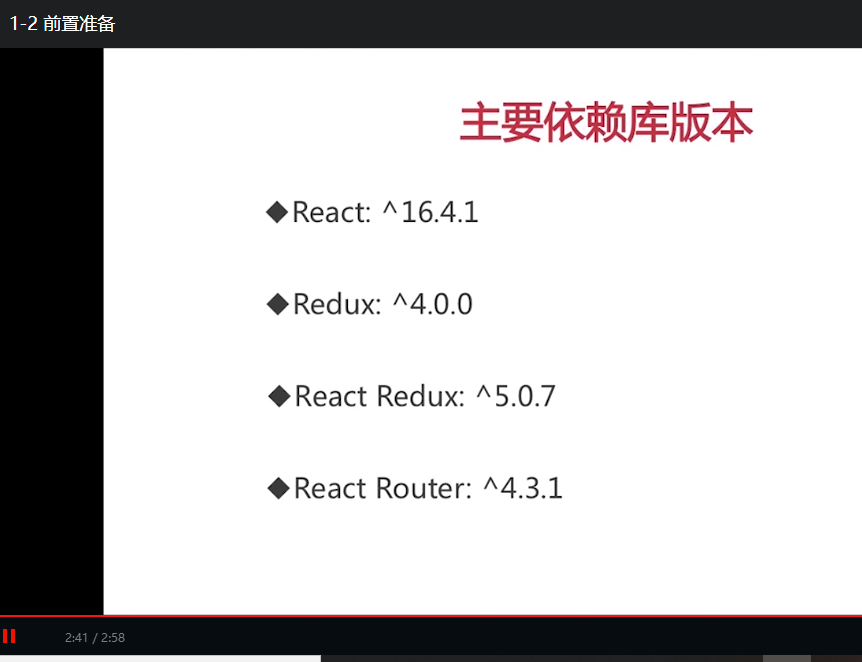
React16+Redux实战企业级大众点评Web App

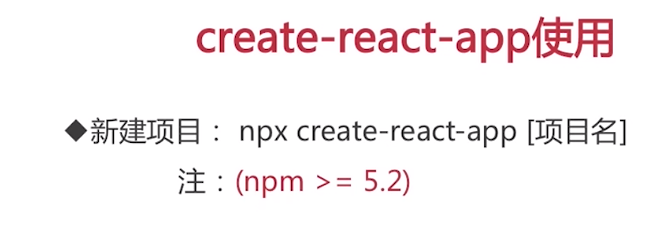
Vscode插件

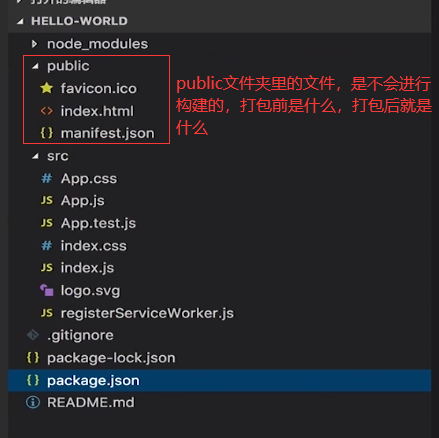
格式化代码插件 Prettier-code

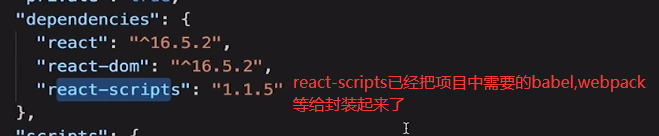
React 生成常用模板的工具 Reactjs code snippets

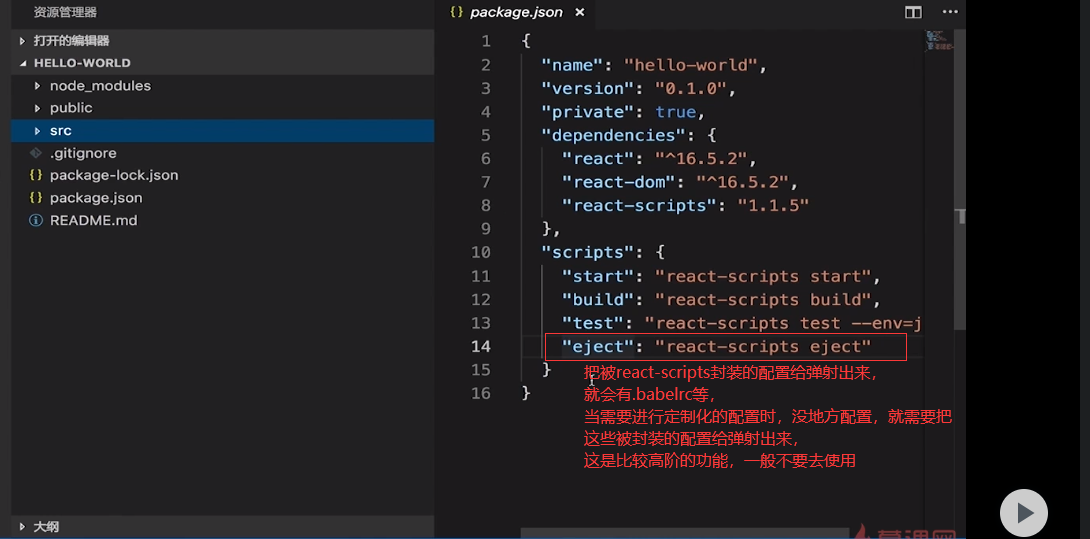
重命名标签时会同时对其闭合标签起作用 Auto rename tag











模仿数据

npm i serve -g

$ serve

INFO: Accepting connections at http://localhost:5000

Package.json中由于版本太高，导致proxy配置不生效

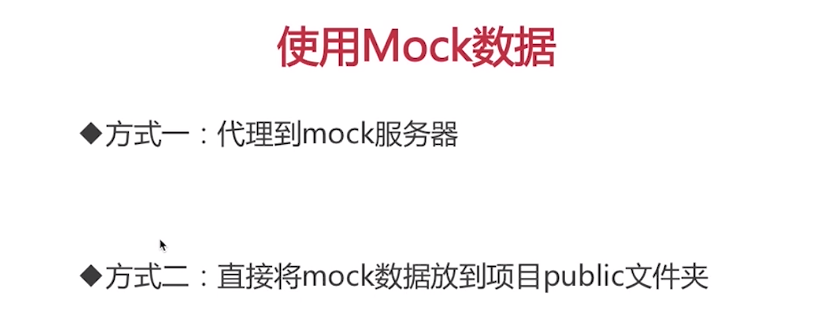
"proxy": {

"/api": {

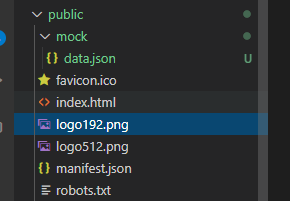
"target": "http://localhost:5000"

