

vue2.0第二节课

一、计算属性:computed

在Vue中，`computed`（计算属性）是一种特殊的属性，用于对数据进行动态计算和处理，并返回计算结果。

计算属性会根据它依赖的响应式数据自动进行缓存，只有在依赖发生变化时才会重新计算，这样可以提高性能并减少不必要的计算。

作用：

- 响应式计算数据，计算属性可以像普通属性一样在模板中使用。

语法举例：

```
<div id="div1">
  <!-- 通过函数名返回计算后的值 -->
  {{value}}
</div>
```

计算属性可以在 vue 组件的 `computed` 属性中定义。以下是计算属性的基本语法：

```
<script>
  // 实例化
  let app = new Vue({
    el: '#div1',
    data: {
    },
    computed: { // 计算属性可以在Vue组件的computed属性中定义。
      value () {
        // 通过return将 计算的结果返回
        return 1024
      }
    }
  })
</script>
```

计算属性可以依赖于其他响应式数据（包括计算属性本身）或者 Vue 实例的数据。当依赖的数据发生变化时，计算属性会重新计算。

实战案例：

```
<div id="div2">
  <h1>要求如下：
    点击购买按钮，自动计算订单件数、总价（每件商品价格为10元） </h1>
  <button @click="shopping">点击购买商品</button>
  <h3>当前订单件数:{{count}}</h3>
  <h2>当前订单总价:{{jiaGe}}</h2>
</div>
```

vue 代码:

```
<script>
  let div2 = new Vue({
    el: "#div2",
    data : {
      count : 0
    },
    methods:{
      shopping(){
        this.count++
      }
    },
    computed: {
      jiaGe(){
        return this.count*10
      }
    }
  })
</script>
```

二、侦听器watch

在 vue 中, watch (侦听器) 是一种用于监听数据变化并执行相应操作的特殊选项。通过定义一个 watch 对象, 你可以监视指定的数据, 并在数据变化时触发相应的回调函数。

watch 选项可以在 vue 组件的选项对象中进行定义。以下是 watch 选项的基本语法展示:

```
<script>
  let div3 = new Vue({
    el: "",      // 挂载
    data : {
      // 数据源
    },
    methods:{
      //方法区
    },
    computed: {
      //计算属性
    },
    watch:{
      // 侦听器
    }
  })
</script>
```

- 普通监听

语法:

```

watch: { // 侦听器
  侦听对象: {
    // 当 侦听对象 发生变化时, handler函数将被调用, 并传入两个参数, 分别表示更新后的
    值和更新前的值。
    // 侦听器对象中的一个属性, 用来指定当侦听的对象发生变化时所执行的逻辑代码。
    handler(newVal, oldVal){
      // 更新后的值 newVal,更新前的值 oldVal
      console.log('侦听到事情后 一般我们会做一些事情 就可以在侦听器内 来写逻辑代
码');
    }
  }
}

```

案例:

```

<div id="div3">
  <h1>要求如下: 点击购买按钮, 自动计算订单件数、总价 (每件商品价格为10元) </h1>
  <button @click="shopping">点击购买商品</button>
  <h3>当前订单件数: {{count}}</h3>
  <h2>当前订单总价: {{jiaGe}}</h2>
</div>

```

```

<script>
  let div3 = new Vue({
    el: "#div3",
    data: {
      count : 0
      // count: {
      //   name: '我是个对象 你无法侦听我'
      // }
    },
    methods: {
      shopping() {
        this.count++
      }
    },
    computed: {
      jiaGe() {
        return this.count * 20
      }
    },
    watch: { // 侦听器
      // 侦听对象: count
      count: {
        // 当count对象发生变化时, handler函数将被调用
        handler(newVal, oldVal){
          console.log('更新后的值', newVal);
          console.log('更新前的值', oldVal);
        }
      }
    }
  })
</script>

```

- 深度监听(deep)

深度监听 (deep) 是 vue 中 watch 选项的一个配置项, 用于开启对对象或数组内部属性的深度观测。

默认情况下, vue 的侦听器 (watch) 只会对数据的引用进行观察, 当引用发生变化时才会触发侦听器。而深度监听 (deep) 可以让 vue 深入观察对象或数组内部的属性, 并在其内部属性发生变化时也触发侦听器。

