Vue预习课

核心知识03——模板语法

Vue模板语法

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。所有 Vue.js 的模板都是合法的 HTML，所以能被遵循规范的浏览器和 HTML 解析器解析

插值文本

数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值

范例：设置标题

```
<div id="app">
    <h2>
        <!-- 插值文本 -->
        {{title}}
    </h2>
</div>

<script src="vue.js"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            title: '开课吧购物车'
        }
    })
</script>
```

> 插值文本

特性

HTML特性不能用Mustache 语法，应该使用v-bind指令

范例：设置标题

```
<div id="app">
    <!-- 特性、属性值绑定使用v-bind指令 -->
    <h2 v-bind:title="title">...</h2>
</div>
```

> 插值文本

列表渲染

我们可以用 `v-for` 指令基于一个数组来渲染一个列表。`v-for` 指令需要使用 `item in items` 形式的特殊语法，其中 `items` 是源数据数组，而 `item` 则是被迭代的数组元素的**别名。**

范例：课程列表

```
<div id="app">
    <!-- 条件渲染 -->
    <p v-if="courses.length == 0">没有任何课程信息</p>
    <!-- 列表渲染 -->
    <ul>
        <li v-for="c in courses">{{c}}</li>
    </ul>
</div>

<script src="vue.js"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            courses: ['web全栈', 'web高级']
        }
    })
</script>
```

> 列表渲染

表单输入绑定

你可以用 `v-model` 指令在表单 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。**它会根据控件类型自动选取正确的方法来更新元素。** `v-model` 本质上是语法糖。它将转换为输入事件以更新数据，并对一些极端场景进行一些特殊处理。

范例：新增课程

```
<!-- 表单输入绑定 -->
<input v-model="course" type="text" v-on:keydown.enter="addCourse"/>
```

> 表单输入绑定

事件处理

可以用 `v-on` 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。

范例：新增课程

```
<!-- 事件处理 -->
<button v-on:click="addCourse">新增课程</button>


<script>
    const app = new Vue({
        methods: {
            addCourse() {
                this.courses.push(this.course);
            }
        },
    })
</script>
```

> 事件处理

class与style绑定

操作元素的 class 列表和内联样式是数据绑定的一个常见需求。因为它们都是属性，所以我们可以用 `v-bind` 处理它们：只需要通过表达式计算出字符串结果即可。不过，字符串拼接麻烦且易错。因此，在将 `v-bind` 用于 `class` 和 `style` 时，Vue.js 做了专门的增强。**表达式结果的类型除了字符串之外，还可以是对象或数组。**

范例：点选样式

```
<style>
    .active {
        background-color: #ddd;
    }
</style>

<ul>
    <!-- class绑定 -->
    <li v-for="c in courses"
        :class="{active: (selectedCourse === c)}"
        @click="selectedCourse = c">{{c}}</li>
    <!-- style绑定 -->
    <!-- <li v-for="c in courses"
            :style="{backgroundColor: (selectedCourse ===
c)?'#ddd':'transparent'}"
            @click="selectedCourse = c">{{c}}</li> -->
</ul>

<script>
    const app = new Vue({
        data: {
            // 保存选中项
            selectedCourse: '',
        },
```

```
    })
</script>
```

> class与style绑定

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
>
> - **监听器更通用**，理论上计算属性能实现的侦听器也能实现
> - 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。