**PWA ServiceWorker 储存上一次的数据。断网情况下进app,依然能看见上次的页面。**

**React基本语法jsx语法；**

1. 在jsx语法中写js语句要加{}；
2. Html标签不加引号，自定义标签首字母大写；
3. 标签中的属性与关键字区分开来，如className, htmlFor等;
4. 最重要，this的指向，一般在construtor中bind(this)[建议，因为方面后面优化]，也可以在render中bind(this)或者箭头函数。
5. 标签上的事件命名 handle+事件名。
6. Fragment。16版本之后出的，就是一个占位符。
7. DangerouslySetInnerHTML = {{\_\_html: }}

**React设计思想**

React的设计思想为数据驱动，通过更改数据来驱动视图变化，而不是通过dom操作来改变视图变化。

**React中组件**

React中组件为树形结构

React父子组件之前传值（面试）：

父-》子 通过标签上面的属性形式，可以传递数据和方法。

子组件通过this.props来调用数据和方法。注意在方法的时候要注意在父组件bind(this)。

子-》父 通过调用父组件传过来的方法，实现通信。

**React优化**

解构赋值？

一般样式在后面引入。

This指向的绑定统一放在页面顶部的constructor里面。

jsx中一般写内容，像一些数据处理的逻辑一般单独提出来。

setState建议传函数而不传对象。传函数的时候是异步的。善用preState和箭头函数。

在react中,不允许我们对state做任何改变。如果真的要改那就复制一份出来改。

**React思考**

1、Jquery及js 都是命令式操作dom。React是面向数据变程，声明式开发。

2、可以与其他框架并存。各个框架管理自己的dom。

3、组件化

4、单向数据流（父组件可以向子组件传值，但是子组件一定不能直接的去改变这个值。如 this.props.list = []这种做法是错误的）优点：方便维护，出问题方便问题定位。

5、视图层的框架。只负责数据和页面渲染上的东西。像组件之间的传值，就由其他框架redux等来解决。

6、函数式编程。优点：前端自动化测试方便便捷，因为代码都是一个个函数，只需给一个输入值，查看结果是否符合预期。

**React调试**

Chrome商店。调试工具对于每个组件的数据都有监听，很方便查看每个组件的数据变化。

**对于接收props参数的校验PropTypes和DefaultProps；**

对于propTypes设置了类型，但是又没有传的属性值，propTypes不会去做校验。

对于propTypes设置了类型并且isRequire，但是又没有传的属性值，propTypes就会报错。但有时候父组件确实没办法传值，这个时候通过DefaultProps来解决这个报错。

看官网的英文文档

**Props,state,render函数之间的关系**

1、当组件的state或props发生变化时，他自己的render函数将重新执行。

2、当父组件的render函数执行时，他的子组件的render函数都将被重新运行（不管子组件有没有props接收父组件的state值）。

总的原因就是state变化引起render

**什么事虚拟DOM**

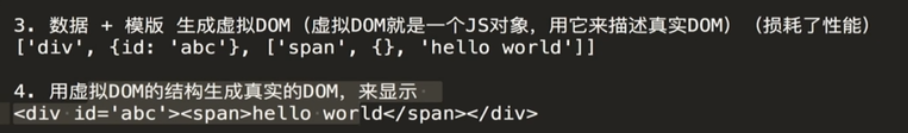


js创建一个js对象很简单，但是创建一个dom的时候消耗是比较大的。

通过虚拟dom减少了真实dom的生成和对比。

**深入了解虚拟DOM**

真实底层是先生成虚拟DOM在生成真实dom



jsx 编译成 React.createElment(‘div’, {} , ‘item’) 函数，然后通过该函数先生成js对象即虚拟dom，然后再生成真实的dom。

