计算属性：

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。在模板中放入太多的逻辑会让模板过重且难以维护。例如：

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

在这个地方，模板不再是简单的声明式逻辑。你必须看一段时间才能意识到，这里是想要显示变量 message 的翻转字符串。当你想要在模板中多次引用此处的翻转字符串时，就会更加难以处理。

所以，对于任何复杂逻辑，都应当使用计算属性。

例子：

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // 计算属性的 getter
    reversedMessage: function () {
      // `this` 指向 vm 实例
      return this.message.split('').reverse().join('')
    }
  }
})
```

Vue computed的实现原理

1. computed是计算一个新的属性，并将该属性挂载到vm（Vue实例）上，而watch是监听已经存在且已挂载到vm上的数据，所以用watch同样可以监听computed计算属性的变化（其它还有data、props）
2. computed本质是一个惰性求值的观察者，具有缓存性，只有当依赖变化后，第一次访问 computed 属性，才会计算新的值，而watch则是当数

据发生变化便会调用执行函数

3. 从使用场景上说，computed适用一个数据被多个数据影响，而watch适用一个数据影响多个数据；

当组件初始化的时候，computed和data会分别建立各自的响应系统，Observer遍历data中每个属性设置get/set数据拦截

1. 初始化computed会调用initComputed函数

    a. 注册一个watcher实例，并在内实例化一个Dep消息订阅器用作后续收集依赖（比如渲染函数的watcher或者其他观察该计算属性变化的watcher）

    b. 调用计算属性时会触发其Object.defineProperty的get访问器函数

    c. 调用watcher.depend()方法向自身的消息订阅器dep的subs中添加其他属性的watcher

    d. 调用watcher的evaluate方法（进而调用watcher的get方法）让自身成为其他watcher的消息订阅器的订阅者，首先将watcher赋给Dep.target，然后执行getter求值函数，当访问求值函数里面的属性（比如来自data、props或其他computed）时，会同样触发它们的get访问器函数从而将该计算属性的watcher添加到求值函数中属性的watcher的消息订阅器dep中，当这些操作完成，最后关闭Dep.target赋为null并返回求值函数结果。

2. 当某个属性发生变化，触发set拦截函数，然后调用自身消息订阅器dep的notify方法，遍历当前dep中保存着所有订阅者wathcer的subs数组，并逐个调用watcher的 update方法，完成响应更新。