第3章 Vue实例核心选项

3.1 数据选项

在实例化Vue对象时,需要为Vue的构造函数提供一系列配置信息,代码如下:

```
new Vue({
    //选项
})
```

当使用new操作符创建Vue实例时,可以为实例传入一个选项对象,选项对象中有很多类型的数据,具体内容如下。

3.1.1 data选项

data选项支持Object和Function类型的数据,是Vue实例的数据对象。在Vue中使用递归将data的property转换为getter/setter,从而让data的property能够响应数据变化。对象必须是纯粹的对象,代码如下:

```
var data = { a: 1 }

//直接创建一个实例
var vm = new Vue({
  data: data
})
vm.a // => 1
vm.$data === data // => true
```

当定义组件时,data必须声明为一个返回初始数据对象的函数。因为组件可能被用来创建多个实例,如果data仍然是一个纯粹的对象,则所有的实例将共享引用同一个数据对象。通过提供data()函数,每次创建一个新实例后,可以通过调用data()函数,返回初始数据的一个全新副本数据对象,代码如下:

```
//Vue.extend() 中 data 必须是函数
var Component = Vue.extend({
  data: function () {
```

```
    return{ a: 1 }
  }
})
```

如果在自定义组件中声明的 data 的 property 使用了箭头函数，那么 this 不会指向这个组件的实例，不过仍然可以将其实例作为函数的第一个参数访问，代码如下：

```
data:vm => ({ a: vm.myProp })
```

3.1.2 props 选项

props 选项的值可以是数组或对象，用于接收来自父组件的数据。props 可以是简单的数组，或者使用对象作为替代，对象允许配置高级选项，如类型检测、自定义验证和设置默认值。

props 选项的值如果为对象类型，则对象的语法具体使用如下。

（1）type：值的类型可以是 String、Number、Boolean、Array、Object、Date、Function、Symbol、任何自定义构造函数或上述内容组成的数组，可以通过该属性检查一个 prop 是否是指定的类型，否则会抛出警告。

（2）default：值为 any 类型，为该 prop 指定一个默认值。

（3）required：值为 Boolean 类型，定义该 prop 是否为必填项。

（4）validator：值为函数，自定义验证函数会将该 prop 的值作为唯一的参数代入。

props 选项代码如下：

```
//简单语法
Vue.component('props-demo-simple', {
  props: ['size', 'myMessage']
})

//对象语法，提供验证
Vue.component('props-demo-advanced', {
  props: {
    //检测类型
    height: Number,
    //检测类型 + 其他验证
    age: {
      type: Number,
      default: 0,
      required: true,
      validator: function (value) {
        return value >= 0
      }
    }
  }
})
```

3.1.3 computed 选项

8min

computed 是计算属性，其值的类型是一个对象，对象中的属性值为函数。计算属性将被混入 Vue 实例中，所有 getter 和 setter 的 this 上下文自动绑定为 Vue 实例。如果计算属性的值为一个箭头函数，那么 this 不会指向这个组件的实例，不过仍然可以将其实例作为函数的第一个参数访问。

计算属性有以下特点。

（1）模板中放入太多的逻辑会让模板过重且难以维护，使用计算属性可以让模板更加简洁。

（2）计算属性是基于它们的响应式依赖进行缓存的。

（3）computed 比较适合对多个变量或者对象进行处理后返回一个结果值，也就是说多个变量中的某一个值发生了变化则我们监控的这个值也会发生变化。

index.html 文件代码如下：

```
<div id="app">
    <!--
        当多次调用 reverseString 的时候
        只要里面的 num 值不改变，它会把第一次计算的结果直接返回
        直到 data 中的 num 值改变，计算属性才会重新发生计算
    -->
    <div>{{reverseString}}</div>
    <div>{{reverseString}}</div>
      <!-- 调用 methods 中的方法的时候，它每次会重新调用 -->
    <div>{{reverseMessage()}}</div>
    <div>{{reverseMessage()}}</div>
  </div>
  <script type="text/JavaScript">
    /*
      计算属性与方法的区别：计算属性是基于依赖进行缓存的，而方法不缓存
    */
    var vm = new Vue({
      el: '#app',
      data: {
        msg: 'Nihao',
        num: 100
      },
      methods: {
        reverseMessage: function(){
          console.log('methods')
          return this.msg.split('').reverse().join('');
        }
      },
      //computed 属性的声明和 data 属性、methods 属性是平级关系
      computed: {
        //reverseString 这个是我们自己定义的名字
        reverseString: function(){
```

```
            console.log('computed')
            var total = 0;
            //当 data 中的 num 的值发生改变时,reverseString 函数会自动进行计算
            for(var i = 0;i <= this.num;i++){
              total += i;
            }
            //在 reverseString 函数中必须有 return,否则在页面中无法获取计算后的值
            return total;
          }
        }
      });
    </script>
```