createElment是一个react偏底层的函数，完全可以通过该函数进行开发。

JSX之所以存在是提高开发效率，语法简单。

同理vue里面的虚拟dom的机制也是同react里面的虚拟dom基本一致的。

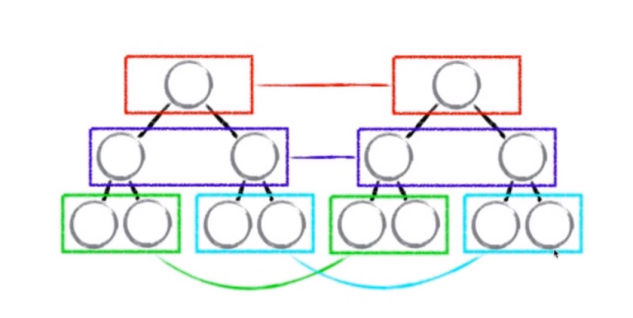
虚拟DOM的优点：

1. 性能提升；
2. 跨端应用得以提升。Native（因为js对象在原生应用里面能够识别）

**虚拟DOM的diff算法**

setState异步为了提高react底层性能。例如，高频的setState操作是，会将多个setState合并成同一个setState, 减少比对次数。

Diff算法 同层虚拟DOM比对，比对速度快。



列表循环中引入Key值，也是提高dom比对性能，key要保持稳定，且在最外层。在项目中能不用index做key就不用。（实践）

**React中ref的使用**

在react中用ref来操作dom的。

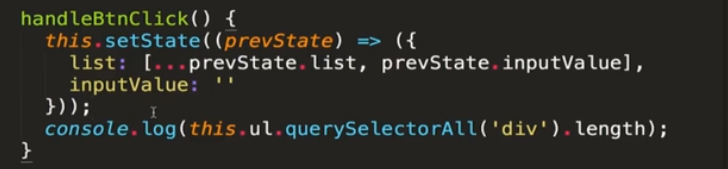
获取dom

1、e.target;