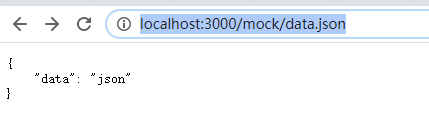
}

}

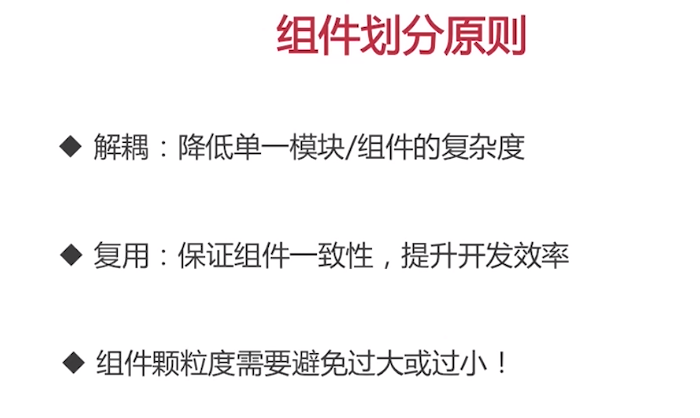


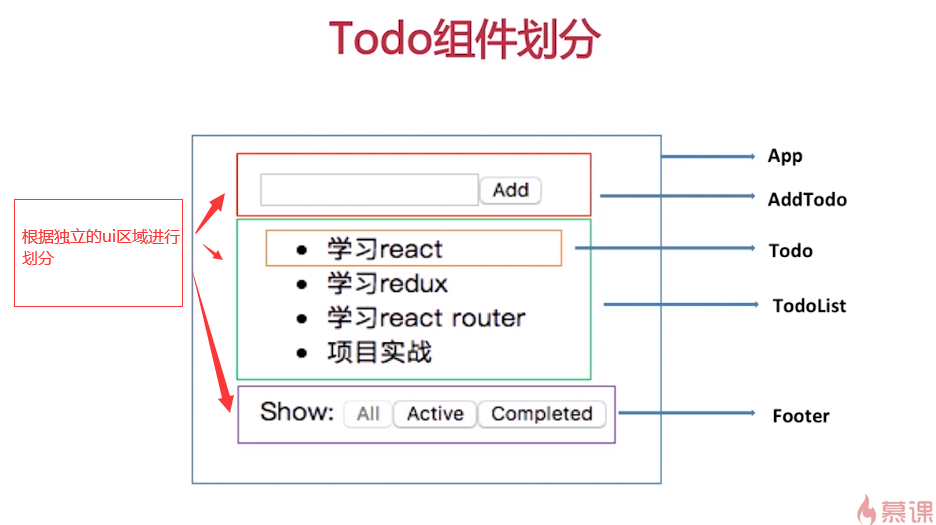
<http://localhost:3000/mock/data.json>

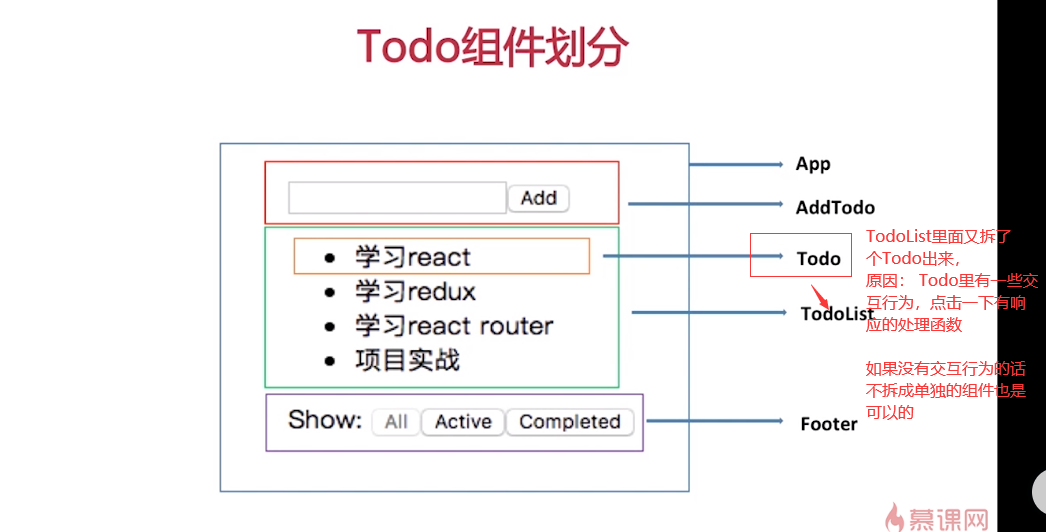




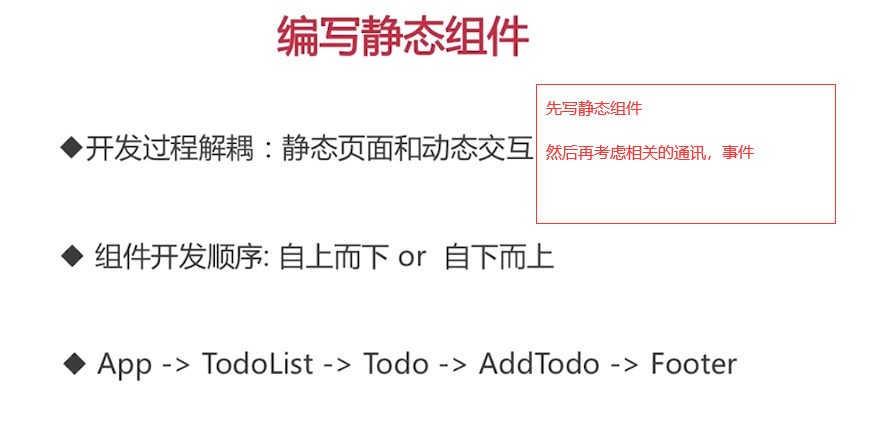
3-1 组件划分





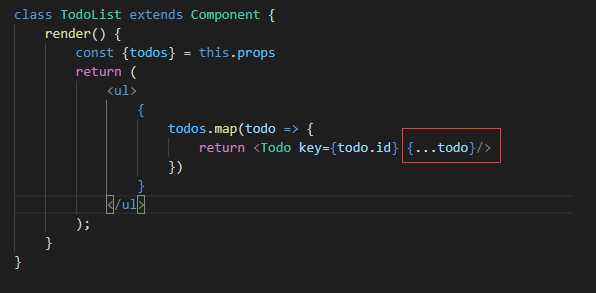


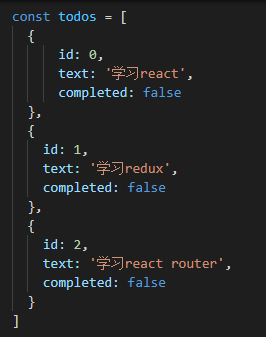
3-2

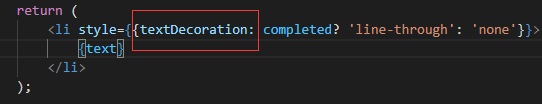


rcc是React create component ,reactjs code snipet提供的

TodoList.js的todos是父组件传进来的，先在父组件定义个常量，方便调试





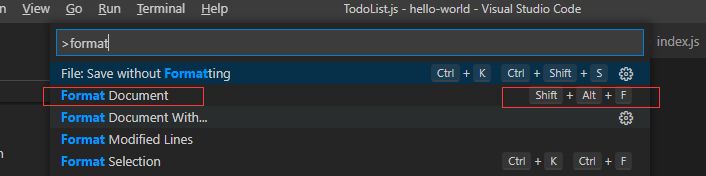


Jsx中用驼峰来写样式属性





prettier格式化



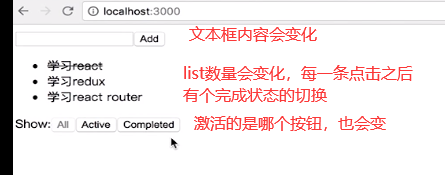
3-3

1 控制view,与ui直接相关

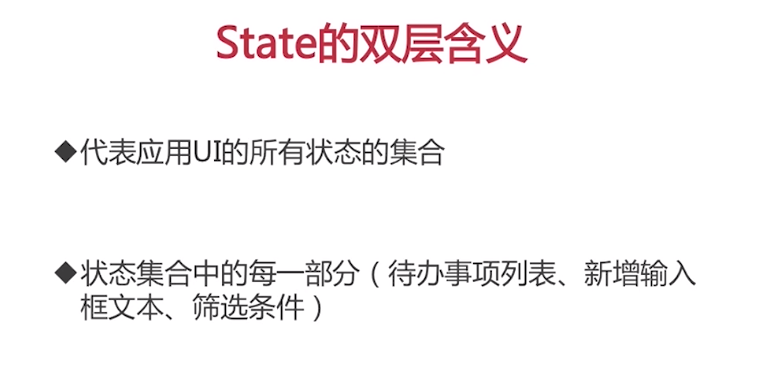
2 可变

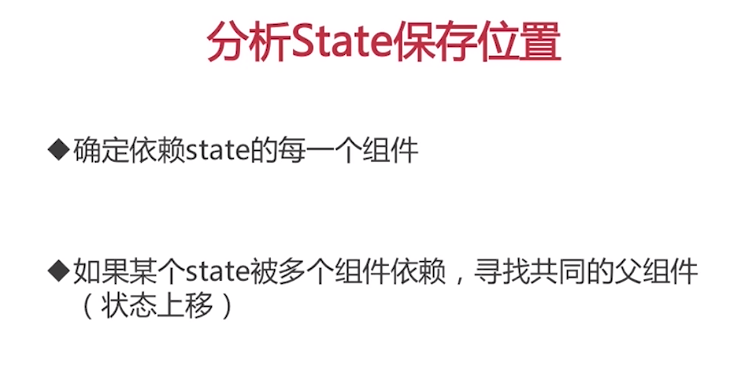






3-4



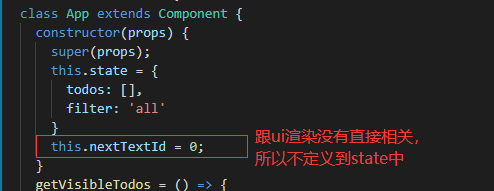


TodoList.js 和 AddTodo.js都用到了todos这个数组，所以就要移动到其父组件里面去。

Footer.js的状态all,active,completed会影响到todos，因此也要状态上移。

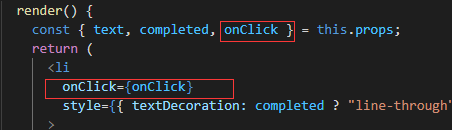
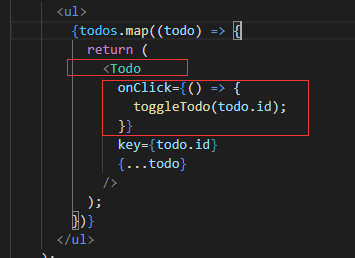
3-5



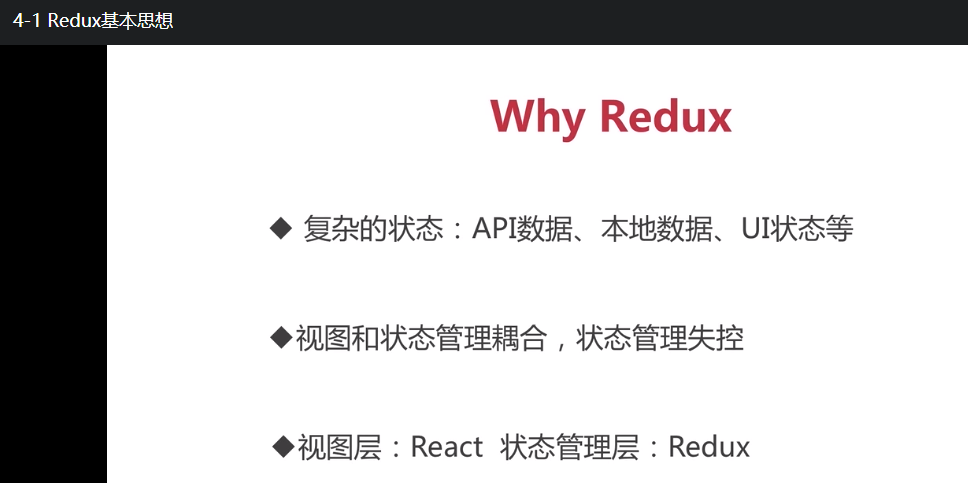


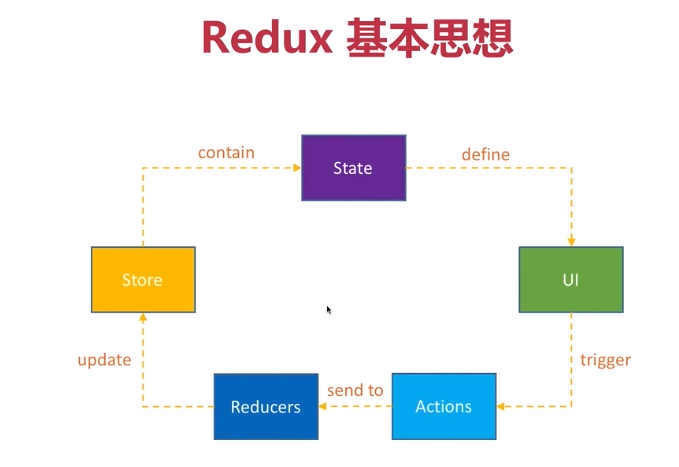
Todo组件上的onClick只是作为props传进去，并不能监听到里面的dom元素点击事件，

