搭建react

1. 搭建node
2. npm install -g create-react-app 下载脚手架
3. create-react-app todolist 创建一个react项目
4. Npm run start 运行成功

项目介绍

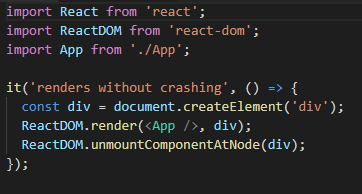
项目介绍，可以自己随便编写

自己找一个小图标改成这个文件名会覆盖以前的形成新 的图标

容错，禁止使用<script>除非你 重新使用js的适配项



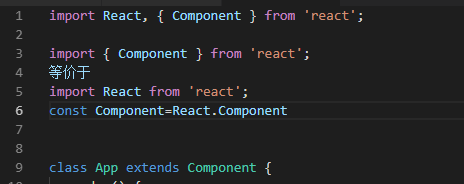
写移动端可以变写网页一样，而且可以当离线的时候，再次进入这个页面还是可以访问 进去 是一种PWA的理念



做自动化测试的文件，（写函数的时候）

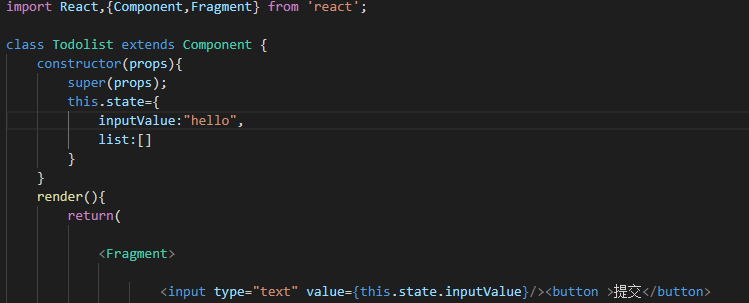
pwa 的时候用的配置项

组件

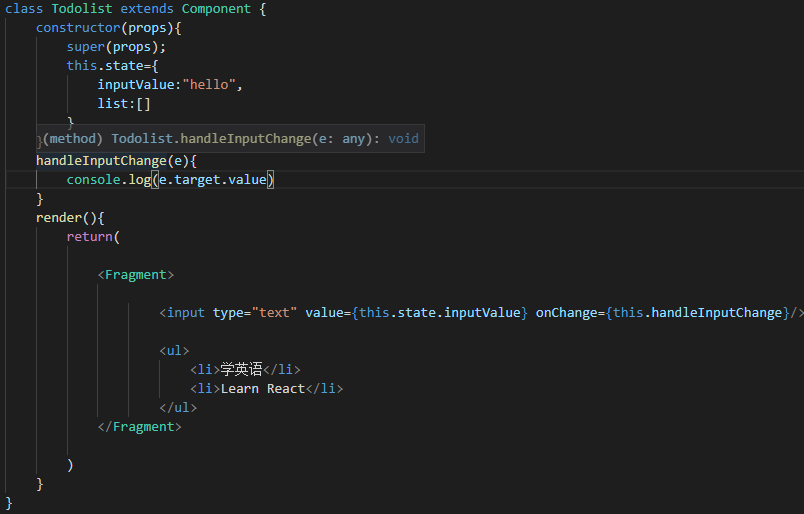
 上面那样写了 下面就直接 Component就行了



<Fragment/>组件能代替最外层的div 这是占位符，在显示时等于少嵌套一层div但是有 符合react的编码规范

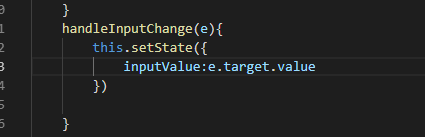


This.state如果固定了值，在组件中用this.state时是固定的，修改不了，在text框修改 也改不了。但是value的值其实已经改了只是显示不了

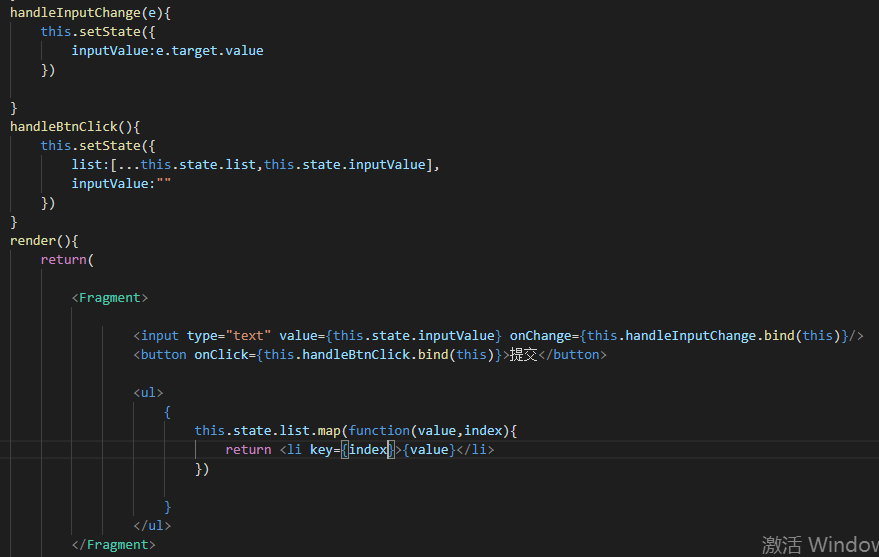


OnChange={this.fun}绑定函数就行了，但是这个this指向是undefind，所以还是需要 this.fun.bind(this)来重新绑定到组件上，e.target指的是 

e.target.value指的是input的值