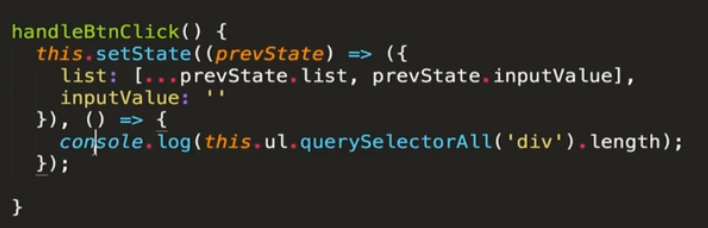
2、ref = {(dom) => {this.xxx = dom}}

一般情况下react不建议用ref（动画例外）。

ref在使用中常见的问题，与setState合用：



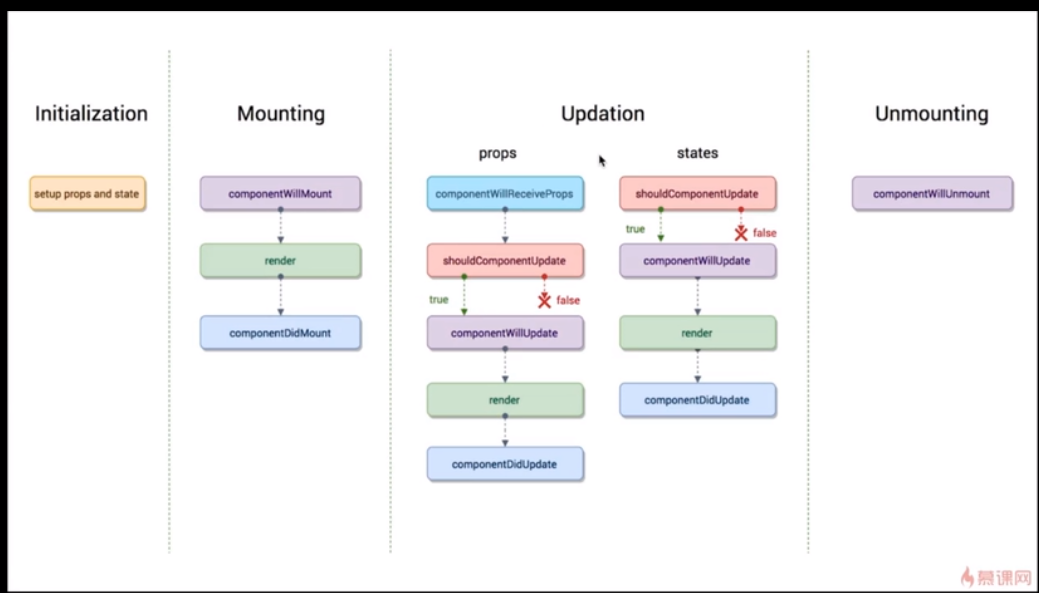
setState是异步的，下面的dom操作可能先运行也可能后运行。正确的为：



**React 生命周期函数（重要）**

自动

constructor 也是在组件创建的时候自动执行的，但是它是属于es6语法里面的，不在react里面，所以不归为react生命周期。



Initialization（初始化）

constructor

Mounting(挂载) 挂载指组件第一次在页面中呈现出来（或存在于dom中）的一个过程。

componentWillMount //在组件即将被挂载到页面之前自动执行

render //页面的挂载时自动执行

componentDidlMount //在组件即将被挂载到页面之后自动执行

Updating(组件跟新)

shouldComponentUpdate //组件在被跟新之前，会自动执行。需要返回一个布尔值， 来决定组件是否更新。

componentWillUpdate //组件在被跟新之前，会自动执行。但是shouldComponentUpdate 之前执行。

componentDidUpdate //组件在被跟新之后，会自动执行。

componentWillReceiveProps // 一个组件从父组件接受了props参数，只要父组件render 函数重新执行，子组件生命周期就会自动自行这个函数。（实践 子组件的mounting 与父组件的mounting是同时的，实时不是一起执行的。）（实践证明子组件没有接受props也会执行呢？）

Unmounting(卸载) //在页面中剔除的一个过程。

componentWillUnmount //当这个组件即将从页面中剔除的时候，会被执行。（实践 删除只执行这一个生命周期函数）

如果没有mount即没有在DOM呈现出来(无论是子组件还是父组件),那后面的周期函数是永远不执行的

**生命周期函数的使用场景**

component 内置了除render之外的其他函数，所以在自定义class中 render是不允许删除的。

场景1：父组件重新render时，不希望其子组件重新render。

处理：用shouldComponentUpdate(nextProps, nextState); 来防止子组件的不必要渲染，从而提升性能。面试中经常问react的性能优化，多总结

场景2：页面数据加载请求（与后面的事件触发请求相区别）。

处理：constructor或componentWillMount或componentDidMount都可以，他们在生命周期中只执行一次。业界争议很大，但是建议用componentDidMount。因为在做react Native 或者服务器端的同构是，可能会产生冲突。在render会无限循环。

注意：请求回来的数据[...res.data]这样复制一份出来操作。

**react中用css3实现动画**

补充：css动画效果transition

css过渡动画效果keyframes; forwords参数能够将动画最后一帧保存下来。

**react-transition-group实现动画**

CSStransition模块

1、自动根据属性in的布尔值给标签添加删除className;

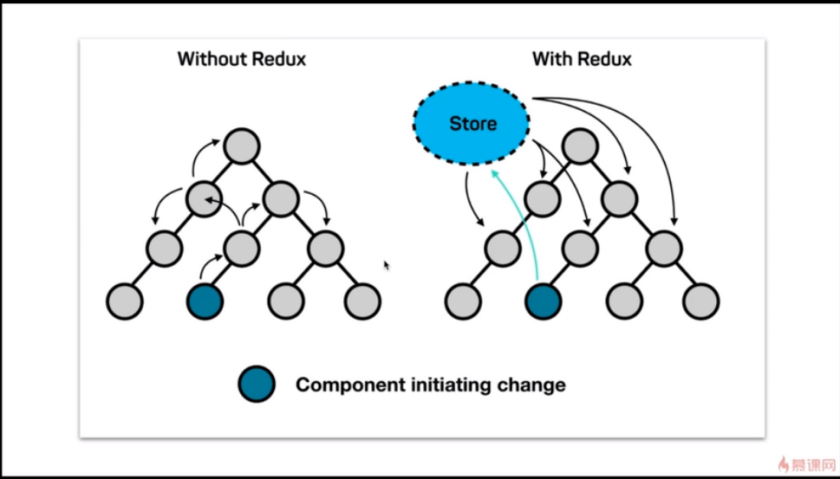
2、钩子函数(onEnterd入场动画完成第一帧时候触发, appear页面一进来就触发)，钩子函数是跟生命周期函数一样的东西，在某个特定的时间会自动执行的函数。