要在Todo组件里的dom节点上绑定。



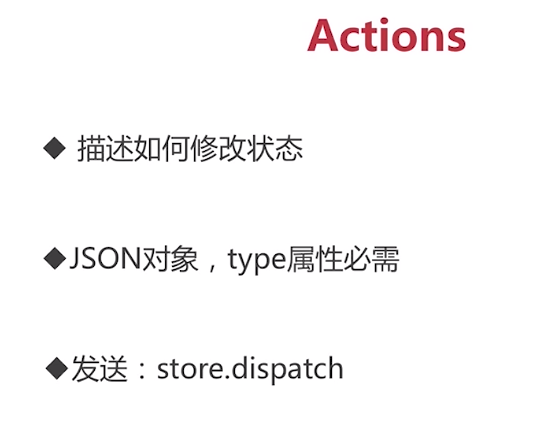
4-1 Redux



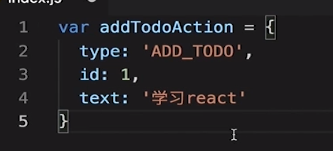




4-3

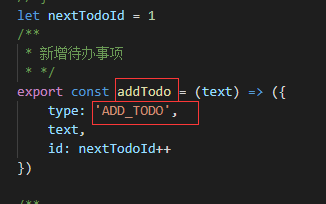


type属性是必须的

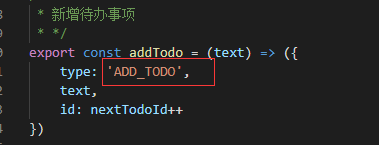


不过一般会通过一个函数来生成，

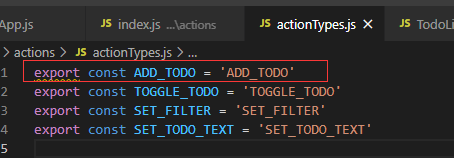
Type和方法名保持一致

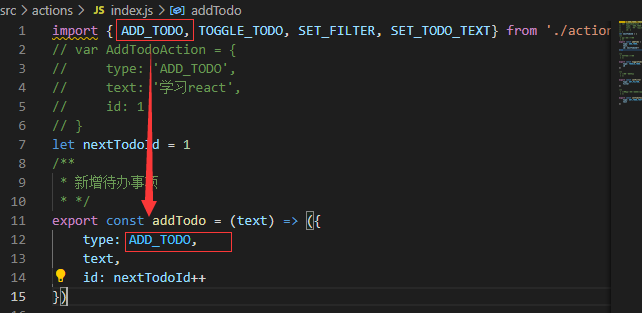


还会把type抽取成常量来使用，因为reducer中还要用到

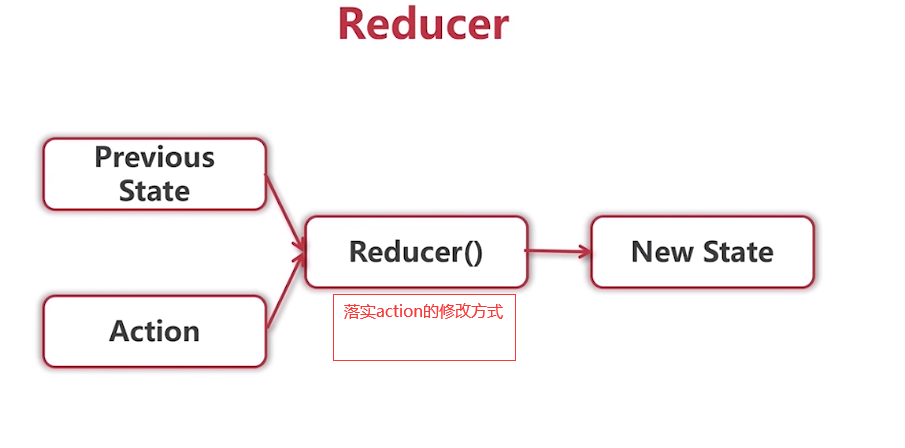


像这样

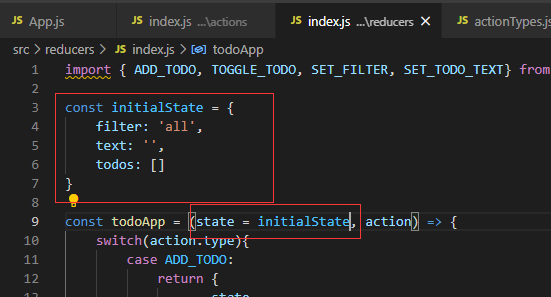




4-4

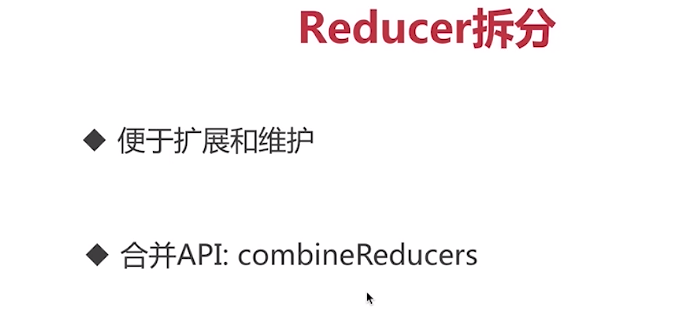


应用启动的时候，state是undefined，要给个初始值

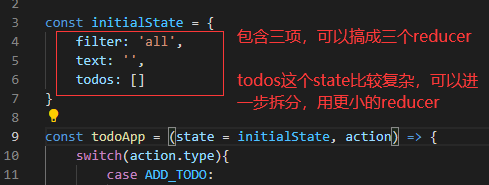


4-5

随着action越来越多，reducer.js文件越来越大，显然不合适



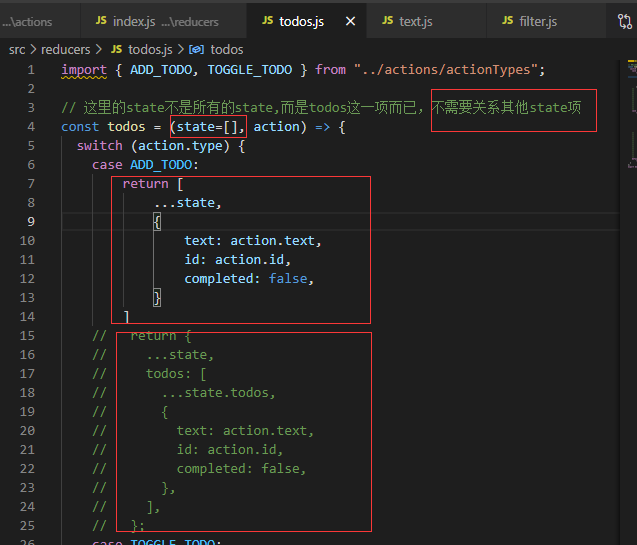
State包含三项，所以可以拆分成三个reducer



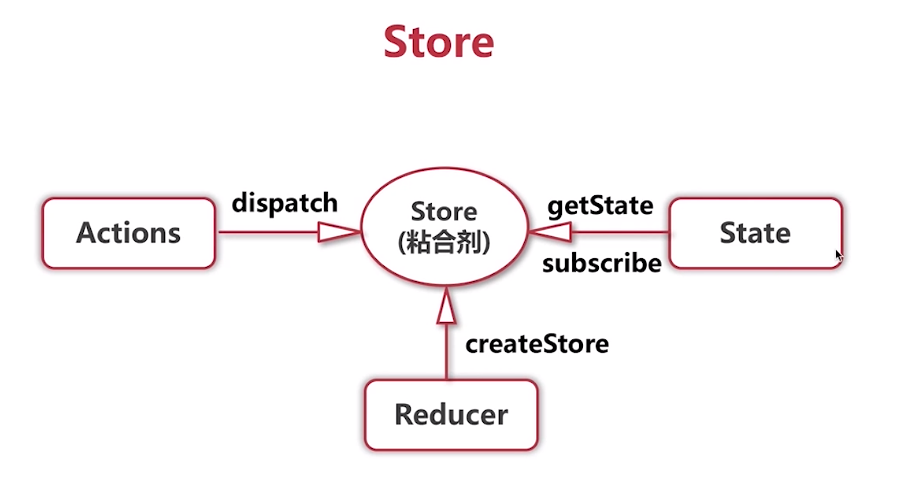
拆分后的reducer todos.js

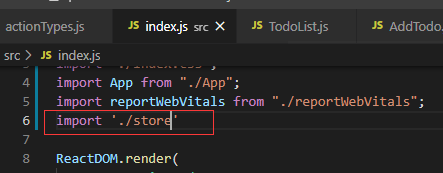
// 这里的state不是所有的state,而是todos这一项而已，不需要关系其他state项

