03-指令补充+样式绑定+计算属性+侦听器

一、指令补充

1. 指令修饰符

1.1 什么是指令修饰符?

所谓指令修饰符就是让指令的`功能更强大，书写更便捷`

1.2 分类

1.2.1 按键修饰符

- @keydown.enter：当enter键按下时触发
- @keyup.enter：当enter键抬起时触发

代码演示:

```
<script setup>
  // 处理按键函数
  const onKeyDown = () => {
    console.log('onKeyDown')
  }
</script>
<input type="text" @keydown.enter="onKeyDown" />
```

1.2.2 事件修饰符

1. @事件名.stop —> 阻止冒泡
2. @事件名.prevent —>阻止默认行为
3. @事件名.stop.prevent —>可以连用 即阻止事件冒泡也阻止默认行为

```
<script setup>

  // p标签的点击事件
  const onPClick = () => {
    console.log('onPClick')
  }

  // div标签的点击事件
```

```
  const onDivClick = () => {
    console.log('onDivClick')
  }
</script>

<template>
  <!-- 事件修饰符 -->
  <div @click="onDivClick">
    <!-- .prevent: 阻止默认行为 -->
    <a href="https://baidu.com" @click.prevent>百度一下</a>
    <!-- .stop: 阻止冒泡，同名事件不会向上级传递 -->
    <p @click.stop="onPClick"></p>

    <!-- 修饰符的链式调用: 表明两个同时阻止 -->
    <a href="https://baidu.com" @click.stop.prevent>百度一下</a>
  </div>
</template>

<style>
  div {
    width: 400px;
    height: 200px;
    background: plum;
  }
  div a {
    display: block;
    width: 100px;
    text-decoration: none;
    background: tomato;
    text-align: center;
    color: #fff;
  }
  div p {
    width: 200px;
    height: 100px;
    background: rebeccapurple;
  }
</style>
```


1.2.3 v-model修饰符

- v-model.trim —> 去除首尾空格
- v-model.number —> 尝试用parseFloat()转数字
- v-model.lazy —> 失去焦点时同步数据，而不是输入时同步数据

```vue
<script setup>
  import { reactive } from 'vue'
  const goods = reactive({
    name: '',
    price: ''
  })
</script>

<template>
  <div>
    <!-- .lazy修饰符 -->
    <!-- 名称:
    <input
      type="text"
      v-model.lazy="goods.name" /><br /><br /> -->

    <!-- .trim修饰符 -->
    名称:
    <input
      type="text"
      v-model.trim="goods.name" /><br /><br />

    <!-- .number修饰符 -->
    价格:
    <input
      type="text"
      v-model.number="goods.price" /><br /><br />
  </div>
</template>

<style scoped></style>
```


1.3 总结

1. 如何监听回车键?

   答: 1、`@keydown.enter` -> 回车按下

   2、@keyup.enter -> 回车抬起

2. 如何阻止默认行为、阻止冒泡?

   答: 1、@click.prevent -> 阻止默认行为

   2、@click.stop -> 阻止冒泡

   3、`@click.prevent.stop` -> 可以链式调用, 二者都阻止

3. v-model的3个修饰符的作用是什么?

答: 1、.lazy -> 失去焦点再同步

2、.trim -> 去除首尾空格

3、.number -> 尝试转数字

2. v-model用在其他表单元素上

2.1 讲解内容

常见的表单元素都可以用 v-model 绑定关联, 作用是可以快速 获取 或 设置 表单元素的值

它会根据 控件类型 自动选取 正确的属性 来更新元素

```
输入框    input:text  ——> value
文本域    textarea    ——> value
下拉菜单  select   ——> value
单选框    input:radio   ——> value
复选框    input:checkbox  ——> checked / value
```