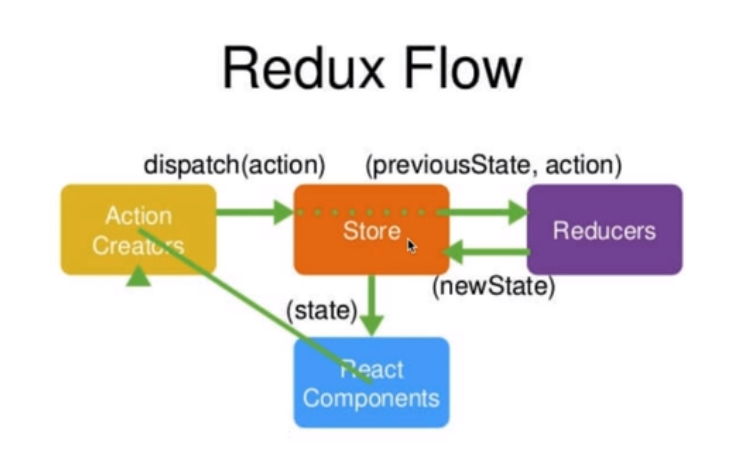
Transition 更底层的模块

transitionGroup 多个dom元素的动画。配合CSStransition且in属性去掉。

**Redux概念简述**

Redux 数据层框架。设计理念：所有数据放在store里面进行管理，其中一个组件改变了store里面的数据，其他组件会感知到，然后重新从store里面拉取数据。从而实现组件之前的传值。





Store存储数据的公共区域。

**Antd**

React的ui

Showcode, style修改原样式

**Store的创建**

import自动会去找index.js

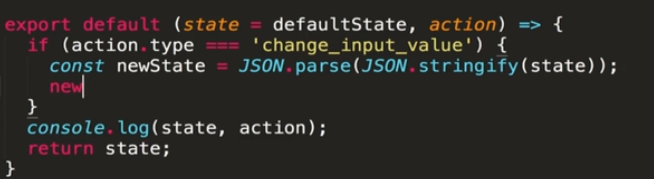
store 接受一个reducer。

reducer 必须返回一个函数，接受state,action两个参数。

**Redux调试开发**

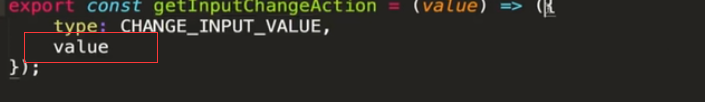
Store再加一个参数window.devtools.extend

（重要）Store.dispatch(action)，store 接受到action后将自动转发给reducer。然后reducer根据type做相应的数据逻辑处理（注意处理的时候还是拷贝出来处理，reducer可以接受state,但绝不可以修改state.）。



组件与store。通过store.subscribe（this.setState(store.getState())）（订阅）方式来监听store数据的变化，然后更新组件store.getState()。(实践5遍以上)

优化：

1. types字符串拆分成actionTypes.js定义成常量。因为字符串写错了不会报错，变量写错了才会报错，方便找出问题原因。
2. action也拆分出来actionCreators，返回各种action对象。

**Redux知识补充**

1. Store必须是唯一的，整改应用中只有一个store公共存储空间
2. 只有store只有自己能改变自己的数据。Reducer里面的state是store里面的数据，所以要拷贝出来操作。Store接受到reducer return 的newState，自己跟新store里面的state。
3. Reducer必须是纯函数。（指给定固定的输入，就一定有固定的输出（有date时间,ajax异步等），而且不会有任何副作用（不要对state,action做修改））；
4. createStore(reducer, devtools), store.dispatch(), store.getState(), store.subScribe()

**UI组件和容器组件的拆分**

1、Ui组件(傻瓜)负责页面的渲染；

2、容器组件来负责业务逻辑；

3、Ui组件与容器组件存在子组件和父组件的关系

RenderItem={(item,index) => (<List.item onClick={() =>{this.props.handleItemDelete(index)}}/>)}