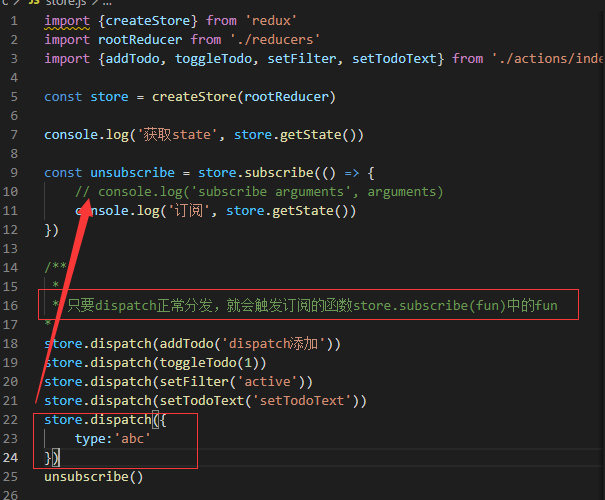




4-6 store用法







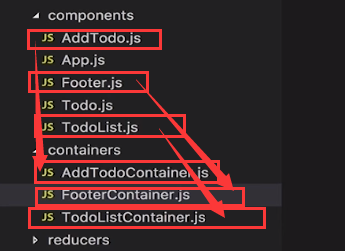
4-7 react-redux封装了react对store中状态的监听

在react中使用redux就需要react-redux这个库来实现





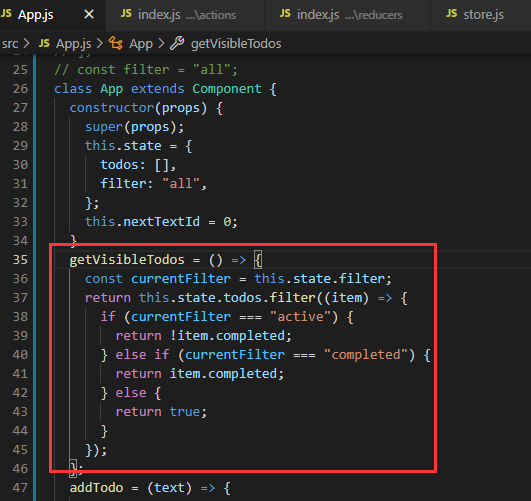
这三个组件需要操作redux中的状态，需要进行链接，链接的方式就是创建一个容器型组件



4-8



App.js中维护了很多状态，现在都不需要了，因为已经维护到redux中了，



npm i react-redux@5.0.7 -S

注意点，用react-redux来管理状态

1 原来组件的state,和修改state,需要从根组件传的props

都通过props传值，把store的state和dispatch(action)映射成props

如何实现，

export default connect(mapStateToProps, mapDispatchToProps)(AddTodo)

2 在根节点使用这些container组件



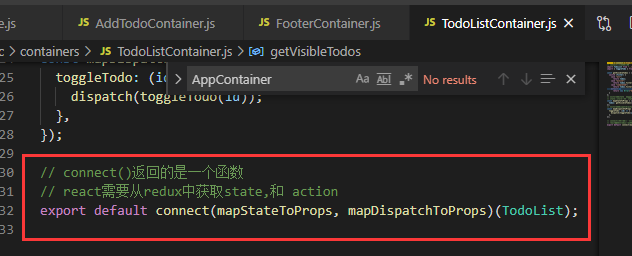
3 在根组件注册store,



4-9 集成react-redux回顾

目前





**连接redux的操作是在TodoListContainer.js这一层，能否给转移到App这个根组件上？**

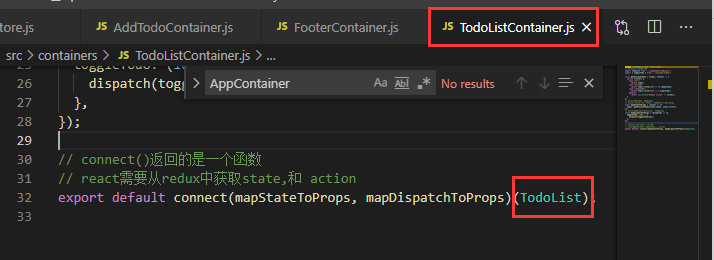
答：不可以。因为react做了很多优化工作，TodoListContainer.js这个组件，只会在依赖的state(即todos和filter)变化的时候才会进行更新，而对其他state( 如 text)变化是没有反应的。

如果说在App.js这一层进行连接redux，就会造成不必要的更新操作，里面的state就包括所有的state,只要有一个变化了，所有的组件就都要更新，text变化，TodoList.js组件本来可以不更新的，也要更新了。

**所以，尽量在底层级组件进行连接redux,保证container组件跟尽可能少的state关联。**

**另一个问题，TodoListContainer组件用到了TodoList组件，是否一定要这样使用呢？**

答案：不一定，当TodoList组件不需要复用，仅仅是给container组件使用时，就可以把TodoList组件的代码给写到container组件里面去。

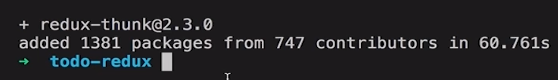


4-10 异步Action

fetchTodos() 返回的是一个函数fun，store.dispatch(fun)是无法处理的，要借助个中间件

redux-thunk(还有其他中间件，有兴趣可以上网找)

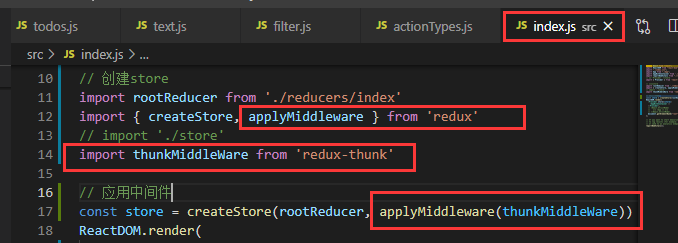




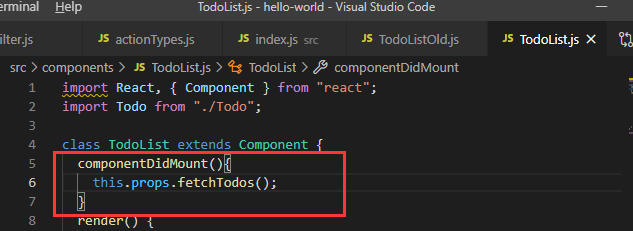
Reducer todos 里的state结构变化

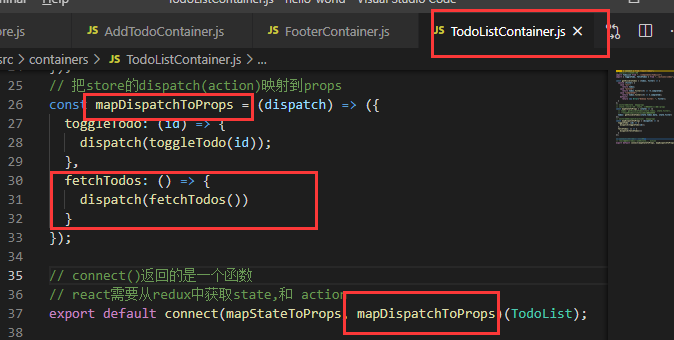
TodoListContainer.js 的mapStateToProps中获取todos的变化

在index.js里把redux-thunk引入，来处理异步



什么时候调用异步的action呢，在TodoList.js挂载的时候

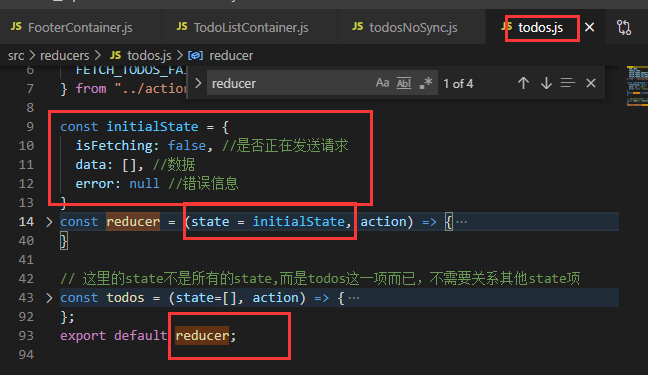




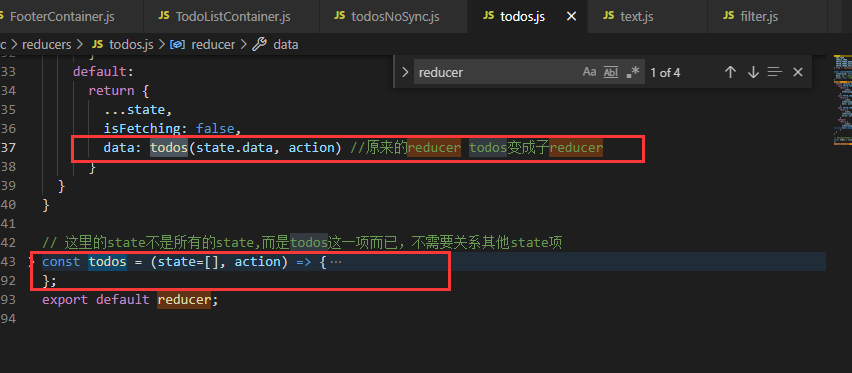
rootState的结构看下面



至于import todos from './todos', 这里的todos，是什么结构，reducer todos.js



可以有子reducer



4-12 Redux调试工具

<https://github.com/zalmoxisus/redux-devtools-extension>

<https://github.com/reduxjs/redux-devtools>



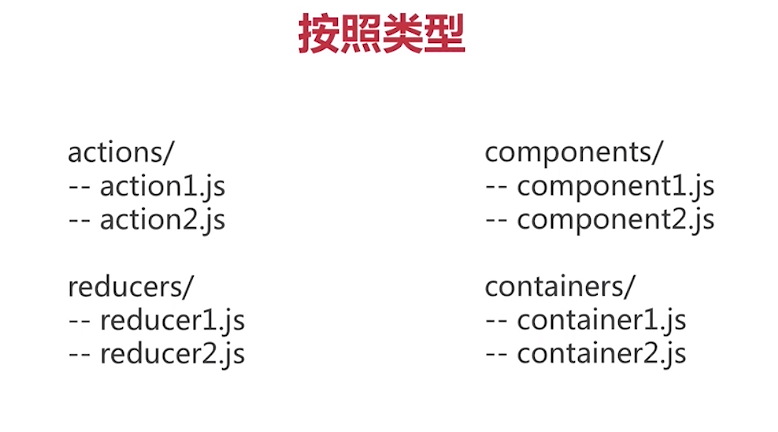
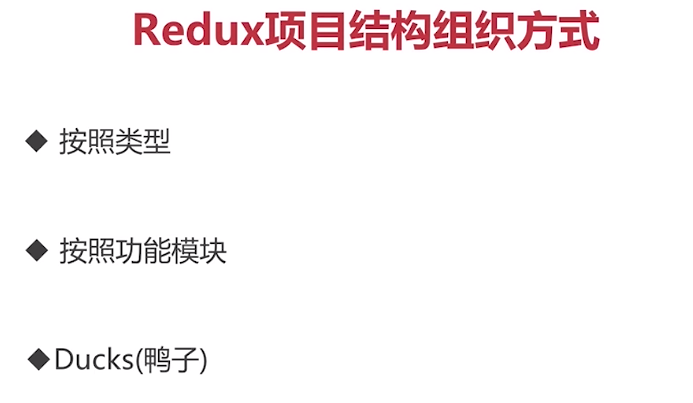
这里重点介绍第一种

