# Repaint和 Reflow

reflow并不是就会执行UI渲染，UI渲染需要等待js执行完毕才会执行，可以理解为浏览器对js的执行和UI渲染都是同一个线程（虽然表现是这样，但是底层应该是js一个线程，UI渲染一个线程，只是浏览器只能执行一个线程）

reflow是在执行js的过程中执行的，它对性能有很大的影响

而UI渲染是js执行之后才执行的，它对性能的消耗更加巨大

因此，UI更新的性能目标有两个：

1减少reflow

2减少UI渲染次数

React与AngularJs比较类似，都是在某些关键点（程序自己决定什么时候开始执行更新算法）

AngularJs通过dirty check算法找到差异，并更新UI

React则是通过virtual DOM的对比找到差异，然后更新UI

4.2 为什么 virtual DOM 快？

下面我们讨论一下为什么virtual DOM会比其他框架的UI更新（dirty check & setter）策略要快

首先，使用defineProperty自动检测变化或者setter类型的就不参与讨论了，每次改属性都会进入绑定流程，想想都可怕

剩下AngularJs和react，他们的更新逻辑的入口都是在关键点调用更新接口

它们的共同点都是一次更新逻辑只会造成一次UI更新

AngularJs通过类似死循环的$digest循环扩展浏览器的原生事件循环，所有更新逻辑都是在js中执行完

react通过virtual DOM的diff得出改动，然后再统一的更新UI，这个过程也是一个js过程结束

两者都有同样的特征：**通过大量的js计算完成所有的DOM操作，结束之后才返回浏览器的UI渲染线程**

下面根据两者不同点来分析：

AngularJs 的DOM操作是分布式的，DOM操作封装在watcher里面，每当有属性变更，就会触发watcher，然后执行DOM操作

而react的DOM操作是集中式的，在diff之后，根据最终的patches执行DOM操作

集中式的DOM操作可以最大限度的利用浏览器的优化机制，详见【4.1.1 浏览器渲染机制的优化】

AngularJs 组件自带store，组件之间的互相影响可能会引起震荡

具体的是当组件A的属性变化之后，对应watcher里面的操作导致了B组件的属性变化，这时就需要触发相对应的watcher，这个过程有可能无穷无尽

另外AngularJs的dirty check是基于循环的，所以有可能watcher改变的是已经经过dirty check的store，因此dirty check要一直循环，直到所有的store都保持稳定，不再有任何新的变化，才能结束，当这个过程很长的时候，页面就会假死，因为浏览器不能执行UI更新，UI事件不能被处理，因为这个过程本身就在一个UI事件的处理期间，其他新的UI事件还在队列里面等着

这个问题的根本原因是AngularJs不能很好的控制组件之间的store

react没有这个问题就是因为react不是vm库，它没有store，看到这个估计大家都会傻眼，确实，AngularJs和react根本就不是一个可对比的库，本质都不一样

react应用，不管是配合flux还是redux，他们都是先把store计算稳定之后，再交给react去更新UI，这整个过程并不会劫持浏览器的原生事件循环，因此不会有页面的假死现象出现

另外，store计算完全是js计算，不会执行DOM的写操作，需要的只有甚至没有DOM的读操作，对于已经稳定的dom tree来说（浏览器的渲染队列里面已经没有缓存的DOM操作），批量的读操作是不会导致浏览器的repain和reflow的，因此store的计算过程会很快

因此，结论：store的稳定计算很快，react本身渲染也很快，所以使用virtual DOM的react很快

然后大家得出：virtual DOM很快

本质上，需要做的工作都是一样的，只是react把store的计算分离出去而已，但这也正体现了react的内聚性

另外还有一点也需要提及：

AngularJs，vue，avalon等vm库，都是用watcher模式，watcher是长存的

react是实时计算的，在diff之后，old tree就会被销毁，然后保留new tree作为下一次diff的old tree

因此在内存占用方面，也是react有优势

参考文献：

<http://imweb.io/topic/573e766f1156025b1dce9404>