通过this.setState({})来修改数据



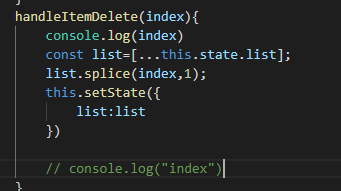
只要是有循环遍历的DOM都可以采用数组的map方法遍历，遍历ul时候，li中会要有key值来保持不同，不然会有警告。

这个的效果的思想是表单设置的值是放到this.state的属性中，由于this.state绑定了数据，所以不能更改，这时候用onChange事件定义一个函数，函数里用this.setState更改表单的value值，当提交的时候定义一个单击事件就用[...this.state.list,this.state.inputValue]改变这个值。这个的意思是，onChange事件导致按下一次输入一次值，所以如果我想把值都赋予到一个li里面，我需要把他放在一个数组里，然后数组的一个集合。



这里就用得分es5和箭头函数的区别了，this指向由于箭头函数的this指向，所以指向li了，如果是es5的话bind（this）绑定的就是组件了，所以要常用es6啊

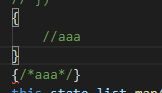
Bind(this,index)还可以绑定索引到函数中 fun（index）就行了

然后拷贝一个数组，在this.setState中删除就行了，

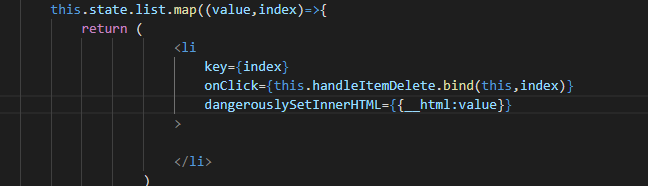
Ps 只要改数据必须在这个里面改，别在this,state里改完再放到this.setState中，不推 荐

Immutable概念里面不允许我们在state中改变，如果要改那就要像上面那样拷贝出一个副本出来，然后再修改。

这样不允许

注释有两种

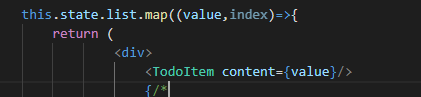
为什么一般不使用class而是用className 因为class和继承的class同名