<https://chrome.google.com/webstore/detail/redux-devtools/Imhkpmbekcpmknklioeibfkpmmfibljd>

具体如何使用，可以看github上关于这个插件的使用



5-1 项目结构组件方式



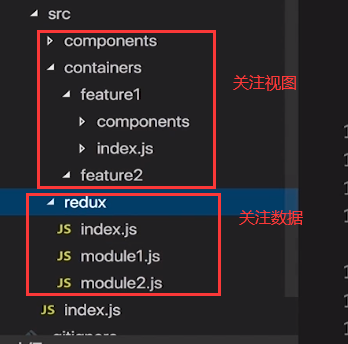
按类型缺点：当需要增加一个功能是，需要动用所有涉及到的原有文件，比如新增个组件，新增个container,改下aciton,改下reducer，不方便。



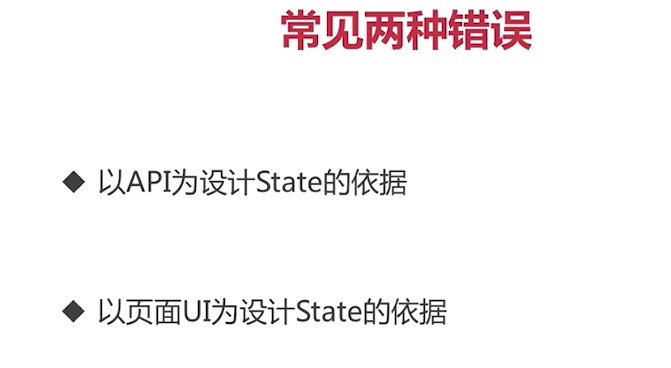
按功能分的缺点：redux的state是全局的， 可能不同的feature的action会相互响应，存在耦合的情况。



比如：



5-2 state的设计原则



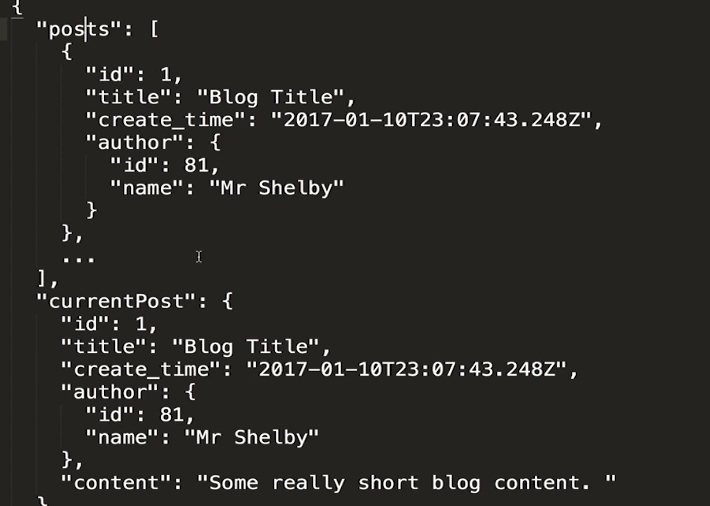
以api为依据，

缺点:

1 不同的state,重复信息太多，比如currentPost的title，createTime,等信息，跟post里面的重复了

2 当我想要找comments里的某条评论时，要遍历这整个数据，查找效率比较低；

缺点根本原因：api的设计是基于服务器端而不是基于前端应用。



以ui为依据

缺点：

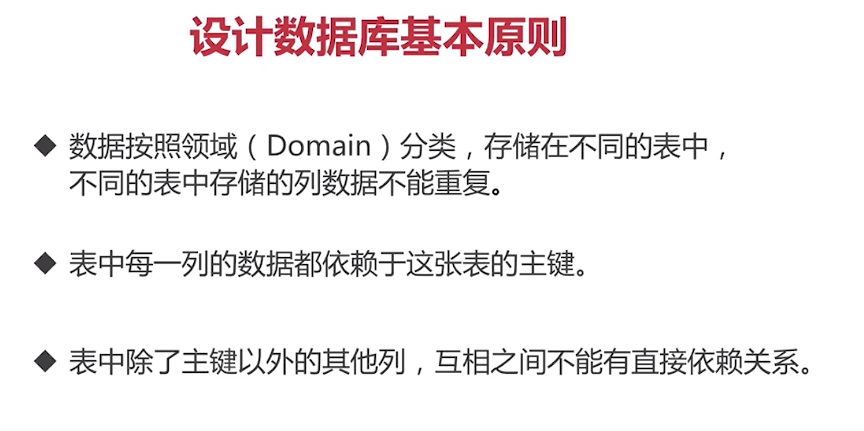
1 存在重复数据，uncompleted,completed的数据在all中存在，

2 当新增一条completed数据时，不仅仅改动一处，在all中也得新增。浪费存储，且有数据不一致风险。

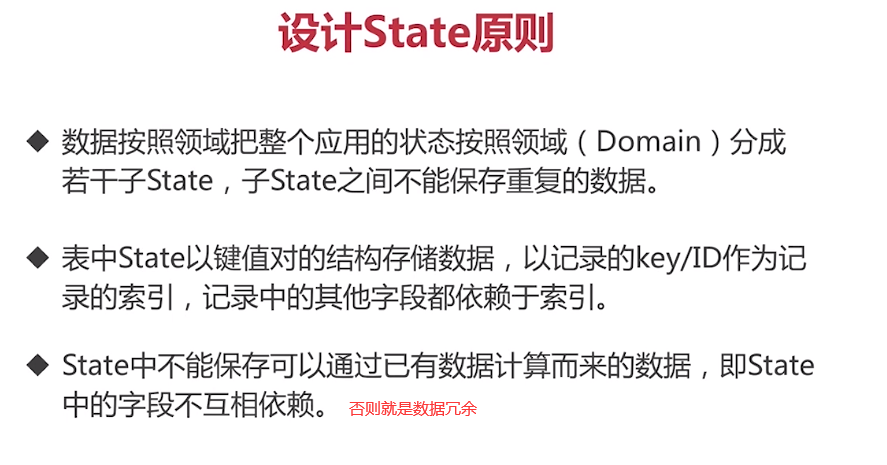


正确的设计state方式





翻译一下就是



例子，把以api为依据的state,重新设计一下



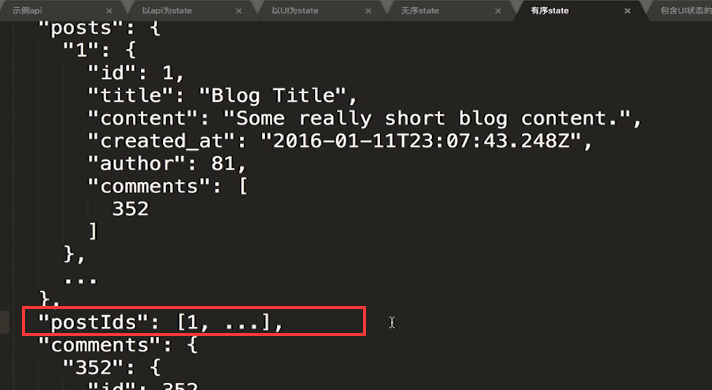
缺点：posts数据给前端应用时，没法保证有序性，

于是再维护一个state, 数组的每一项为posts的key,目的是保证有序性，就是拿key的一个应用，跟key不重复

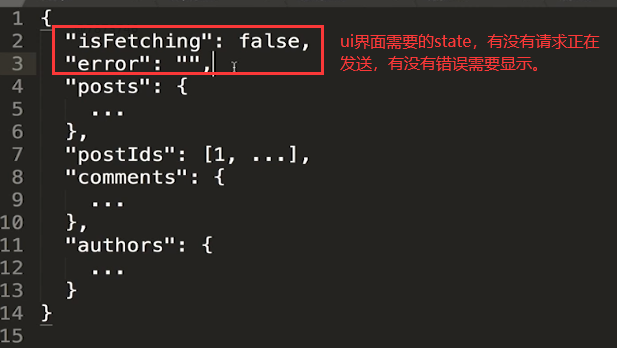
当我们给posts添加一条数据，就需要新增个key，同时要在postIds加个key,

看似有点累赘，但是有一定的道理，这是一种扁平化的设计方式，可以降低访问数据的层级，

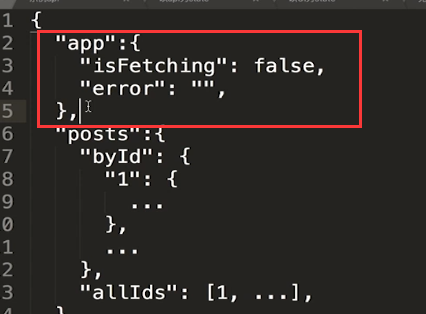
不需要点点点很多次，提高访问相率。



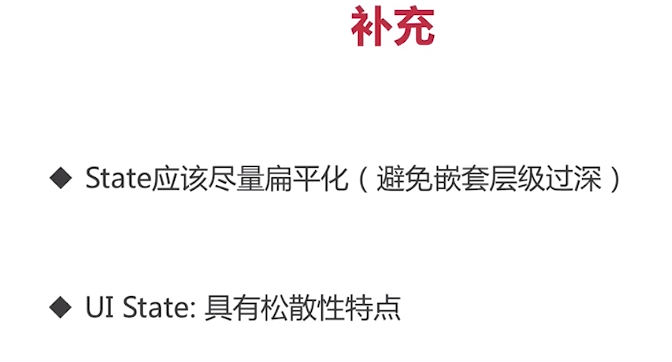
除了这些还有ui层面的state需要维护，比如有没有请求正在发送，有没有错误信息需要展示



