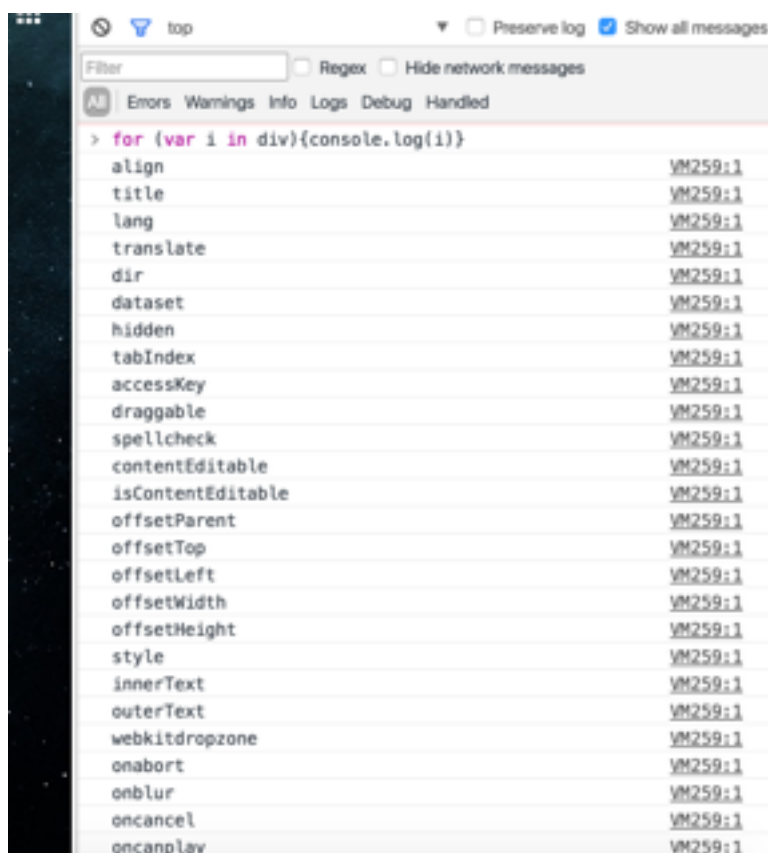


Little Virtual Dom

一个虚拟dom的简单实现

为什么要virtualDom

操作真实dom节点开销太大



spellcheck	VM259:1
contentEditable	VM259:1
isContentEditable	VM259:1
offsetParent	VM259:1
offsetTop	VM259:1
offsetLeft	VM259:1
offsetWidth	VM259:1
offsetHeight	VM259:1
style	VM259:1
innerText	VM259:1
outerText	VM259:1
webkitdropzone	VM259:1
onabort	VM259:1
onblur	VM259:1
oncancel	VM259:1
oncanplay	VM259:1
oncanplaythrough	VM259:1
onchange	VM259:1
onclick	VM259:1
onclose	VM259:1
oncontextmenu	VM259:1
oncuechange	VM259:1
ondblclick	VM259:1
ondrag	VM259:1
ondragend	VM259:1
ondragenter	VM259:1
ondragleave	VM259:1

什么是virtualDom

- element
- diff
- patch

Element实现

```
44  
45 var tree = Element('div',{id:'box'},[  
46     Element('h1',{key:'h1',class:'title',style:'  
        color:blue;font-size:24px'},['simple virtual  
        dom']),  
47     Element('ul',{key:'list1',id:'list'},[  
48         Element('li',{key:'li1',class:'li'},['  
            lalalallaa']),  
49         Element('li',{key:'li2',class:'li'},['  
            hahahahaha']),  
50         Element('li',{key:'li3',class:'li'},['  
            xixixixixixix'])  
51     ])  
52 ])  
53  
54 var theRoot = tree.render()  
55
```

Element实现

```
if(!(this instanceof Element)){
  if(!_.isArray(children) && children !== null){
    //children再处理, 去除掉children非true的元素
    children = _.slice(arguments,2).filter(_.truth)
  }
  return new Element(tagName,props,children)
}
```

Element实现

Render

```
Element.prototype.render = function(){  
  var t = this;  
  var el = document.createElement(t.tagName)  
  var props = t.props  
  
  for(var propName in props){  
    var propValue = props[propName]  
    _.setAttr(el, propName, propValue)  
  }  
  
  _.each(t.children, function(child){  
    var childNode = (child instanceof Element)  
      ? child.render()  
      : document.createTextNode(child);  
    childNode && el.appendChild(childNode)  
  })  
  
  document.body.appendChild(el)  
  return el  
}
```

Diff实现

- 两个相同组件产生类似的DOM结构，不同的组件产生不同的DOM结构；

——意味着相同dom往下走，不同dom直接删掉

- 对于同一层次的一组子节点，它们可以通过唯一的id进行区分。

```
⚠ Warning: Each child in an array or iterator should have a unique "key" prop. Check the render method of Wrapper. See http://fb.me/react-warning-keys for more information. runner-3.34.2.min.js:1
```

——这是React在遇到列表时却又找不到key时提示的警告。虽然无视这条警告大部分界面也会正确工作，但这通常意味着潜在的性能问题。因为React觉得自己可能无法高效的去更新这个列表。

Diff实现

所以diff核心步骤：

递归逐层比较，对比当前new / old node

- 当new node不存在，忽略，因为会在list_diff中被消除
- 当都是文本节点：push({type: patch.TEXT, content: newNode})
- 当tagName相同，且key相同（undefined也是相同），1,比较props，2, diffChildren()
- 其它：push({type: patch.REPLACE, node: newNode})

```
var REPLACE = 0;
var REORDER = 1;
var PROPS = 2;
var TEXT = 3;

patch.REPLACE = REPLACE;
patch.REORDER = REORDER;
patch.PROPS = PROPS;
patch.TEXT = TEXT;
```


Diff实现

- 当tagName相同，且key相同（undefined也是相同），1,比较props, 2, diffChildren()

1.

```
//对props  
if(propsPatches){  
  currentPatch.push({type:patch.PROPS,props:propsPatches})  
}
```

2. diffChildren()内调用list_diff

```
function diffChildren(oldChildren,newChildren,index,patches,currentPatch)  
  var diffs = listDiff(oldChildren,newChildren,'key')  
  newChildren = diffs.children;  
  if(diffs.moves.length){  
    var reorderPatch = {type:patch.REORDER,moves:diffs.moves}  
    currentPatch.push(reorderPatch)  
  }
```

小插曲list_diff

```
var diff = require("list-diff2")
var oldList = [{id: "a"}, {id: "b"}, {id: "c"}, {id: "d"}, {id: "e"}]
var newList = [{id: "c"}, {id: "a"}, {id: "b"}, {id: "e"}, {id: "f"}]

var moves = diff(oldList, newList, "id")
// `moves` is a sequence of actions (remove or insert):
// type 0 is removing, type 1 is inserting
// moves: [
//   {index: 3, type: 0},
//   {index: 0, type: 1, item: {id: "c"}},
//   {index: 0, type: 0},
//   {index: 4, type: 1, item: {id: "f"}}
// ]
```

Patch实现