写完这个之后（记得双下划线）可以在表单里输出html格式时自动编译，但是不太好，容易受到攻击。写了之后那么标签中间的value就要删了



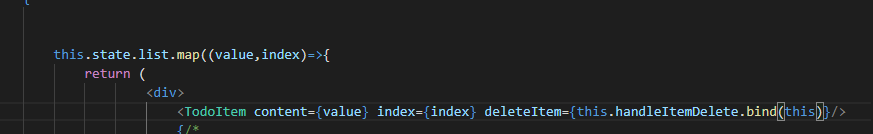
这个for 不能用for因为会认为是循环，会有警告，用htmlfor

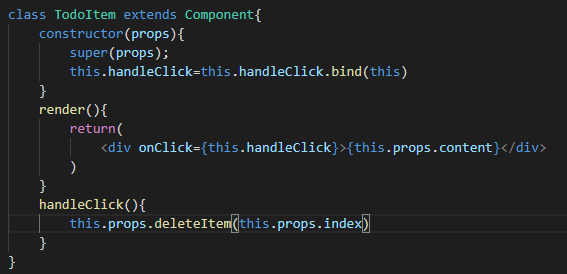


父传子组件传值时，不一定非要用this.state.属性名，感觉只要是个变量就行，这例子就是一个形参代表当前数组的值，通过属性来传递



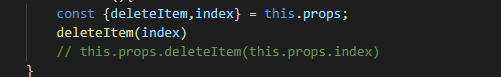
这样写绑定函数性能更好，在constructor函数里面绑定 这样绑定是组件初始化时，就直接绑定好this



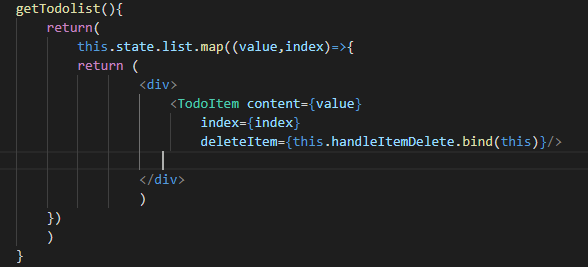


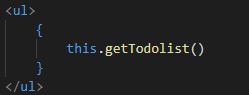
子组件传给父组件值时，父组件给子组件传方法是必须要强制bind(this)绑定到父组件上，不然会报错

**代码优化**

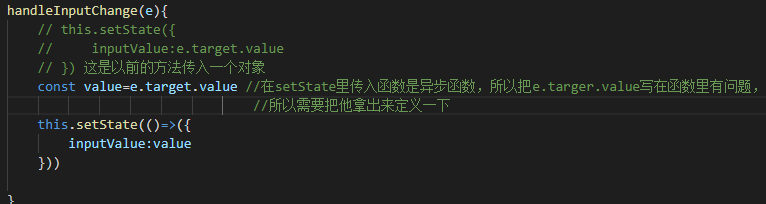


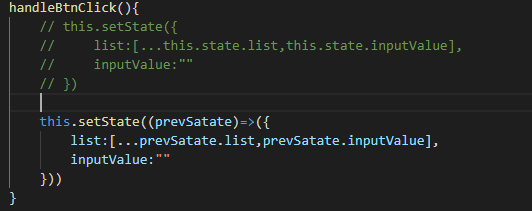
利用解构赋值将this.props.属性名 获取过来，更利于代码优化



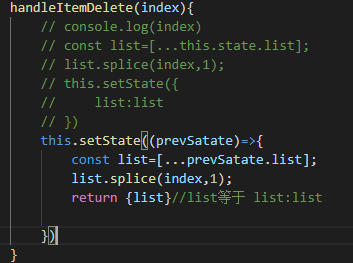


将ul内的遍历封装在一个函数里面再调用函数，可以让render函数内更简单，维护方 便。





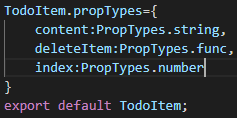
PrevState是指之前传过来的数据，可以代替this.state,性能更好，避免不小心改变state的状态



由于setstate可以返回一个函数，所以可以吧外面函数定义的东西都放到这个函数中，在

es6中只写list的话 等于list : list

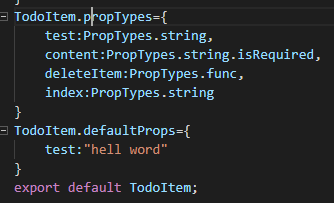


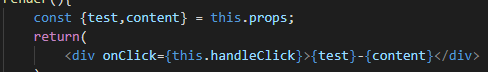




引用的propsTypes 可以规范父组件传过来的值得类型，所以这个放在子组件中的，

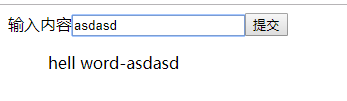
IsRequired是必须传的，不然会给警告





DefaultProps可以设置默认值，即使父组件没传属性也可以在这里直接写就行，传的时候还是默认传到this.props中

这个东西可以用来当你输入一个东西的时候，前面你子组件有个默认传值，给你前面加上去

Lg:



两者中其中一个



接受数组格式的数据两种都行





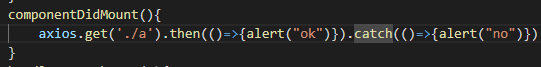
一般不推荐使用ref，因为它可能操作DOM

声明周期函数是指某一时刻组件会自动调用执行的函数

声明ajax最好放在componentDidMount 其实放在componentWIllMount 或者构造函数都行，因为他们都只执行一次，render（）就不行，每次重新渲染执行，导致重复。其中componentWIlMount一般别放，因为在什么服务器重构会出问题。

需要下载 axios npm i axios命令查一下就知道了

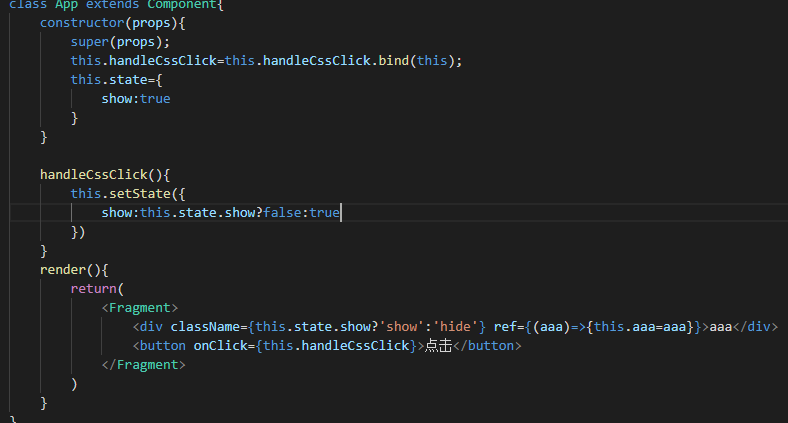






调用ajax

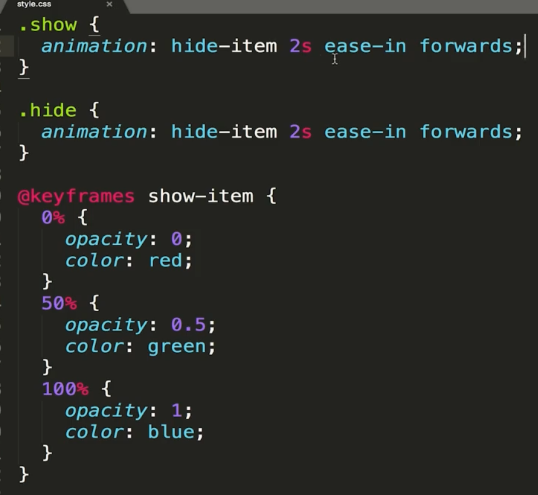
Css过渡动画



记住，react不管做什么都是尽量操作DOM，就比如这个。如果我们需要修改class类名的话，一般用三目运算符在行内直接操作判断，用一个布尔值做标记，来确认使用哪个类名，

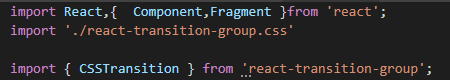
当修改的时候用一个函数里面的this.setState来修改做标记的布尔值就行了，只要布尔值修改，那么就可以决定使用哪个类名

Css动画效果

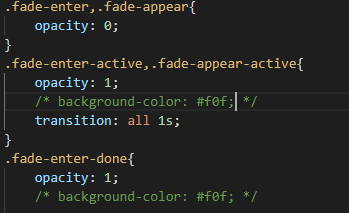


通过forwoards我们可以做特效就一般可以用css3来写了，方法还是上面的标记法和三目运算符

可能我们需要的特效还要牛点，所以需要借助第三方插件 react-transition-group在gitHub里搜索下载npm install react-transition-group --save







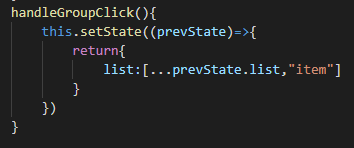
这时候就不需要在标签上用三目判断用那个了，但是还是需要设置show标记，因为点击的时候可以判断用那个。