子组件调用父组件方法，并且需要传递参数index。写箭头函数改变this指向，而不是写成onclick = {this.props.handleItemDelete(index)}

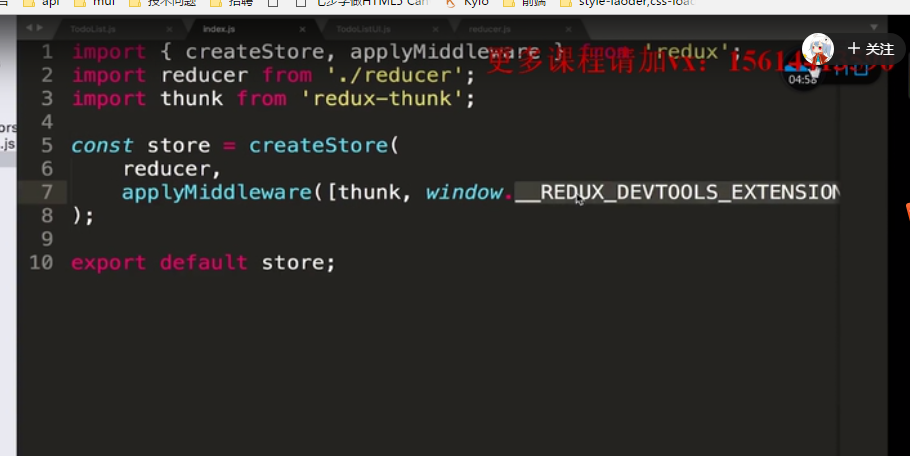
**无状态组件**

无状态组件就是一个js函数，返回一个jsx模板。

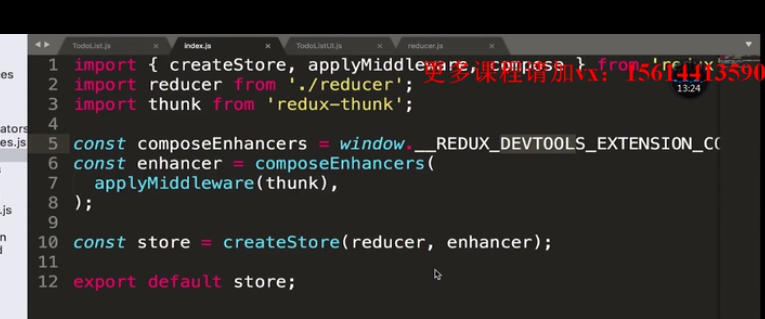
当一个组件中只有render函数的时候，我们完全可以通过一个无状态组件替换掉这个组件。以此提高性能。因为一个是类（会执行生很多命周期函数），一个就是函数

一般ui组件可以用无状态组件替换。（不是完全看具体情况）

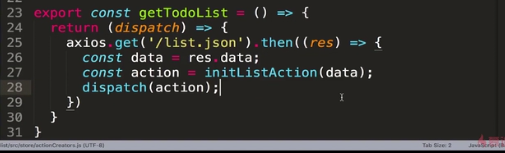
**Redux-thunk中间件进行ajax请求**



实际上这个也是redux的一个中间件，上面的配置是错误的

这个配置才是对的compose包装函数。

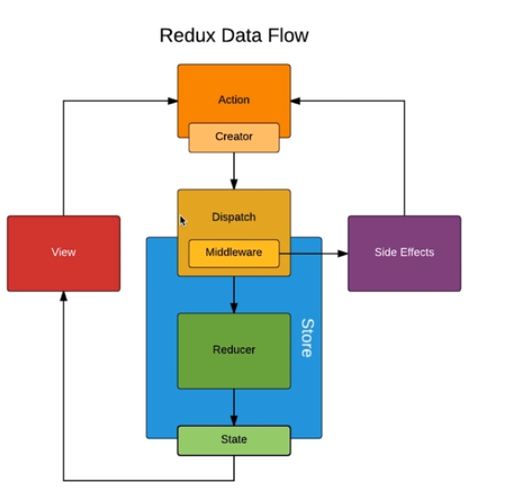
1. 通过redux-thunk,actionCreators里面可以export一个function，也就是dispatch可以接受一个fun，而不再只是接受一个对象。 通过store.dispatch（fun）之后可以自动执行这个fun（里面的axios）。