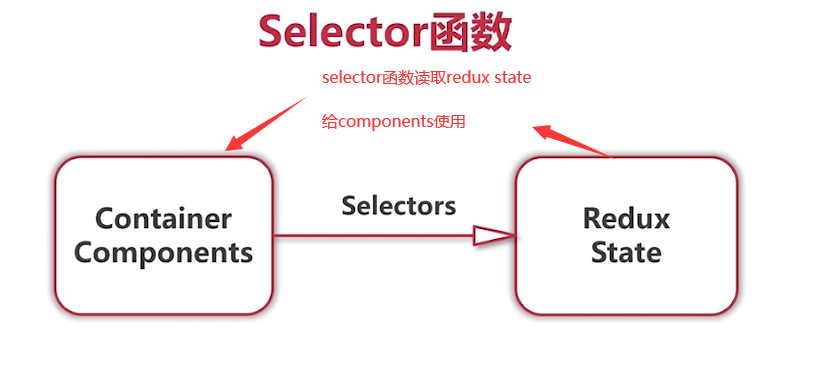
随着应用越来越大，ui界面需要的state就越来越多，而且也比较分散，可以考虑用个state来管理一下



这样设计的reducer就可以减少一些，app这个reducer就可以囊括两个ui的state,



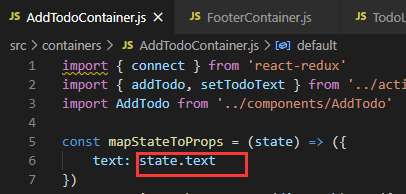
5-3 selectors的使用



目前获取state中 text，都是通过state.text,如果将来不想放到text下了，放到data下，

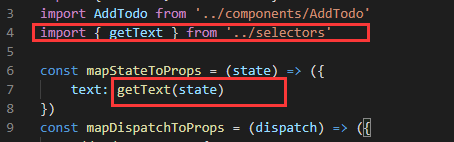
那岂不是要改很多个地方。

如何解决：



用selector函数





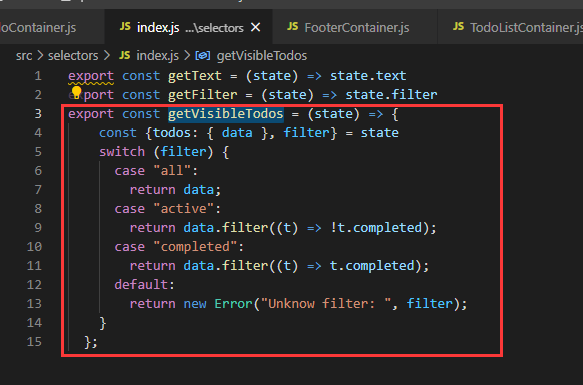
作用：

redux是状态层，对它的访问应该是通过api的方式，告诉别人有哪些api,

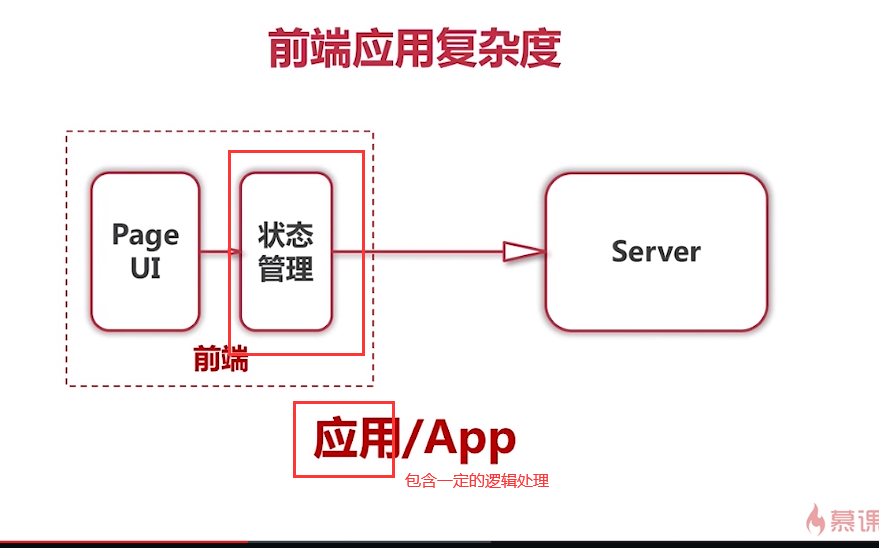
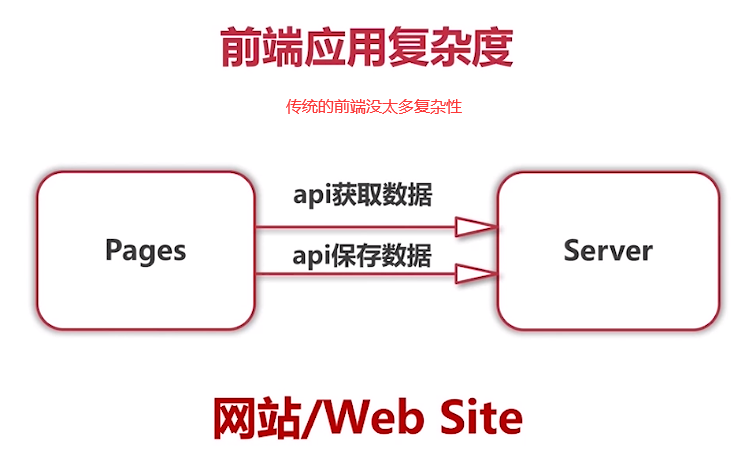
react是view层，