这是钩子函数使用

React-transition-group的

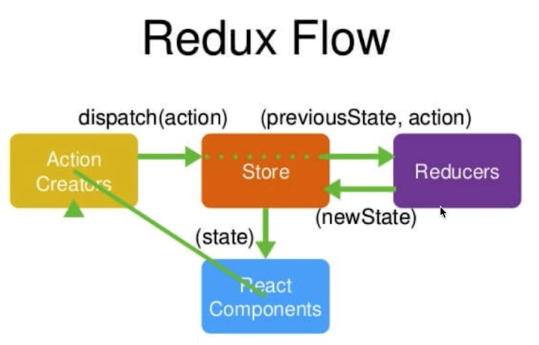






Redux

工作流程

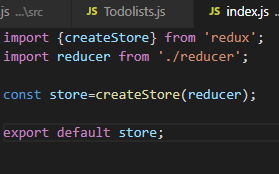


Store相当于管理员

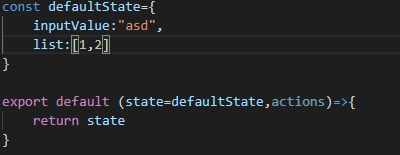
Reducers相当于管理员的笔记本，东西都在这，管理员知道干嘛的

Action Creators相当于借书的人说的话

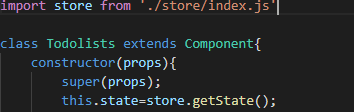
在src目录下创建store文件夹，在里面创建index.js 和reducer.js



Index.js 需要在redux中引入createStore方法，定义一个变量，让它等于这个方法，这个方法里面有个参数时reducer，这是从reducer.js文件中导入过来的一个函数，返回state。



Reducer.js无需任何引入，他只是定义个变量来存储数据的，导出一个函数，有两个参数，一个是state一个是actions。



在组建中引用可以将reducer.js的值调出来，赋予this.state就行了，然后就和正常操作一样了

调试redux 下载：谷歌商店redux-devtools-extension，但还不能用

F12下面有个redux，然后点击进入文档会看到一句话



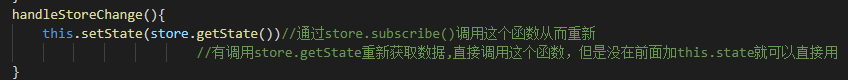
加了就行





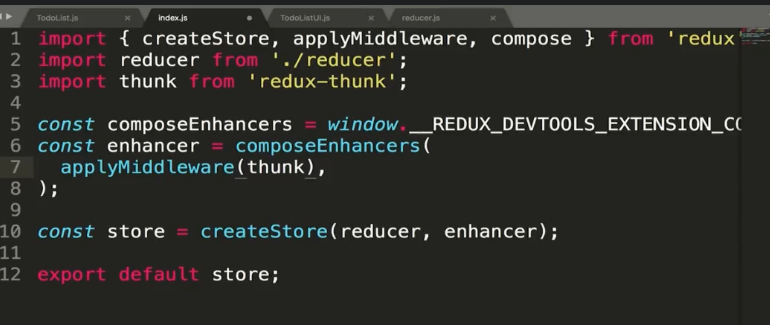
在事件函数中，我们要将我们需要改的数据放到action里面，action是一个定义的对象，两个参数type，value，然后store.dispatch(action)会通过这条语句会告诉store要改的内容，store接受到后，会自动的吧原来的内容和改的内容重新发送到reducers中，然后在reducers里面重新修改，修改完之后返回给store，组件又通过subscribe方法监听store再把新数据调过来