Fun在store环境下执行所以可以接收 dispatch这个函数。

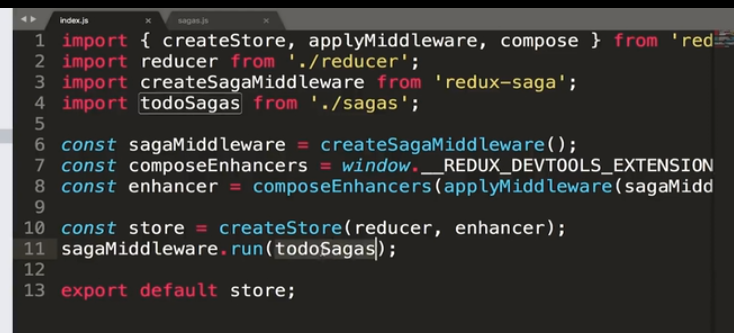
2、优点：随着项目的增大，有必要把数据相关的提出来单独管。还有就是自动化测试时，测一个函数比测一个生命周期函数要方便很多。

**什么事redux的中间件**



是action与store之间的中间件。其实就是对dispatch（派发）方法的一个升级,根据传入参数的不同做不同的处理。

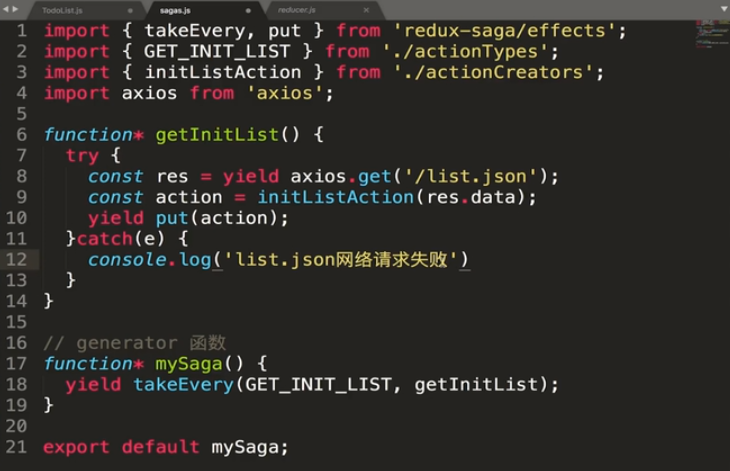
**Redux-saga中间件的使用入门。**



1. 也是异步代码拆分。
2. 与thunk不同就是请求单独提出在一个文件sagas.js;

流程：派发（dispatch）action的时候，不止reducer.js里面可以接收action。 sagas.js里面也可以接收action,然后通过takeEvery()捕捉类型，调用相应的请求函数，请求完数据后通过put(action)在派发action。

注意：1、sagas.js 里面必须是一些generator函数。



1. saga相对与thunk多了很多api。Call, takeList都是经常用的，而thunk很简单。
2. 对于大型的应用建议用saga,可以将复杂的逻辑请求提出来。

**React-redux**

第三方模块，方便我们在react中使用redux

1. provider。<Prodiver store={store}/>，provider连接了store，provider组件里面的组件都可以通过connect获取store里面的内容。注:provider里面内容都在一个div里面
2. Connect。组件和store做连接，通过一定的规则（mapStateToProps, mapDispatchToProps）做连接。