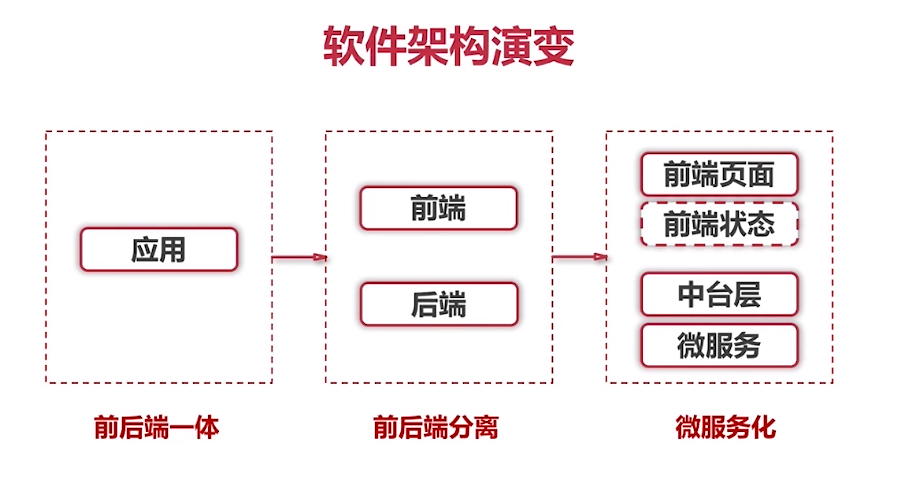
select实现了两个层之间的解耦。

Selector不紧可以读取state，而且还可以对读取到的state进行数据处理，返回对应的结果给相应的地方。getVisibleTodos

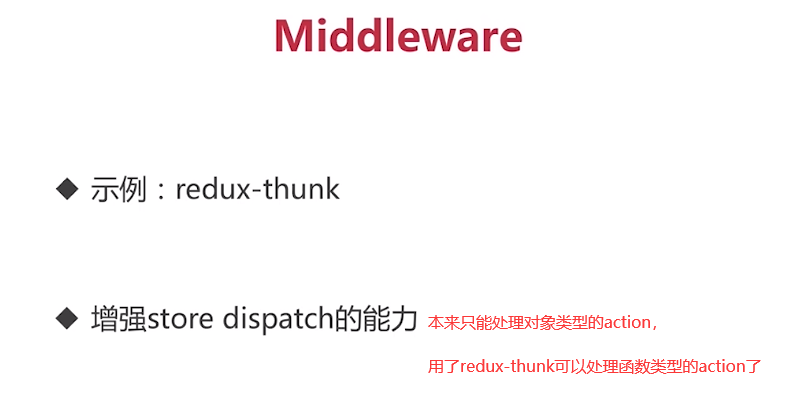


5-4 深入理解前端状态管理思想

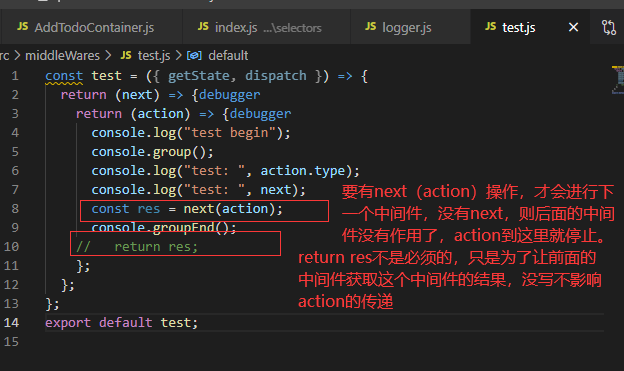


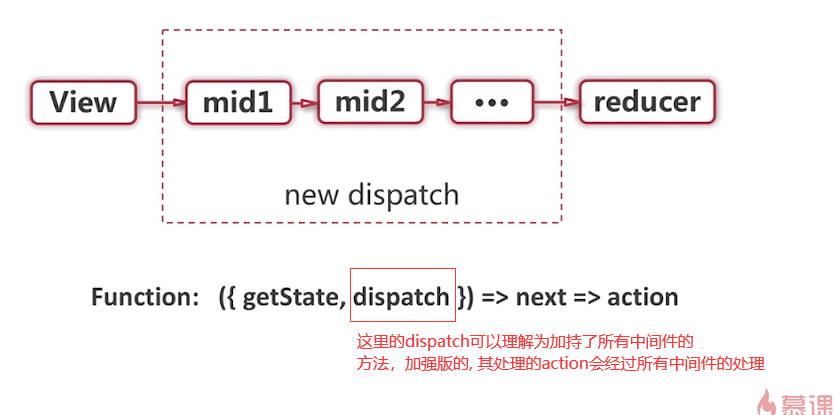


5-5 middleWares



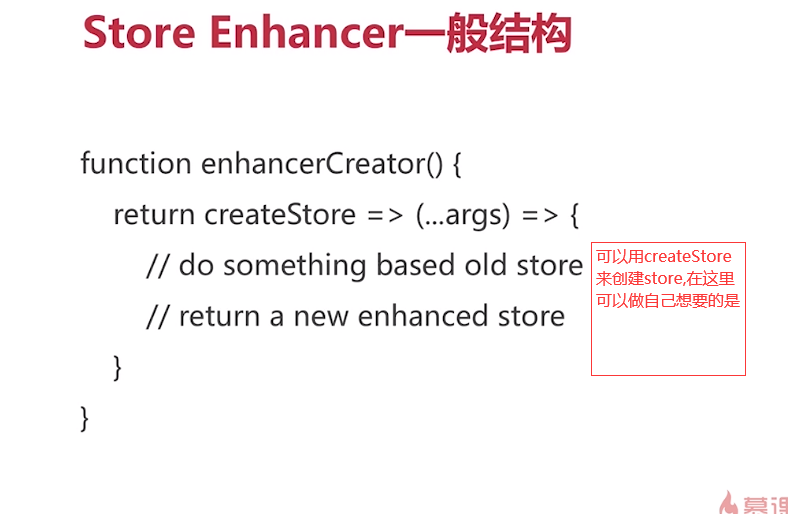


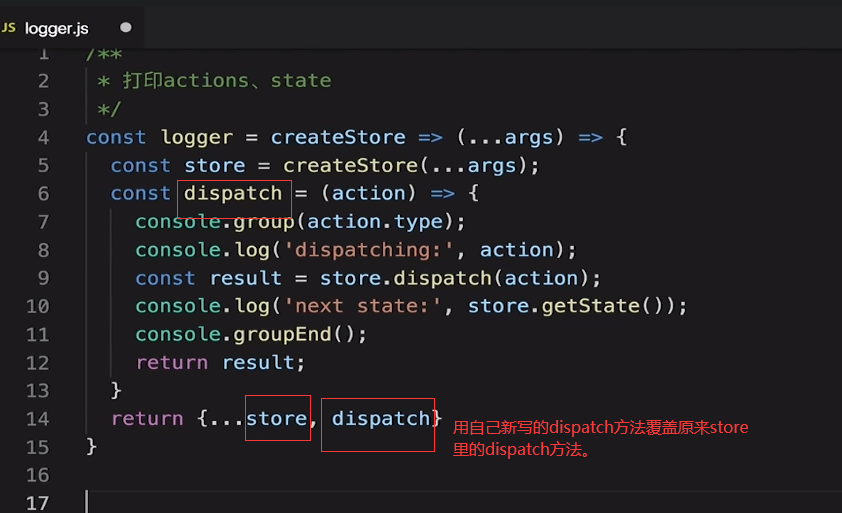




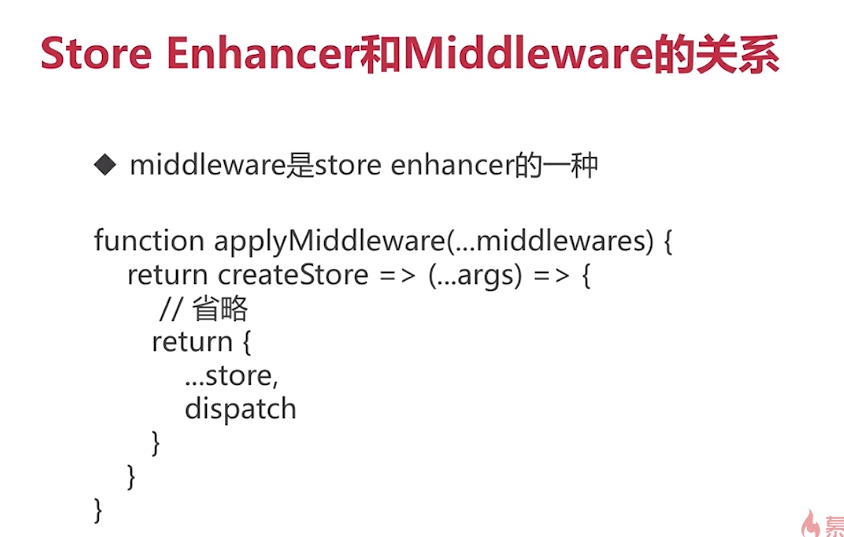
5-6 store enhancer







Store enhancer 和 applyMiddleWare都能获取action并处理，两者有什么区别



但是实际上应该多使用middleWare，少使用store enhancer, 因为store enhancer的过程，会对store的属性进行一定程度的修改，getState，subscribe等方法要是被重写，有可能在一定程度上给原来的api造成底层逻辑的破坏。所以要多使用middleWare.

6-1 客户端路由和服务端路由

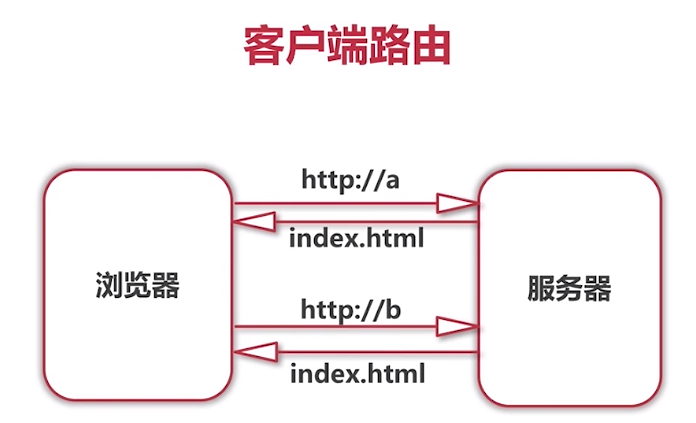
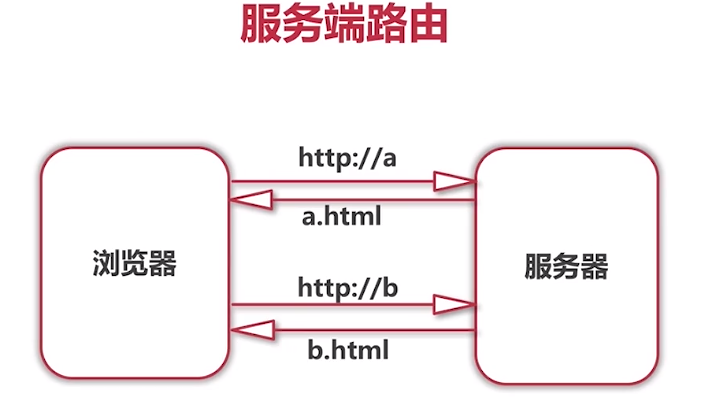




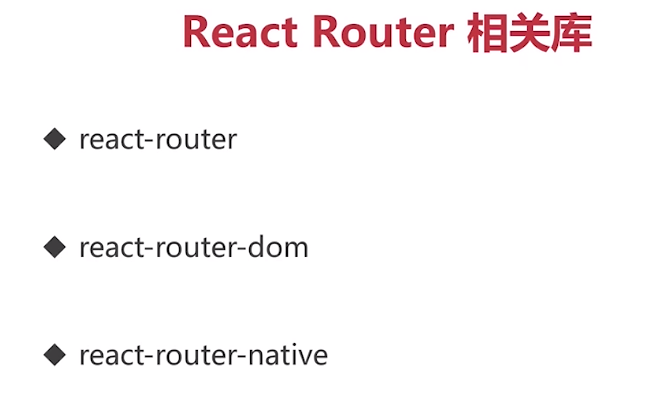
优缺点

多页面应用：每个路径对应不同的html，便于爬虫爬取，方便seo;

单页应用：不同的路径对应相同的html，不便与爬虫爬取，不方便seo;页面切换的时候，不需要重新请求js，不会有白屏的过程，用户体验稍微好一点，路径切换，不需要重新请求js，性能会稍微好点。

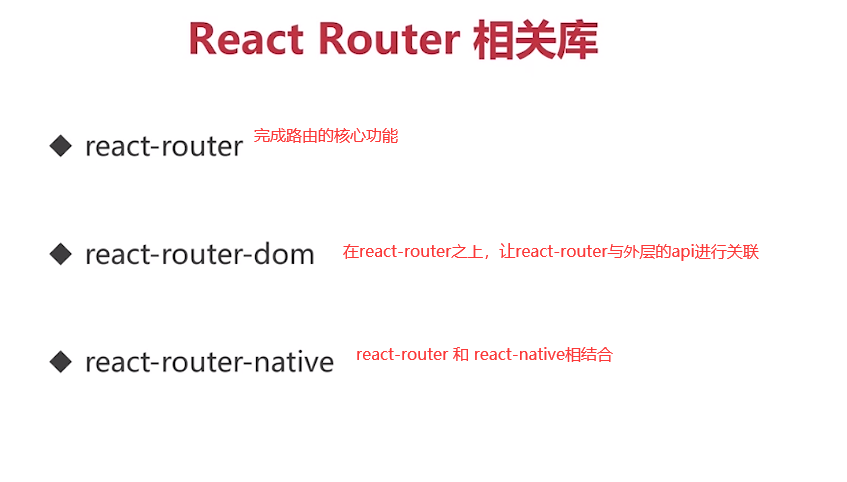


6-3



React-router-dom把react-router和web api进行绑定，让我们可以在web项目中操作react-router，我们这里是web项目，所以用react-router-dom这个库。

