js

React复杂对象的更新 - 不用React官方推荐第三方lib解决方法

React 官方推荐Immutability help 解决复杂对象的更新问题。我们把复杂对象和更新先理一下。

更新

React 处理对象更新的时候，只会在对象内存地址更改的前提下才会更新视图。如果只是更改对象里面的一个值或者对象里面的一个属性，那么视图不会被更新。

复杂对象

js 里面的对象可以有非常多的种类，比如，数组，对象，Map，Set，等等。复杂对象就是表示一个对象里面的数据结构非常复杂，比如，

```
{
 a: {
  b: {
   c: [],
   d: '123',
   e: {
    f: [1,2,3]
   }
  }
 }
}
```

正如之前所讲如果只是更改当前复杂对象里面的一个属性，那么这个对象还是原来那个对象，即内存地址没有更改。我们测试一下是否真是如此，

更新对象a里面的属性，不能改变a在内存里面的地址所以返回了true。

那么为了解决这个问题，我们需要首先对这个对象进行一次复制（注意，不是引用），然后把更新的操作应用在新创建的对象上面，然后保存。然后把这个新创建的对象传给React，那么就能顺利解决这个问题。

那我们以上面的复杂对象为例子，如果我们想对**f**属性进行更新，比如让其原来的数组[1,2,3] 更新成[1,2,3,4,5,6]。那么就会需要完成这几个步骤，

1. 对其进行一个复制，
2. 更新
3. 保存

那么在开始编码之前应该知道，只更新想更新的，而不想更新的就直接复制过来。

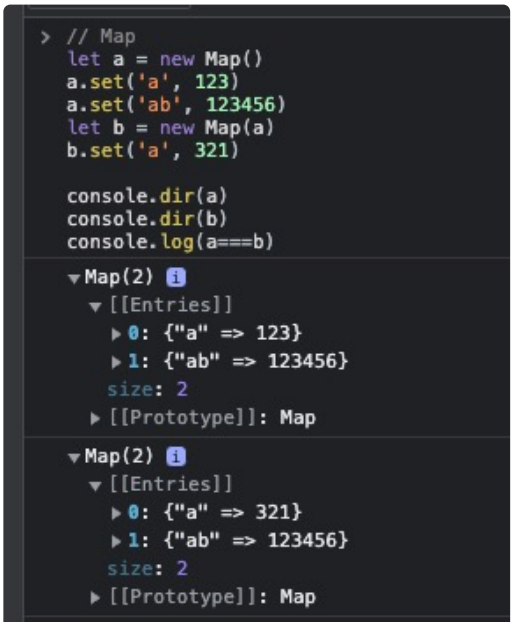
所以我们可以有下面的代码，

```
const origin = {
 a: {
  b: {
   c: [],
   d: '123',
   e: {
    f: [1,2,3]
   }
  }
 }
}

; const update = Object.assign({}, origin, {
 a: Object.assign({}, origin.a, {
  b: Object.assign({}, origin.a.b, {
   e: Object.assign({}, origin.a.b.e, {
    f: [1,2,3,4,5,6]
   })
  })
 })
})

console.dir(origin, 'origin')
console.dir(update, 'update')
console.log(origin === update, 'same obj?')
```

那么测试结果现实我们已经成功完成了更新，而且对象的内存地址已经更改。这里的Object.assign 可以使用 {...} 代替，但是效果一样。

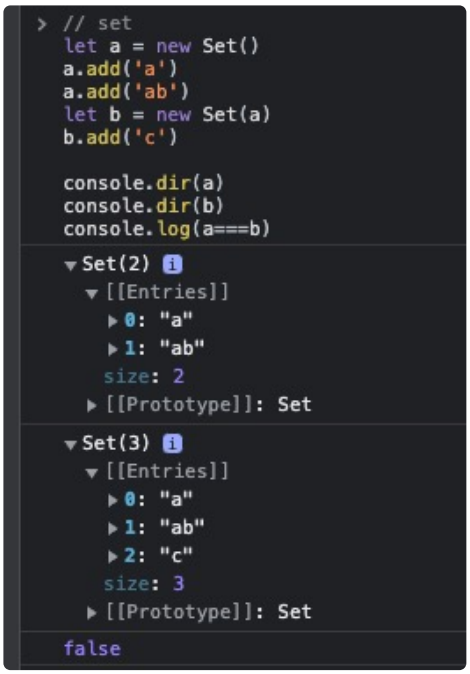
那么解决了【obj Object】 的复杂对象处理，我们对于其他类型的对象比如 Map, Set, Array, 也一起来完成一下吧。

```
// Map
let a = new Map()
a.set('a', 123)
a.set('ab', 123456)
let b = new Map(a)
b.set('a', 321)

console.dir(a)
console.dir(b)
console.log(a===b)
```

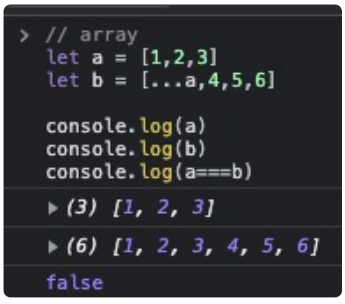


```
false
```

```
// set
let a = new Set()
a.add('a')
a.add('ab')
let b = new Set(a)
b.add('c')

console.dir(a)
console.dir(b)
console.log(a===b)
```




```
// array
let a = [1,2,3]
let b = [...a,4,5,6]

console.log(a)
console.log(b)
console.log(a===b)
```




```
// 如果问数组对象处理，其实和数组一样
```

```
let complex = [
{
    a: 'hello',
    b: {
        c: 123
    }
},
{
    a: 'hello2',
    b: {
        c: 123456
    },
    ab: {
        abc: 'hello world'
    }
}
]

let res = [...complex]
res[0].b.c = 321


console.log(complex)
console.log(res)
console.log(complex===res)
```

好了，基本常见的对象类型的正确更新方法我们已经一一罗列出来，希望能对你有所帮助。

React推荐使用Immutable helper 来解决这个问题，它如何优化上面的操作呢？其实翻看我们对于对象的处理方法会知道，代码语法结构不利于阅读和维护。Immutable helper的出现就是为了有一个看起来更容易阅读和维护的语法，也就是语法糖。比如，

```
import update from 'immutability-helper';

const newData = update(myData, {
  x: {y: {z: {$set: 7}}},
  a: {b: {$push: [9]}}
});
```

那么我们就先讲到这里，后续我们可能会接着这一篇，继续对Immutability-helper的源码来讲解