3.1.4 methods 选项

methods 用来定义 Vue 实例中绑定的函数,其值的类型为对象,对象的属性值必须为函数类型,methods 将被混入 Vue 实例中。可以直接通过 vm 实例访问这些函数,也可以在指令中直接使用。函数中的 this 自动绑定为 Vue 实例,代码如下:

```
var vm = new Vue({
  data:{ a: 1 },
  methods: {
    plus: function () {
      this.a++
    }
  }
})
vm.plus()
vm.a //2
```

15min

3.1.5 watch 选项

watch 是监听器,其值为一个对象,对象中的属性值可以为函数、对象和数组。当 data 中的一个属性需要被观察其内部值的改变时,可以通过 watch 来监听 data 属性的变化。

watch 监听器的基本语法如下。

(1) 使用 watch 来响应数据的变化。

(2) 一般用于异步或者开销较大的操作。

(3) watch 中的属性一定是 data 中已经存在的数据。

(4) 当需要监听一个对象的改变时,普通的 watch 方法无法监听到对象内部属性的改变,只有 data 中的数据才能够被监听到变化,此时需要 deep 属性对对象进行深度监听。

index.html 文件代码如下:

```
<div id = "app">
  <div>
    <span>名:</span>
```

```
    <span>
      <input type="text" v-model='firstName'>
    </span>
  </div>
  <div>
    <span>姓:</span>
    <span>
      <input type="text" v-model='lastName'>
    </span>
  </div>
  <div>{{fullName}}</div>
</div>

<script type="text/JavaScript">
  /*侦听器*/
var vm = new Vue({
    el: '#app',
    data: {
      firstName: 'Jim',
      lastName: 'Green',
      //fullName: 'Jim Green'
    },
    //watch属性和data属性、methods属性是平级关系
    watch: {
      //注意:这里firstName对应data中的firstName
      //当firstName值改变的时候会自动触发watch
      firstName: function(val) {
        this.fullName = val + ' ' + this.lastName;
      },
      //注意:这里lastName对应data中的lastName
      lastName: function(val) {
        this.fullName = this.firstName + ' ' + val;
      }
    }
  });
</script>
```

3.2 DOM 渲染选项

Vue 通过双向数据绑定,将数据实时渲染为浏览器能够解析的 DOM,在 Vue 实例中有一些关于操作 DOM 的选项,具体内容如下。

3.2.1 el 选项

el 可以是字符串也可以是一个节点,当使用字符串时 el 是 CSS 选择器,当使用节点时 el 是一个 HTMLElement 实例,代码如下:

```
<div id = "app">
</div>
<script type = "text/JavaScript">
    new Vue({
        el: '#app'
    })
</script>
```

3.2.2 template 选项

template 选项是一个字符串模板，属性值为字符串类型，作为 Vue 实例的标识使用。模板会替换挂载的元素，挂载元素的内容都将被忽略，除非模板的内容有分发插槽。

3.2.3 render 选项

render 是字符串模板的替代方案，可以使用 JavaScript 编程的方式实现。该渲染函数接收一个 createElement()方法作为第一个参数用来创建 VNode。如果组件是一个函数组件，则渲染函数还会接收一个额外的 context 参数，为没有实例的函数组件提供上下文信息。

25min

3.3 生命周期钩子

每个 Vue 实例都有完整的生命周期，即从开始创建、初始化数据、编译模板、DOM 挂载、数据更新并重新渲染、组件卸载等一系列过程，我们称为 Vue 实例的生命周期。在整个周期中，每个节点都会有一个钩子函数来处理该阶段的某些事物，这些钩子函数被称为生命周期函数。生命周期每个阶段具体的钩子函数内容如下。

3.3.1 create 初始化

1. beforeCreate

在 Vue 实例初始化之后，在数据观测和事件配置之前被调用，这时自定义组件的选项对象还没有被创建出来，el 和 data 选项并未初始化，所以在该钩子函数中无法访问 methods、data、computed 等选项上的方法和数据。

2. created

在 Vue 实例已经创建完成之后被调用，在该钩子函数中可以对 data 数据进行观测，对实例上的属性和方法进行运算，执行 watch 和 event 事件的回调，完成 data 数据的初始化等操作。但是在该阶段，由于还未到挂载阶段，所以 $el 属性仍然不能访问。

在 created 钩子函数中，我们常对一些实例进行预处理操作，例如发送 ajax 请求等。因为在该阶段可以调用 methods 中的方法，并对 data 中的数据进行修改。由于该阶段还未渲染 DOM，所以在该阶段中不能进行有关 DOM 操作的处理。