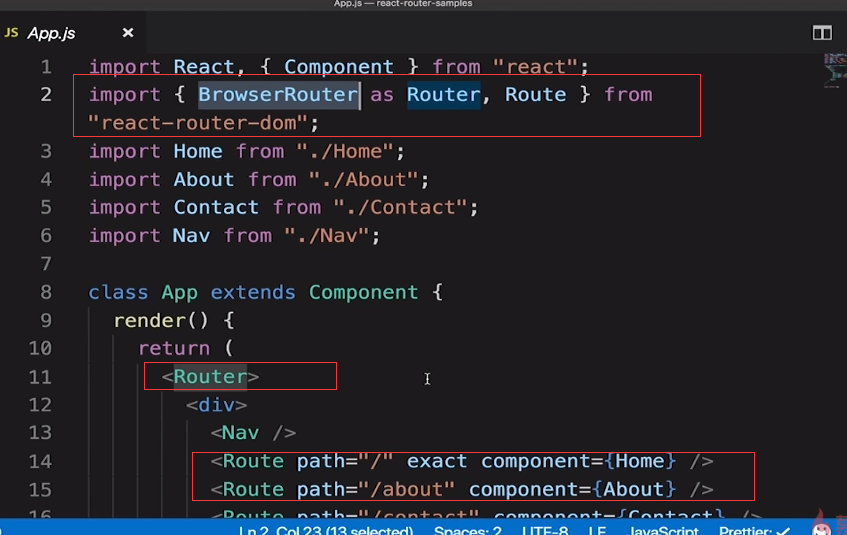
Npm i react-router-dom -S

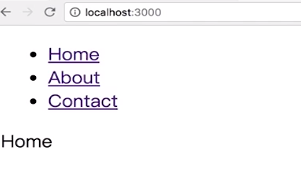


要保证所有的ui组件都能有react-router的功能，需要在跟组件外层包裹一个router组件。



具体例子







额外配置： 一个错误路径，不返回404，而是返回index.html

Hash router 用的是hash值。



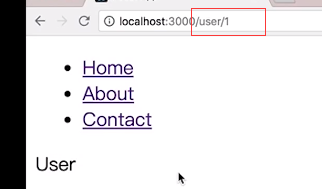
效果



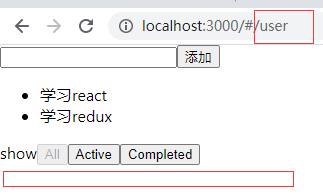


6-3 路由配置

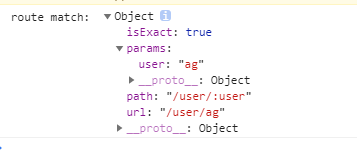


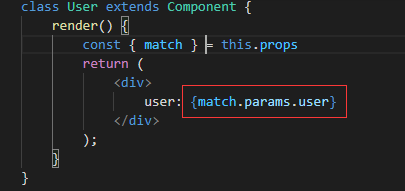


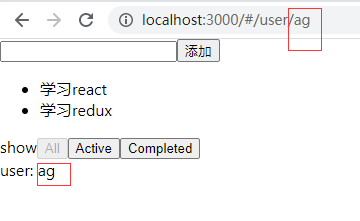
但是/user就匹配不到了



可以在组件内获取match，通过match.params.user获取对应的参数

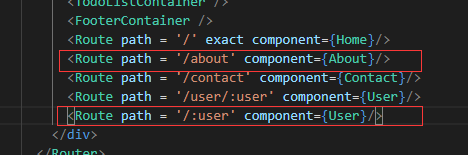


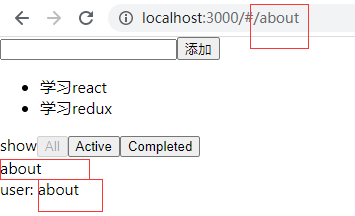




6-4

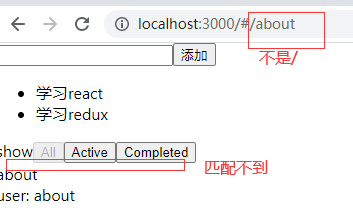
react 每个route都会跟url进行匹配，成功了就显示



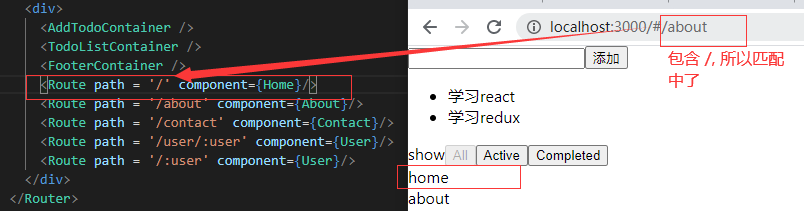


exact, 与路径完全匹配



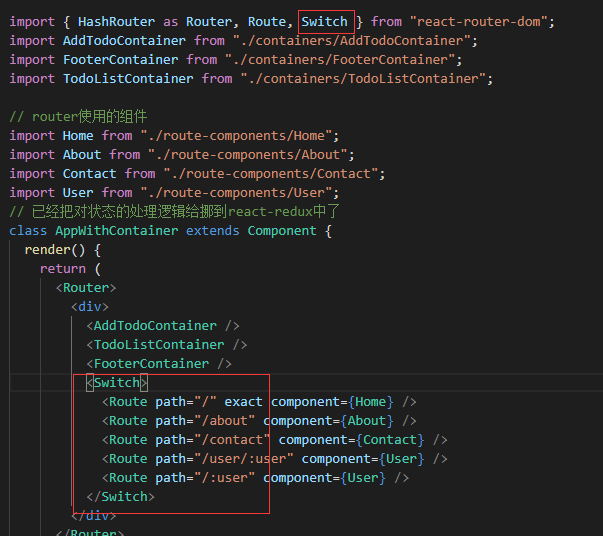


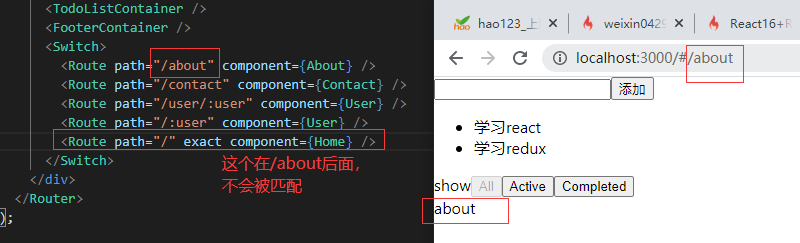
没有exact，路径包含path即可



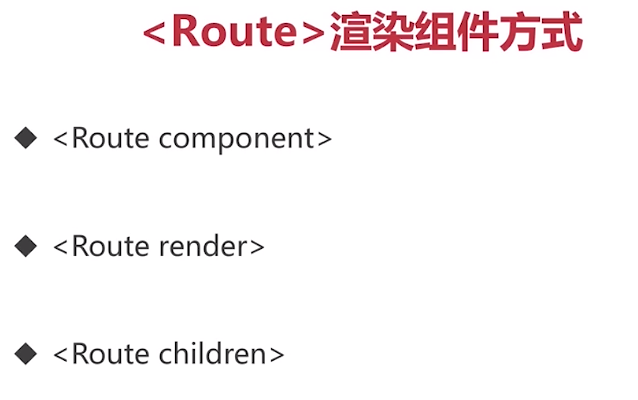
Switch

Switch里面第一个Route匹配中之后，后面的Route就不再匹配了。





6-5 路由渲染组件的方式



第一

component可以用一个组件，用一个函数也是可以的，但是每次重新渲染的时候，就会重新跑一下这个函数，

会导致这个组件先卸载，再生成个新的组件，内部的状态丢失。



第二种

用render代替component好处

1 render接收的函数，生成的组件，不会在重新渲染的时候卸载生成的组件，会复用，会保存之前的状态；

2 接收的props里面包含了match,history,location等属性，可以传给组件去用，组件用这些属性，做更强大的功能；





3 还以传其他的属性给组件，比如{...extraProps}



第三种

children={(props) => (

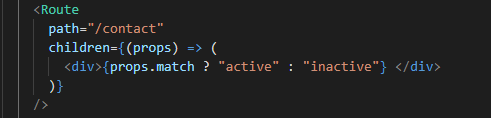
<div>{props.match ? "active" : "inactive"} </div>

)}

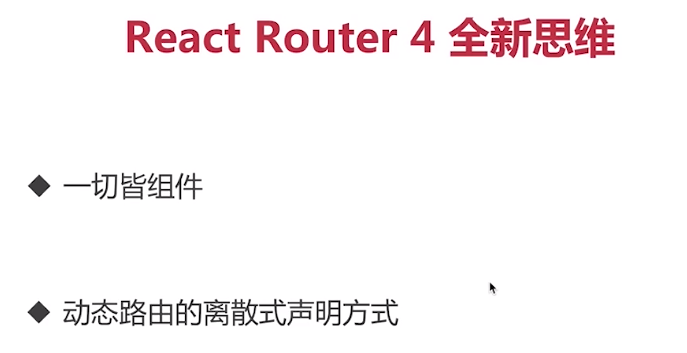
不管路径是否匹配，都会渲染，只是props.match值不一样。

当路径不匹配时，props.match是undefined,但是会渲染，显示inactive

当路径匹配时，props.match为对象，这时显示的是active，



6-6 React router4 思维



相较之前的版本有思维方式的重构，

1 遵从react 一切皆组件的思想， 之前的<Route/>组件是没有render方法的，之前的版本不算是个真正的react组件； 每个<Route/>都会被渲染， 没有给路径匹配上也是会被渲染的，如果匹配上，<Route/>就被渲染成相应的component, 如果没有匹配上，就被渲染为null,

即一个空的react组件，

2 移除了之前关于router的声明周期，完全使用react的声明周期；

3 动态路由，可以离散式使用。

之前版本的路由一般都是用一个route，把所有的路由都集中写在一起，页面一加载的时候全部加载下来，是静态路由。 react router4 可以不这么写,可以在任意需要的地方写<Route/>,



