Vue预习课

核心知识04——计算属性和监听器

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，此时就可以考虑计算属性和监听器。

范例：课程数量统计

```
<p>
    <!-- 绑定表达式 -->
    <!-- 课程总数：{{courses.length + '门'}} -->
    <!-- 计算属性 -->
    <!-- 课程总数：{{total}} -->
    <!-- 监听器 -->
    课程总数：{{totalCount}}
</p>

<script>
    const app = new Vue({
        computed: {
            total() {
                return this.courses.length + '门'
            }
        },
        // 下面这种不能生效，因为初始化时不会触发
        // watch: {
        //     courses(newValue, oldValue) {
        //         this.totalCount =  newValue.length + '门'
        //     }
        // },
        watch: {
            courses: {
                immediate: true,
                // deep: true,
                handler(newValue, oldValue) {
                    this.totalCount =  newValue.length + '门'
                }
            }
        },
    })
</script>
```

> 计算属性和监听器

> 计算属性 vs 监听器

> - 处理数据的场景不同，监听器适合**一个数据影响多个数据**，计算属性适合**一个数据受多个数据影响**
> - **计算属性有缓存性**，计算所得的值如果没有变化不会重复执行
> - 监听器选项提供了更通用的方法，适合**执行异步操作或较大开销操作**的情况

神奇的模板语法是如何实现的

在底层的实现上，Vue 将模板编译成虚拟 DOM 渲染函数。结合响应系统，Vue 能够智能地计算出最少需要重新渲染多少组件，并把 DOM 操作次数减到最少。

```
// 输出vue替我们生成的渲染函数一窥究竟
console.log(app.$options.render)
```

```
// 它长这个样子
(function anonymous(
) {
with(this){return _c('div',{attrs:{"id":"app"}},[_c('h2',{attrs:
{"title":title}},[_v("\n            "+_s(title)+"\n        ")]),_v("
"),_c('input',{directives:[{name:"model",rawName:"v-model",value:
(course),expression:"course"}],attrs:{"type":"text"},domProps:{"value":
(course)},on:{"keydown":function($event)
{if(!$event.type.indexOf('key')&&_k($event.keyCode,"enter",13,$event.key,"Enter"
))return null;return addCourse($event)},"input":function($event)
{if($event.target.composing)return;course=$event.target.value}}}),_v("
"),_c('button',{on:{"click":addCourse}},[_v("新增课程")]),_v(" "),(courses.length
== 0)?_c('p',[_v("没有任何课程信息")]):_e(),_v("
"),_c('ul',_l((courses),function(c){return _c('li',{class:{active:
(selectedCourse === c)},on:{"click":function($event){selectedCourse = c}}},
[_v(_s(c))])}),0)])}
})
```

改写为渲染函数版本试试，02-cart-render.html

```
const app = new Vue({
    // 引入上面的render函数
    render() {
        with (this) { return ... }
    }
})
```

结论：Vue通过它的**编译器**将模板编译成**渲染函数**，在数据发生变化的时候再次执行**渲染函数**，通过对比两次执行结果得出要做的dom操作，模板中的神奇魔法得以实现。

这些功能到底是怎么实现的，我们将在正课中给大家带来答案。

> 如果你熟悉虚拟 DOM 并且偏爱 JavaScript 的原始力量，也可以不用模板，直接写渲染 (render) 函数。