你真的明白 compouted 与 watch 的区别？——基本使用方法以及试用场景的区别

在 Vue 中，computed 与 watch 声明的方式很相似，总有些同学傻傻啥啥分不清楚，接下来给给大家做一个详细的总结，并且在之后的文章中更深入的探究一下 computed 与 watch 的底层实现。

computed 与 watch 的基本使用

computed 的基本使用

computed 计算属性，尽管以函数的形式声明，但是调用的时候使用的方式作为属性进行调用的。
computed 计算属性，真正的意义在于派生，就是根据已有属性新产生一个新的属性，这个属性就是计算属性，已有属性不局限于 data 中的数据，props 或者其他的计算属性也可以。计算属性允许我们对指定的视图，复杂的值计算。这些值将绑定到依赖项值，只在需要时更新。

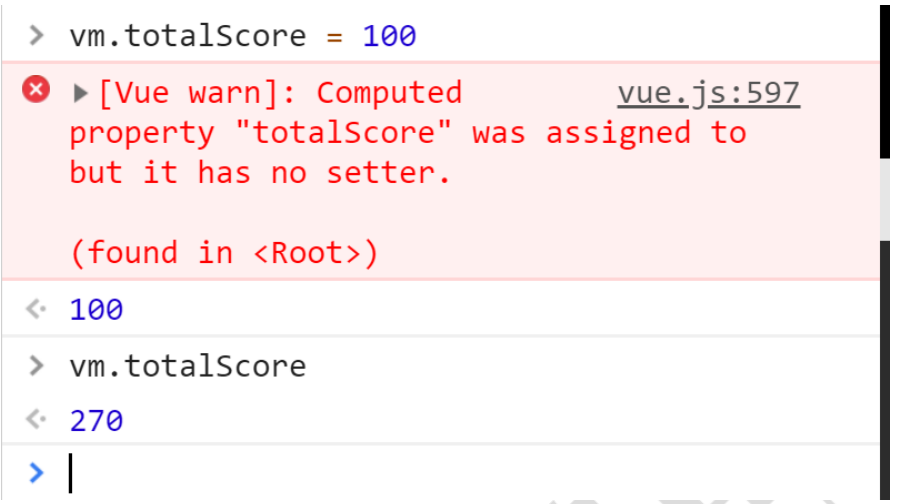
```
<div id="app">
    <!-- 调用的时候作为正常的属性进行调用，而非方法 -->
    <p>{{ totalScore }}</p>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            subjects: [
                {
                    subject: 'Englist',
                    score: 100
                },
                {
                    subject: 'Math',
                    score: 80
```

```
                    },
                    {
                        subject: 'History',
                        score: 90
                    }
                ]
            },
            computed: {
                // 声明的方式是函数式声明
                totalScore() {
                    return this.subjects.reduce((prev, next) => {
                        return prev + next.score
                    }, 0);
                }
            }
        })
    </script>
```

对于计算属性，Vue 会记住计算的属性所依赖的值（在我们这个示例中，那就是 subjects）。通过这样做，Vue 只有在依赖变化时才可以计算值。否则，将返回以前缓存的值。这也意味着只要 subjects 还没有发生改变，多次访问 totalScore 计算属性会立即返回之前的计算结果，而不必再次执行函数。

上面示例也说明，**在 Vue 中计算属性是基于它们的依赖进行缓存的，而方法是不会基于它们的依赖进行缓存的。从而使用计算属性要比方法性能更好**。

计算属性是属性，那他能不能修改呢？

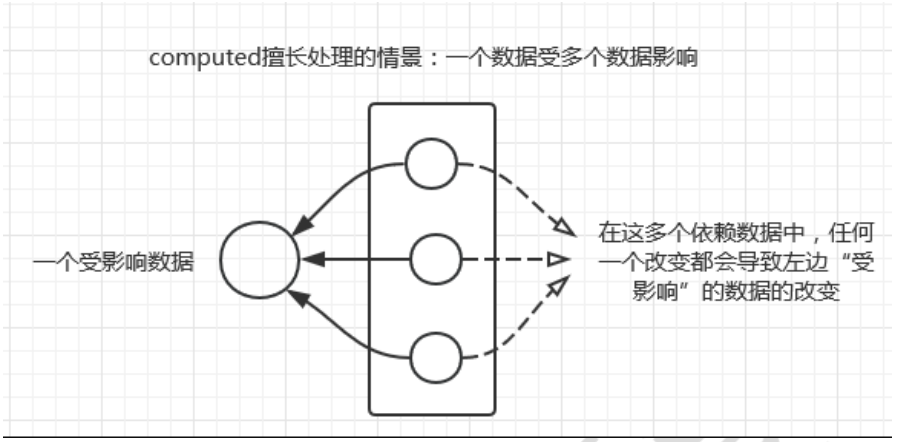


很明显，通过报错信息来说是不能修改，他是一个**“只读”**属性，no setter!
计算属性默认只有 getter，不过在需要时你也可以提供一个 setter。
这里要说明一下，这里设置的 setter 函数，并不是为了给这个计算属性进行赋值，而是当对该计算属性进行 **“赋值”** 时调用的函数。并不是真正的改变这个计算属性。
下面借助官网的例子说明：

```
computed: {
  fullName: {
    // getter
    get: function () {
      return this.firstName + ' ' + this.lastName
    },
    // setter
    set: function (newValue) {
      var names = newValue.split(' ')
      this.firstName = names[0]
      this.lastName = names[names.length - 1]
    }
  }
}
```