3.3.2 mount 组件挂载

1. beforeMount

在挂载之前被调用，Vue 中的 render()函数第一次被调用，在该阶段虚拟 DOM 已经完成了模板编译，把 data 中的数据和模板生成了 HTML，完成了 el 和 data 初始化工作。该阶段虽然已经完成了基本的初始化工作，但是还没有执行挂载操作。

2. mounted

在挂载完成后调用，也就是模板中的 HTML 已经被渲染到了浏览器中，一般会在该阶段执行 ajax 操作，或者执行 DOM 元素操作。

3.3.3 update 组件更新

1. beforeUpdate

在数据更新之前被调用，发生在虚拟 DOM 被重新渲染和打补丁之前，可以在该钩子中进一步更改状态，不会触发附加重复渲染过程。

2. updated

当 data 的数据发生改变时，虚拟 DOM 被重新渲染后会调用 updated()钩子函数。在调用时，组件 DOM 已经发生了更新，在这个阶段可以执行依赖 DOM 的操作。在该阶段应该避免出现操作数据的情况，因为可能会导致虚拟 DOM 被重新渲染，从而使更新进入无限循环的状态。该钩子函数在服务器端渲染期间不会被调用。

3.3.4 destroy 组件销毁

1. beforeDestroy

在 Vue 实例销毁之前调用，此时的实例还是完全可用状态。在该阶段可以使用 this 获取 Vue 实例，一般情况下会在该钩子函数中进行一些重置操作。例如，清除组件中的定时器，或清除监听 DOM 的事件等。

2. destroyed

在 Vue 实例被销毁之后调用，当钩子函数被调用后，所有的事件监听都会被移除，并且所有的组件都会被销毁。该钩子函数在服务器端渲染时不会被调用。

3.4 资源选项

3.4.1 directives 选项

7min

在 Vue 中除了内置的指令，例如 v-model 和 v-bind 等，Vue 还允许手动注册自定义指令。在 Vue 2 中，代码复用和抽象的主要形式是组件，然而在有些情况下，仍然需要对普通 DOM 元素进行底层操作，这时就需要用 directives 选项来自定义指令。

一个指令定义对象可以提供以下几个钩子函数。

(1) bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

(2) inserted：被绑定元素插入父节点时调用（仅保证父节点存在，但不一定已被插入文档中）。

(3) update：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。

(4) componentUpdated：指令所在组件的 VNode 及其子 VNode 全部更新后调用。

(5) unbind：只调用一次，指令与元素解绑时调用。

如果想要在页面渲染完成后自动让输入框获取焦点，可以使用 directives 选项创建一个自定义指令，代码如下：

```
directives: {
  focus: {
    //指令的定义
    inserted: function (el) {
      el.focus()
    }
  }
}
```

在组件中使用自定义指令，代码如下：

```
< input v - focus >
```

11min

3.4.2 filters 选项

Vue.js 允许自定义过滤器，可以被用于一些常见的文本格式化。过滤器可以用在两个地方：双花括号插值和 v-bind 表达式。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符号指示，代码如下：

```
var vm = new Vue({
  el: '#app',
  data: {
    msg: ''
  },
  //filters 属性的声明和 data 属性、methods 属性是平级关系
  //定义 filters 中的过滤器为局部过滤器
  filters: {
    //upper 自定义的过滤器名字
    //upper 被定义为接收单个参数的过滤器函数，表达式 msg 的值将作为参数传入函数中
    upper: function(val) {
      //过滤器中一定要有返回值，这样外界使用过滤器的时候才能得到结果
      return val.charAt(0).toUpperCase() + val.slice(1);
    }
  }
});
```

过滤器可以串联使用，在此处是将 filter1 的值作为 filter2 的参数使用，代码如下：

```
{{data|filter1|filter2  }}
```

还可以为过滤器传递参数，例如，filterA 被定义为接收 3 个参数的过滤器函数，其中 message 的值作为第 1 个参数，普通字符串 'arg1' 作为第 2 个参数，表达式 'arg2' 的值作为第 3 个参数，代码如下：

```
<div id = "box">
  {{ message | filterA('arg1', 'arg2') }}
</div>
<script>
  //在过滤器中第 1 个参数对应的是管道符前面的数据 n,此时对应 message
  //第 2 个参数 a 对应实参 arg1 字符串
  //第 3 个参数 b 对应实参 arg2 字符串
  Vue.filter('filterA',function(n,a,b){
  if(n < 10){
    return n + a;
  }else{
    return n + b;
  }
});
new Vue({
  el:"#box",
  data:{
    message: "哈哈哈"
  }
})
</script>
```