****

预期的效果：

监听input的输入，input在输入的时候，会触发 watch与computed函数，并且会更新原始的input的数值。所以，我们能够预测的一共就是3个地方的改变。（其实上会有连带性的触发，例如：触发watch中的input函数，该函数又会触发与answer数据有关的处理）

看看内部是如何处理的：

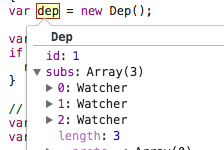
Vue在初始化data的时候，会通过Object.defineProperty重新定义input的set与get访问接口，同时会创建一个记录并且保持其数据对应的依赖watcher对象的Dep对象。

简单来说

1．watch中的input函数会生成一个new watcher对象，因为它与input有关系，所以需要在input的Dep中保存一份引用

2．computed中的compiledMarkdown函数会生成一个new watcher对象，，因为它与input有关系，所以需要在input的Dep中保存一份引用

所以input的监控内部创建的Dep的结构就是如下



watch与computed明明只有2个对应的watcher对象，为什么subs会有3个呢？多增加的一个是干什么的？这里先跳过

所以最终可以简单的梳理下：当input数据发生变化的时候，只需要调用响应依赖的watcher对象，watcher对象就会负责自己的更新处理。这里面向对象的设计优势就体现出来了，将行为分布在各个对象中，并让这些对象负责自己的行为，所以每个不同watcher对象更新各自的特点，处理各自的逻辑

更新逻辑：

队列更新

vue2在vue1的设计上，继续保留了队列的处理方式，vue2最终的dom更新上没有用队列+直接更新的方式了，这种简单粗暴。而是采用时下最流行的 virtual dom

更新原理：

记得在Vue1中，每个Watcher对象都会保存各自的dom节点的处理方式，通过对Watcher的的处理达到直接更新dom的目的。Vue2因为引入的Virtual Dom的机制，所以Watcher的工作就需要变化了，大多数的Watcher不再直接负责DOM的更新操作，而只是更新数据。这里用了大多数，因为还有一个Watcher是跟Virtual Dom相关的。所以这就是在上文提到的Dep中会多一个Watcher的原因了

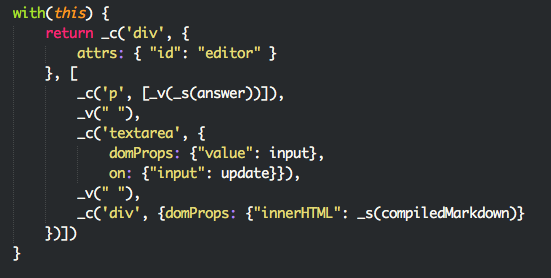
Virtual DOM

虚拟DOM的文章现在已经很多了，但是如何紧密结合vue中，到实际的运用是我们分析的重点，这里只是粗略下，我还要抽时间把算法看完先

原理：

简单的说，直接通过JS操作浏览器API去更新DOM节点是很慢的，大量的页面处理中，开发者不经意就会调用更多多余或者重复的操作，这种是有性能开销的。那么有什么办法减少这种是误操作呢？就是通过一种方式能算出来最小的更新量，从而提高效率。既然要计算出对小的更新量，那么就会有对比，需要通过对新旧两个节点的对比从而计算出。DOM的操作很慢，但是JS的确很快的，既然我们可以用JS对象表示DOM结构，那么当数据状态发生变化而需要改变DOM结构时，我们先通过JS对象表示的虚拟DOM计算出实际DOM需要做的最小变动，然后再操作实际DOM，从而避免了粗放式的DOM操作带来的性能问题。

根据上面的原理，在实现上首先要把整个真实的DOM树编译成一个JS渲染函数，通过渲染函数与真实数据结合后，就会编译出一个叫vnode的东东，这个就是虚拟DOM了。上面的结构我截图看看渲染函数长什么样子



很多时候，靠手工直接优化dom绝对是比virtual dom的效率更高，简单的dom结构用手工优化没有问题，但当页面结构很庞大，结构很复杂时，手工优化会花去大量时间，而且可维护性也不高，不能保证每个人都有手工优化的能力。virtual dom很多时候都不是最优的操作，但它具有普适性，在效率、可维护性之间达平衡。

代码逻辑：

继续上文，逻辑上请自行跳跃。上文提到input这个数据内部实现Dep对象，这个Dep就是对Watcher对象的引用的保存。那么接下来就是需要遍历所有依赖的Watcher对象进行相应的处理。