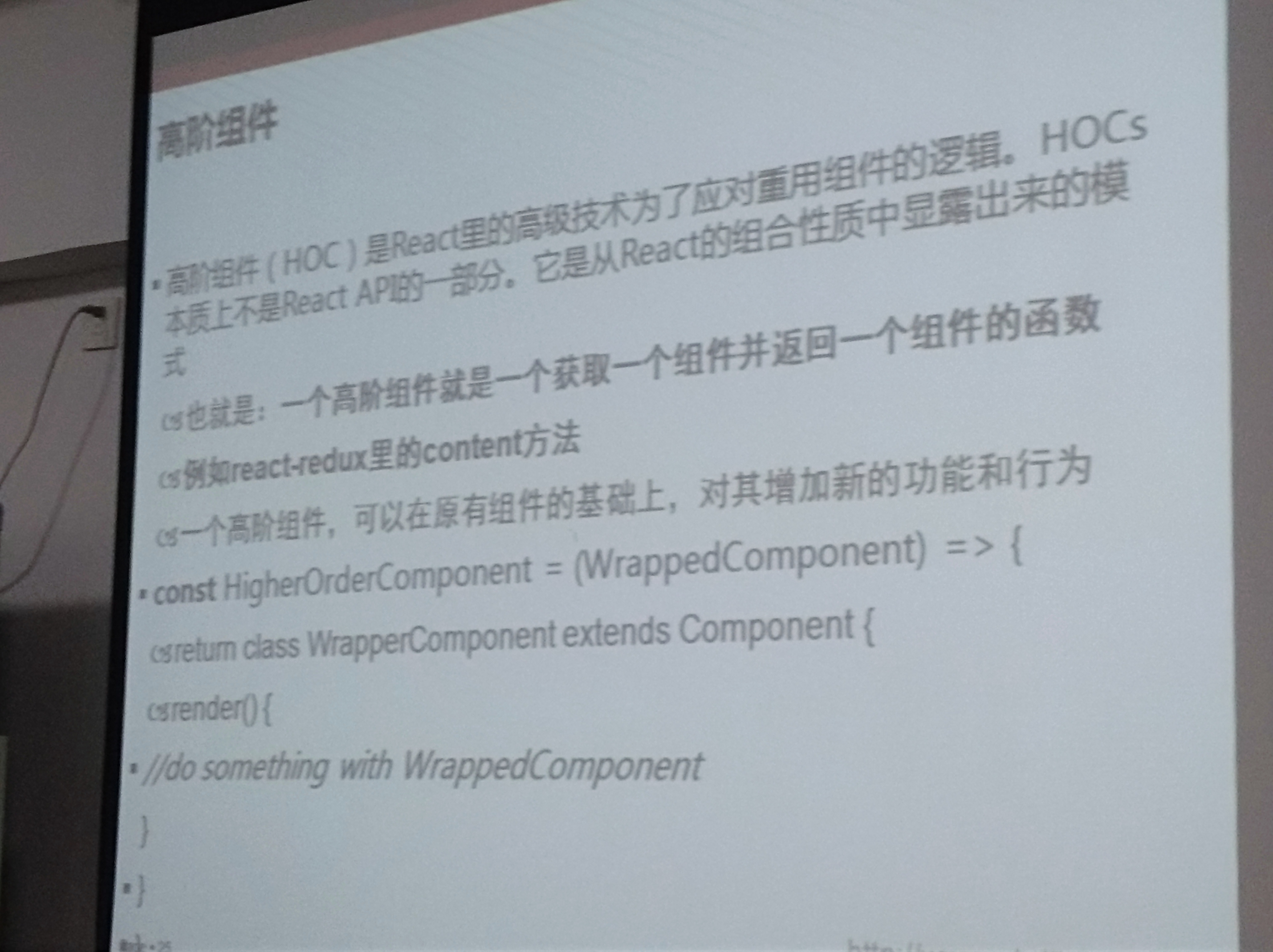


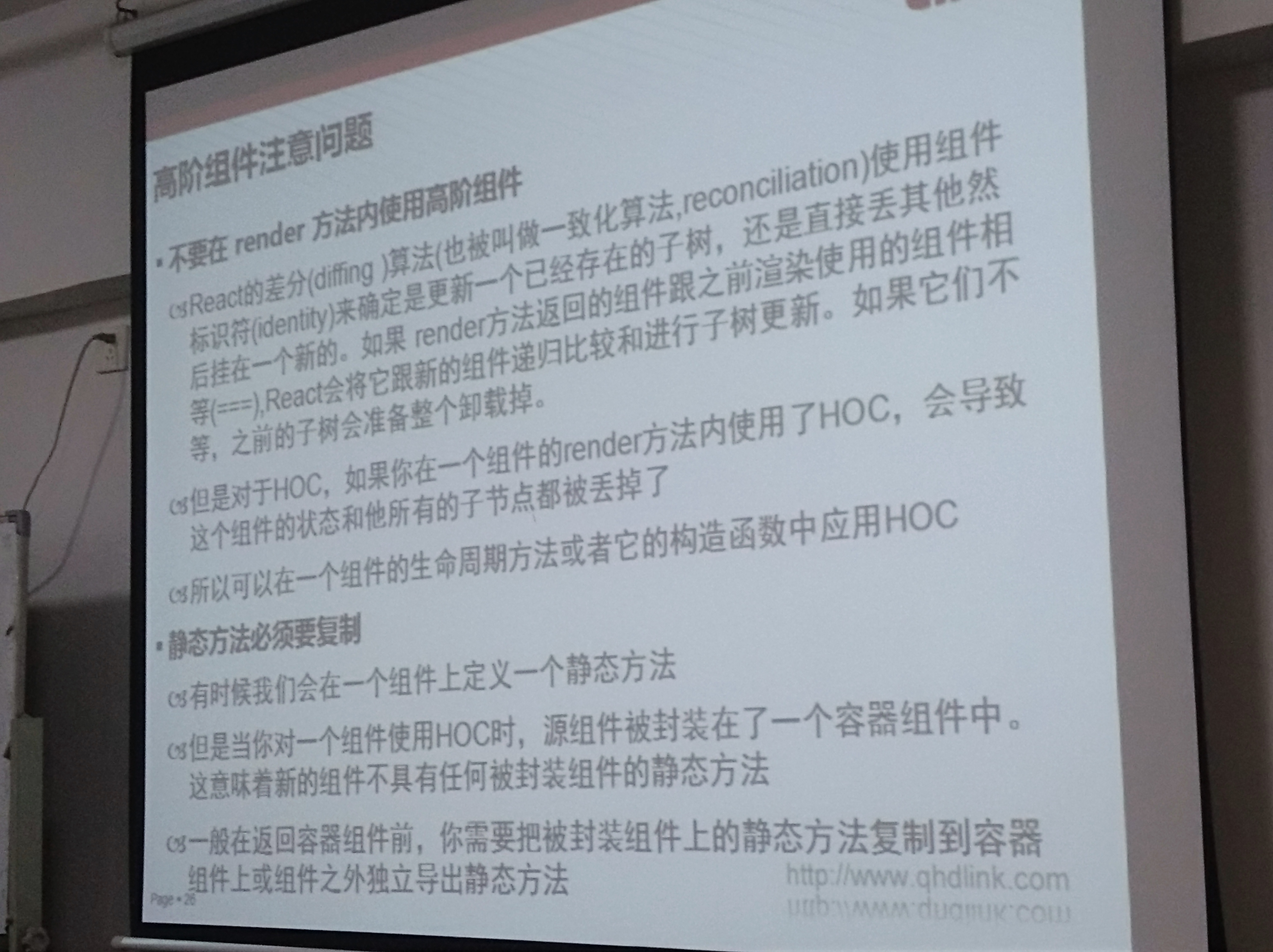


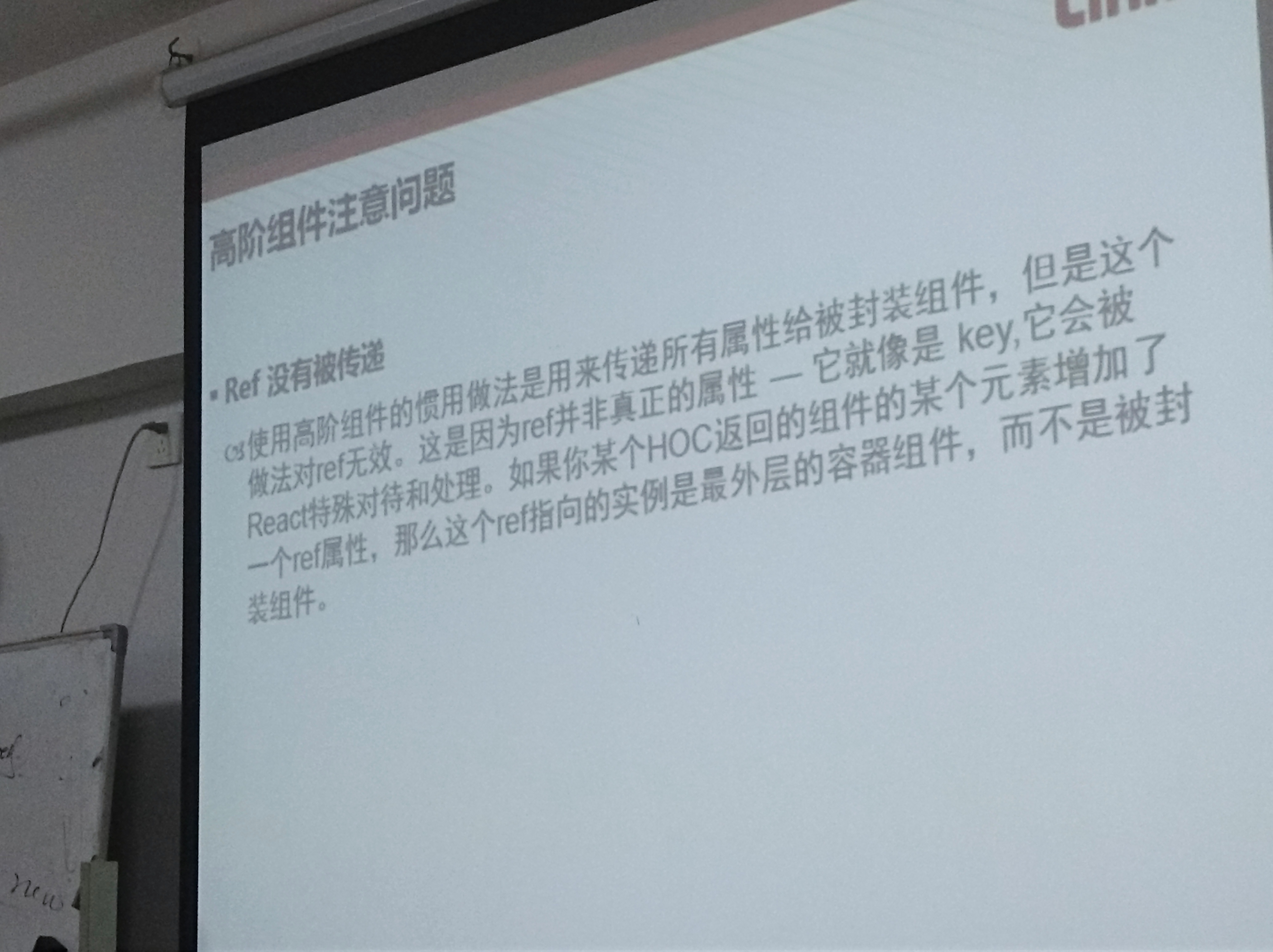


Redux-thunk能和devtools一起使用的方式，github里面搜索devtools 往下翻 1.2的

高阶组件







//withPersistentData就是一个高阶组件，它返回一个新的组件，在新组件的componentWillMount中统一处理从LocalStorage中获取数据的逻辑，然后将获取到的数据以属性的方式传递给被包装的组件WrappedComponent，这样在WrappedComponent中就可以直接使用this.props.data获取需要展示的数据了，如MyComponent2所示。当有其他的组件也需要这段逻辑时，继续使用withPersistentData这个高阶组件包装这些组件就可以了。  
function withPersistentData = (key) => (WrappedComponent) => {  
  return class extends Component {  
    componentWillMount() {  
      let data = localStorage.getItem(key);  
        this.setState({data});  
    }  
   
    render() {  
      // 通过{...this.props} 把传递给当前组件的属性继续传递给被包装的组件WrappedComponent  
      return <WrappedComponent data={this.state.data} {...this.props} />  
    }  
  }  
}  
   
class MyComponent2 extends Component {    
  render() {  
    return <div>{this.props.data}</div>  
  }  
   
  //省略其他逻辑...  
}  
   
class MyComponent3 extends Component {    
  render() {  
    return <div>{this.props.data}</div>  
  }  
   
  //省略其他逻辑...  
}  
   
const MyComponent2WithPersistentData = withPersistentData('data')(MyComponent2);  
const MyComponent3WithPersistentData = withPersistentData('name')(MyComponent3);