在这里我们调用 vm.fullName = ‘John Doe’ 时，会调用 set 函数，这个函数里面修改的是 firstName 以及 lastName，当 fullName 这个计算属性相关的属性发生变化，这个属性会重新计算，所以说计算属性不能修改，就算设置 setter 函数也是改变已有属性的值，使得计算属性重新计算。

计算属性的适用情况：



监听器的基本使用

虽然计算属性在大多数情况下更合适，但有时也需要一个自定义的侦听器。这就是为什么 Vue 通过 watch 选项提供了一个更通用的方法，来响应数据的变化。当需要在数据变化时执行异步或开销较大的操作时，这个方式是最有用的。

Vue 确实提供了一种更通用的方式来观察和响应 Vue 实例上的数据变动：watch 属性。当你有一些数据需要随着其它数据变动而变动时，你很容易滥用 watch。然而，通常更好的想法是使用计算属性而不是命令式的 watch 回调。

假设有如下代码：

```
<div>
    <p>FullName: {{fullName}}</p>
    <p>FirstName: <input type="text" v-model="firstName"></p>
</div>

new Vue({
  el: '#root',
  data: {
    firstName: 'Dawei',
    lastName: 'Lou',
    fullName: ''
```

```
  },
  watch: {
    firstName(newName, oldName) {
      this.fullName = newName + ' ' + this.lastName;
    }
  }
})
```

上面的代码的效果是，当我们输入 firstName 后，wacth 监听每次修改变化的新值，然后计算输出 fullName。

handler 方法和 immediate 属性

这里 watch 的一个特点是，最初绑定的时候是不会执行的，要等到 firstName 改变时才执行监听计算。那我们想要一开始就让他最初绑定的时候就执行改怎么办呢？我们需要修改一下我们的 watch 写法，修改过后的 watch 代码如下：

```
watch: {
  firstName: {
    handler(newName, oldName) {
      this.fullName = newName + ' ' + this.lastName;
    },
    // 代表在 wacth 里声明了 firstName 这个方法之后立即先去执行 handler 方法
    immediate: true
  }
}
```

注意到 handler 了吗，我们给 firstName 绑定了一个 handler 方法，之前我们写的 watch 方法其实默认写的就是这个 handler，Vue.js 会去处理这个逻辑，最终编译出来其实就是这个 handler。

而 immediate:true 代表如果在 wacth 里声明了 firstName 之后，就会立即先去执行里面的 handler 方法，如果为 false 就跟我们以前的效果一样，不会在绑定的时候就执行。

deep 属性

watch 里面还有一个属性 deep，默认值是 false，代表是否深度监听，比

如我们 data 里有一个 obj 属性：

```
<div>
    <p>obj.a: {{obj.a}}</p>
    <p>obj.a: <input type="text" v-model="obj.a"></p>
</div>

new Vue({
 el: '#root',
 data: {
  obj: {
   a: 123
  }
 },
 watch: {
  obj: {
   handler(newName, oldName) {
      console.log('obj.a changed');
   },
   immediate: true
  }
 }
})
```

当我们在在输入框中输入数据视图改变 obj.a 的值时，我们发现是无效的。受现代 JavaScript 的限制 (以及废弃 Object.observe)，Vue 不能检测到对象属性的添加或删除。由于 Vue 会在初始化实例时对属性执行 getter/setter 转化过程，所以属性必须在 data 对象上存在才能让 Vue 转换它，这样才能让它是响应的。

默认情况下 handler 只监听 obj 这个属性它的引用的变化，我们只有给 obj 赋值的时候它才会监听到，比如我们在 mounted 事件钩子函数中对 obj 进行重新赋值：

```
mounted: {
 this.obj = {
  a: '456'
 }
```

```
}
```

这样我们的 handler 才会执行，打印 obj.a changed。

相反，如果我们需要监听 obj 里的属性 a 的值呢？这时候 deep 属性就派上用场了！

```
watch: {
  obj: {
    handler(newName, oldName) {
      console.log('obj.a changed');
    },
    immediate: true,
    deep: true
  }
}
```

deep 的意思就是深入观察，监听器会一层层的往下遍历，给对象的所有属性都加上这个监听器，但是这样性能开销就会非常大了，任何修改 obj 里面任何一个属性都会触发这个监听器里的 handler。

优化，我们可以是使用字符串形式监听。

```
watch: {
  'obj.a': {
    handler(newName, oldName) {
      console.log('obj.a changed');
    },
    immediate: true,
    // deep: true
  }
}
```

这里要注意：采用字符串监听的方式进行监听，对于数组不可以写成

```
watch: {
      ‘subjects[0].score’: {
            …
      }
}
```

这么写 Vue 回报错：

```
▶[Vue warn]: Failed watching vue.js:597
path: "subjects[0].score" Watcher only
accepts simple dot-delimited paths. For
full control, use a function instead.

(found in <Root>)
```

监听器只接受简单的点分隔路径。要完全控制，请改用函数。
那那怎么办使用

```
watch: {
    'subjects.0.score': {
        ...
    }
}
```

这种方式违背我们对于数组的操作，但确实是可以的。

这样 Vue.js 才会一层一层解析下去，直到遇到属性 a，然后才给 a 设置监听函数。
监听器适用场景：