mapStateToProps将store数据映射到组件的props（数据映射props）。

mapDispatchToProps 将store.dispatch映射到props (方法映射到props，方法里面可以调用dispatch)。

mapStateToProps是数据，mapDispatchToProps 是业务逻辑

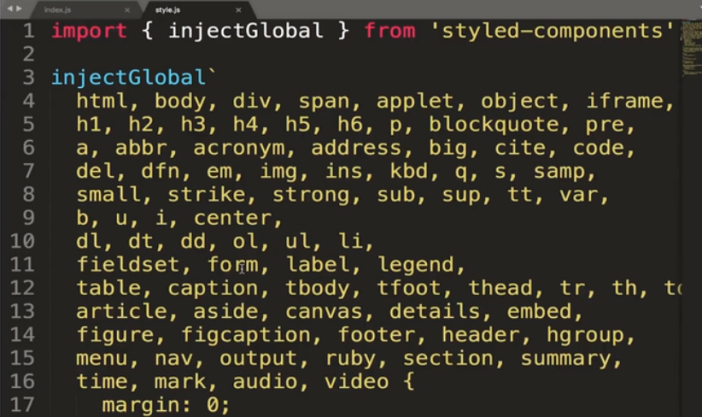


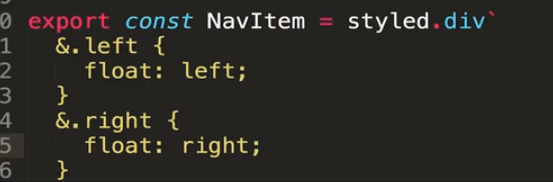
**Connect(mapStateToProps, mapStateToProps)(UI组件) = 容器组件**

3、mapDispatchToProps 这样之后TODOList 就相当于一个ui组件。方法数据都从props接受。

**实战**

1、styled-component 将css换成style.js

新版本已改为createGlobalStyle了



styled-component实际上就是返回一个个带样式的组件，并且样式不互相影响。（但是个人觉得这个组件将样式和标签混合在一起了）还有就是样式不能复用，不能使用css预处理器。

类似的:

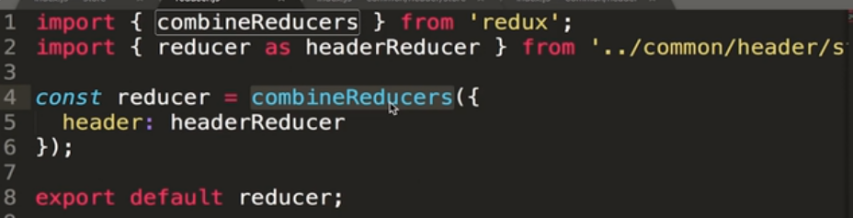
styled-jsx 可读性不是很好，但是符合React 将内容、样式和动态功能聚集在一个模块中，是高聚合的表现。

css-moudle 产生局部作用域的唯一方法，就是使用一个独一无二的class的名字（不是组件最外层元素类名唯一，而是每个类名都会编译成哈希字符串），不会与其他选择器重名。

1. reset.css 各个浏览器做到样式统一。各个浏览器对标签的默认样式是不一样的。

**combineReducer完成对数据的拆分管理**

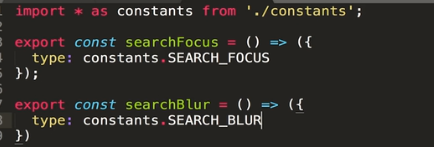
combineReducer把各个小的reducer整合成一个reducer, 从而在redux-devtool tree中看到数据层多了一层。

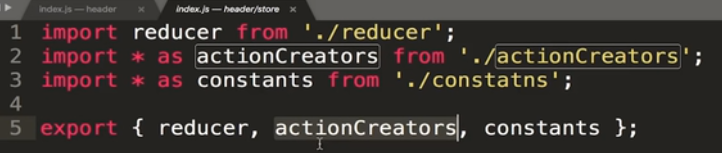
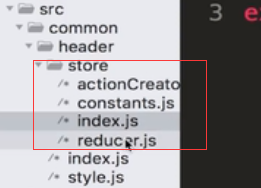


