新的props时调用。

componentWillUpdata：

组件初始化时不调用，只有在组件将要更新时才调用，此时可以修改state

componentDidUpdate：

组件更新完成，此时可以获取真实的DOM节点

componentWillUnmount：

组件将要卸载时调用，一些事件监听和定时器需要在此时清除。

细节：

setState()在哪些生命周期可以使用，或者哪些生命周期可以修改数据

render函数在哪些时期可以调用

哪些生命周期可以获取到真实的DOM元素，可以操作DOM

// 函数嵌套函数，想用父级函数的this指针，则必须代理this指向

// 解决办法：

// 1、可以写箭头函数

// 2、缓存this 在constructor函数中缓存 或者绑定

组件自身的属性是不可逆的，或者不可改变的,改变属性，改变的是子组件的属性，不能修改自身组件的属性

react中不能直接修改state数据，必须通过setState({}) 小程序中修改数据必须通过setData({})

react组件通信

react组件通信:

父传子

通过props进行传递,发送数据的组件可以单独传递数据，也可将父组件内所有数据传递下去 如：{...this.state}

放在子组件上，子组件继续向下传递数据，则在当前子组件上使用 {...this.props},下面的组件从组件自身的属性

中去获取数据

子传父

1、

子组件发送数据，父组件接收数据，回调函数

父组件中定义一个函数，必须接收形参，将此函数以属性的方式传递给子组件

子组件调用父组件传来的函数，传入实参

子组件调用了父组件的函数 传入数据

可以单个数据发送接收，可以整个state发送 接收

2、

父组件中通过ref 指向子组件，获取子组件实例

子组件不变

同级之间

1、子传父，父传子

2、自定义事件 总线代理 万能的

通过bus.$emit()发送数据，在挂载中的生命周期可执行

通过bus.$on()接收数据，在挂载之前的生命周期可执行

跨级之间

1、props层层传递

2、总线代理 $emit() $on()

3、context 不讲 过时 不推荐使用 容易造成组件混乱

4、借助了props，将数据从深层组件传递到外层组件