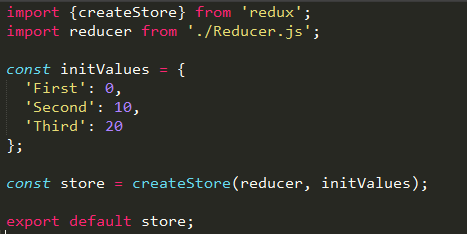
【关于redux如何起作用】

由于描述起来比较复杂，本笔记只做提醒作用。直接看看不懂。

首先，store提供一个创建store的方法creatStore，这个方法一般接受两个参数，第一个参数是reducer函数，第二个参数是store的初始值（初始state）

如下



InitValues是初始值。我们可以不管初始值里面是什么，理解这个也很简单。

但是我们要理解reducer。

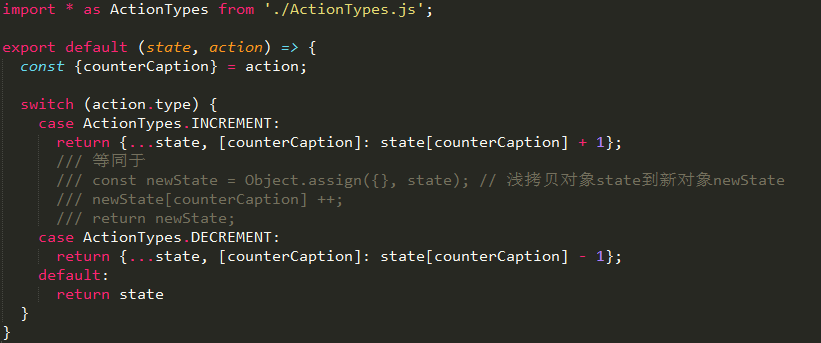
Reducer是一个纯函数。纯函数的意思是：函数不改变任何参数。

这个函数会对所有要更新store数据的行为做修饰，也就是说所有要修改store的数据都要先经过这个函数。

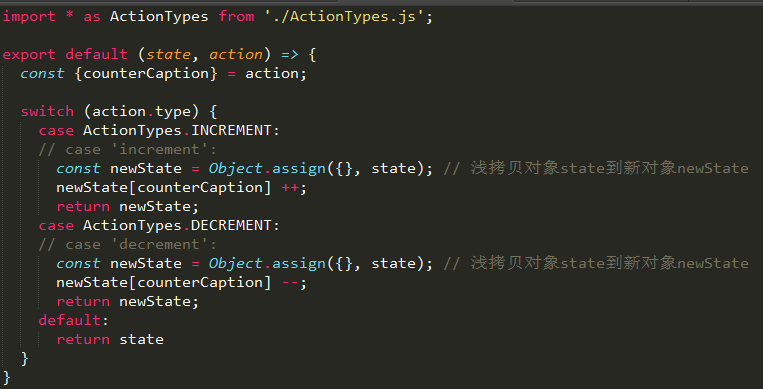
Reducer函数的目的是，不改变原有state，每次改变数据，都返回新的state。

一个state对应一个新的组件视觉。

Reducer函数如下



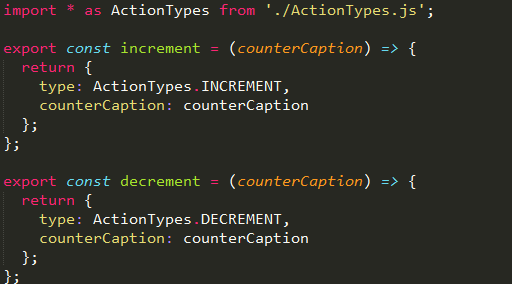
这种写法太装逼了，我更喜欢如下这种写法。



传入的参数，一个是state，一个是action。

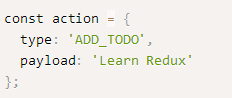
State参数就是原本的store的state

action参数是一个对象。这个对象一般写成这样：



（这个文件叫action.js，以上这两个函数根据调用的地方调用Action.increment()或者Action.decrement()来获得对应的action对象）

或者这样：



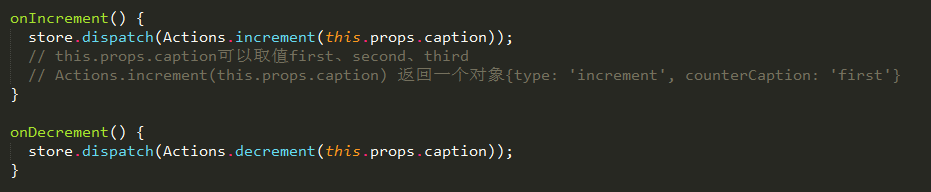
关于action怎么设置有一个社区规范（英文）

https://github.com/redux-utilities/flux-standard-action

ActionTypes这个js可以暂时不管。

然后到了组件中。

组件中定义了两个方法来改变store中的值



通过store.dispatch来通知store改变store的state。

Actions.increment(this.props.caption)获得一个对象，可能获得如下对象：

{

type: ’increment’,

counterCaption: ’first’

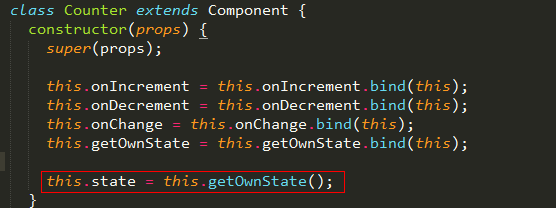
}

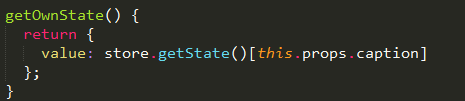
这个对象在reducer函数中就是action参数。

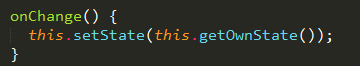
这样，就对应上了reducer函数中的行为。

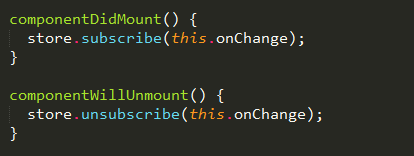
**以下是不规范内容，可以不记，了解原理即可。**

既然知道了组件如何改变store状态，那如果store的数据改变，应当也可以改变组件的状态才对。









我们回顾一下组件的生命周期，componentDidMount是组件渲染之前的函数，componentWillUnmount是组件即将卸载前的状态

Store.subscribe是给store添加一个监听，假设store.state改变，那么执行这里面传入的函数，也就是this.onChange

在组件卸载的时候把这个监听删去。

这样一旦store.state改变，就会通知这个组件改变，执行this.onChange函数

这个onchange函数在这个例子里正好是改变当前组件的state的函数。当然也可以做别的。

**不过，这样做并不好。因为对于一个组件来说，他应该获取唯一的数据源，只进行数据源的渲染形成view**

**这样做，为每个组件都添加了一个state监听，不利于组件间通讯的简洁。**

**正式的解决方法在下面**

【关于context】

规范上来说，不应该允许所有组件都引入store。因为大型项目store可能很庞大。

而且，如果所有的组件都和store通信，那么组件既要给store发送数据，又要从store中监听、获取数据，非常繁杂。

组件繁杂不利于维护，而且会让项目越来越复杂。

所以

最好的方法是让一个根组件提供store，然后所有的子组件（其实就是这个项目的所有组件）都可以通过类似以下代码来获取store中的所有数据