**actionCreators与constants的拆分**

1、将actionTypes替换成constents





2、

Store文件里面的东西通过index.js文件暴露出去，而不去一个一个文件的引入。

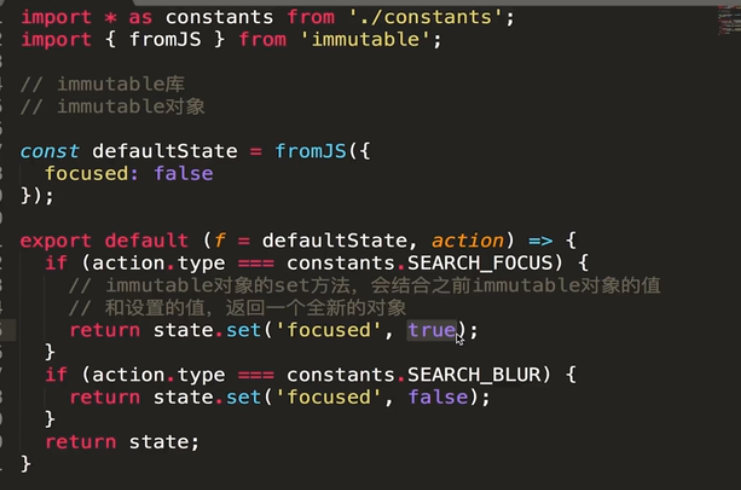
**使用Immutable.js（Facebook团队）来管理store里面的数据**

在reducer中经常会对store里面的数据拷贝，出来进行操作比较麻烦，而且经常忘记。

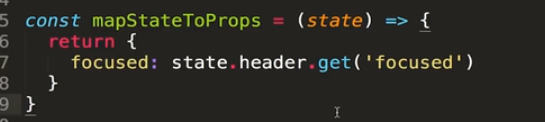
Immutable对象是不可改变的。所以可以通过fromJS(...)把state改成immutable对象来解决。

Immutable对象通过set和get 方法获取数据，修改数据从而不用拷贝数据了。

set方法，会结合之前immutable对象的值和设置的值，返回一个全新的对象。只是结合之前，没有改之前。



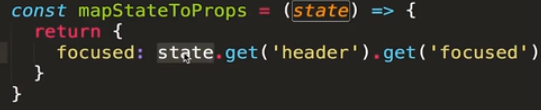
**使用redux-immutable统一数据格式**



sate是一个js对象，state.header是一个immutable对象。

因为store中combineReducers（redux）那里生成的数据不是immutable对象。

处理：combineReducers 模块从 redux-immutable 引入。

 = state.getIn([‘header’,’focused’]), 看api

**什么是路由，react中如何使用路由**

路由就是根据url的不同显示不同的内容。



path 是交集匹配，/detail 可以匹配 /detail 和 / 。但是/就只能匹配/不能匹配/detail

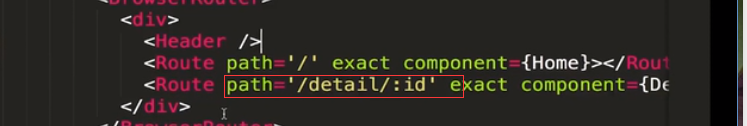
加上exact就是严格匹配。

**单页应用的跳转**

Link to点击改变url；Route是匹配url做内容的改变。

Redrict 自动改变url

**页面路由参数的传递**

1. 动态路由获取参数

通过props.math.params.id 获取id参数。（建议用这种方式）

1. 通过？传递参数



通过props.location.search.id 获取参数。

**使用PureComponent性能优化**

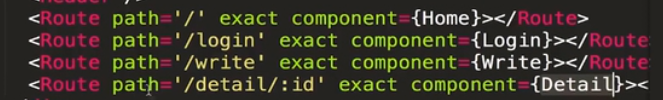
Store变化 所有组件的render都会执行，必须使用shuldComponentUpdate选择性更新。

1. pureComponent 相较于 component 多了一个shuldComponentUpdate方法，不用每个组件去写。
2. pureComponent 必须 配合immutable.js 使用。不然会遇到很多坑

**异步组件(懒加载)及withRouter路由方法的调用**

React-loadable 就是 import(./)这种语法。React官方推荐的是Lazy。后端是@loadable-component

放在route下的组件都可以获取到路由的信息，而非路由下的组件则可以通过withRouter()方法获得路由信息



**版本升级**

**注意 componentWillMount**

16版本不会做太大的升级 只是微调

只需看新内容

**自己实现一个route**