一般用于监听对象, 可以深度监听到对象中的值

```
<div id="div4">
  <h1>要求如下:
    点击购买按钮, 自动计算订单件数、总价 (每件商品价格为10元)
  </h1>
  <button @click="shopping">点击购买商品</button>
  <!-- 将 数据源 count 换成对象 -->
  <h3>当前订单件数:{{count.money}}</h3>
  <h2>当前订单总价:{{jiage}}</h2>
</div>
```

深度监听 deep 默认为关闭 false, 打开将 deep 设置为 true

```
watch: { // 侦听器
  侦听对象:{
    // 这里的i是默认形参名, 可以随意指定。
    handler(i){
      console.log(i.键);
      console.log('侦听到事情后 一般我们会做一些事情 就可以在侦听器内 来写逻辑代码');
    },
    // 开启侦听器, 深度监听到对象中每一个属性的变化
    deep : true
  }
}
```

```
<script>
// 实例化
let div4 = new vue({
  el: "#div4",
  data: {
    count: { //将 数据源 count 换成对象
      money:0
    }
  },
  methods: {
    shopping() {
      this.count.money++
    }
  },
  computed: {
    jiage() {
      return this.count.money * 10
    }
  }
})
```

```

    }
  },
  watch: { // 侦听器
    // 侦听对象: count
    count: {
      // 函数handler有1个默认形参
      handler(i) {
        console.log(i.money);
        console.log('侦听到事情后 一般我们会做一些事情 就可以在侦听器内
来写逻辑代码');
      },
      // 开启侦听器
      // 控制侦听器深度监听到对象中每一个属性的变化
      deep: true
    }
  }
}
})
</script>

```

- 自动触发

```

// 可以控制侦听器自动触发一次 ,默认为false
immediate: true

```

```

<script>
  // 实例化
  let div4 = new Vue({
    el: "#div4",
    data: {
      count: { //将 数据源 count 换成对象
        money: 0
      }
    },
    methods: {
      shopping() {
        this.count.money++
      }
    },
    computed: {
      jiaGe() {
        return this.count.money * 10
      }
    },
    watch: { // 侦听器
      // 侦听对象: count
      count: {
        // 函数handler有两个默认形参
        // 分别代表 更新后和更新前的值
        handler(i) {
          console.log(i.money);
          console.log('侦听到事情后 一般我们会做一些事情 就可以在侦听器内
来写逻辑代码');
        },

```

```

        // 开启侦听器
        // 控制侦听器深度监听到对象中每一个属性的变化
        deep : true ,
        // 可以控制侦听器自动触发一次 ,默认为false
        immediate: true
    }
}
})
</script>

```

三、ref 获取 dom

在 `vue` 中, 可以使用 `ref` 特殊属性来获取DOM元素的引用。 `ref` 可以用于在 `vue` 组件中给DOM元素添加一个标识符, 然后通过 `this.$refs` 来访问该DOM元素的引用。

```

<div id="div5" >
  <!--通过ref来获取这个dom 取名叫domref -->
  <div ref="domref">
    我是被操作的dom
  </div>
  <button @click="setdom">点击</button>
  (通过操控dom的方式 来使div变成红色)
</div>

<script>
  // 实例化
  let div5 = new vue({
    el: "#div5",
    methods: {
      setdom() {
        // this.$refs.自定义的ref标识符.style.样式属性 = 样式属性值
        // this.$refs.domref 找到dom domref 然后通过style.color 设置颜色
        this.$refs.domref.style.color = 'red'
        this.$refs.domref.style.height = '100px'
        //在JavaScript中, 使用连字符(-)的CSS属性名需要使用驼峰命名法。
        this.$refs.domref.style.backgroundColor = 'pink'
      }
    }
  })
</script>

```

- `ref`是 `vue` 里的方法, 更加安全, 不会依赖class或者id的样式变了而影响布局
- `vue` 的主要目的是减少 `dom` 的操作。减少 `dom` 节点的消耗