2.2 代码准备

```
<script setup>
  import { ref } from 'vue'
  // 自我介绍
  const intro = ref('')

  //  收集城市
  const city = ref('SH')

  // 收集血型
  const blood = ref('ab')

  // 是否同意用户协议
  const isAgree = ref(false)

  // 收集爱好
  const hobby = ref(['ZQ', 'PB'])
</script>

<template>
  <div>
    <!-- 文本域 -->
    <textarea
      v-model="intro"
      cols="30"
```

```
25        rows="4"
26        placeholder="请输入自我介绍"></textarea>
27
28      <br />
29      <br />
30      <!-- 下菜菜单 -->
31      <select v-model="city">
32        <option value="BJ">北京</option>
33        <option value="SH">上海</option>
34        <option value="SZ">深圳</option>
35        <option value="HZ">杭州</option>
36      </select>
37      <br />
38      <br />
39      <!-- 单选框：多个当中只能选择一个，需要给单选框手动添加 value 属性 -->
40      <input
41        type="radio"
42        value="a"
43        v-model="blood" />A
44      <input
45        type="radio"
46        value="b"
47        v-model="blood" />B
48      <input
49        type="radio"
50        value="ab"
51        v-model="blood" />AB
52      <input
53        type="radio"
54        value="o"
55        v-model="blood" />O
56
57      <br />
58      <br />
59      <input
60        type="checkbox"
61        v-model="isAgree" />是否同意用户协议
62
63      <br />
64      <br />
65      <input
66        v-model="hobby"
67        type="checkbox"
68        value="LQ" />篮球
69      <input
70        v-model="hobby"
71        type="checkbox"
```

```
72        value="ZQ" />足球
73      <input
74        v-model="hobby"
75        type="checkbox"
76        value="YMQ" />羽毛球
77      <input
78        v-model="hobby"
79        type="checkbox"
80        value="PPQ" />乒乓球
81      <br />
82      <input
83        v-model="hobby"
84        type="checkbox"
85        value="PB" />跑步
86      <input
87        v-model="hobby"
88        type="checkbox"
89        value="YY" />游戏
90      <input
91        v-model="hobby"
92        type="checkbox"
93        value="PLT" />普拉提
94      <input
95        v-model="hobby"
96        type="checkbox"
97        value="LDW" />拉丁舞
98    </div>
99  </template>
100
101 <style scoped></style>
```


2.3 总结

1. v-model如何收集下拉列表的值?

   答: `v-model写在select上， 关联是选中option的`value`

2. v-model如何收集单选框的值?

   答: 给单选框添加value属性, v-model收集选中单选框的value

3. v-model作用在复选框上组要注意什么?

   答: 1、一个复选框, v-model绑定 `布尔值` , 关联 `checked` 属性

   2、一组复选框, v-model绑定 `数组` , 关联 `value` 属性, 给复选框 `手动添加 value`

二、样式绑定

1. 基本介绍

1概述

为了方便开发者进行样式控制, Vue 扩展了 v-bind 的语法, 可以针对 **class 类名** 和 **style 行内样式**

两个属性进行控制, 进而通过数据控制元素的样式

2、分类

2.1、操作class

2.2、操作style

2、操作class

1、语法

`:class = "三元表达式 / 对象"`

2、三元表达式

```
1 <p :class="条件 ? '类名1' : '类名2'"></p>
```

3、对象语法

当class动态绑定的是**对象**时，**键就是类名，值就是布尔值**，如果值是**true**，就添加这个类，否则删除这个类

```
1 <p class="box" :class="{ 类名1: 布尔值1, 类名2: 布尔值2 }"></p>
```

适用场景：一个类名，来回切换

4、静态class与动态class共存

静态class与动态class共存可以**共存**，二者会**合并**

```
1 <p class="item" :class="条件 ? '类名1' : '类名2'"></p>
```

4、代码演示

```vue
<script setup>
  import { ref } from 'vue'

  // 是否处于激活
  const isActive = ref(true)
</script>

<template>
  <div>
    <!-- 1. 三元绑定 -->
    <p :class="isActive ? 'active' : ''">Active1</p>
    <!-- 2. 对象绑定 -->
    <p :class="{ active: isActive }">Active2</p>
    <!-- 3. 静态class与动态class共存 -->
    <p class="item" :class="{ active: isActive }">Active3</p>
  </div>
</template>

<style>
  .active {
    color: red;
  }
</style>
```

5、总结

1. 如何通过class动态控制元素的样式?

   答: 1、`:class="三元表达式"`

   2、`:class="{ 类名: 布尔值 }"`,布尔值为true则添加类名;否则移除

2. 静态和动态class可以共存吗?

   答:可以`共存, 二者会合并`

3、京东秒杀-tab栏切换导航高亮

1、需求

当我们点击哪个tab页签时,哪个tab页签就高亮

2、静态代码(可复制)

```vue
<script setup>
  // tabs 列表
  const tabs = [
    { id: 1, name: '京东秒杀' },
    { id: 2, name: '每日特价' },
    { id: 3, name: '品类秒杀' }
  ]
</script>

<template>
  <div>
     <ul>
      <li><a class="active" href="#">京东秒杀</a></li>
    </ul>
  </div>
</template>

<style>
  * {
    margin: 0;
    padding: 0;
  }
  ul {
    display: flex;
    border-bottom: 2px solid #e01222;
    padding: 0 10px;
  }
  li {
    width: 100px;
    height: 50px;
    line-height: 50px;
    list-style: none;
    text-align: center;
  }
  li a {
    display: block;
    text-decoration: none;
    font-weight: bold;
    color: #333333;
  }
  li a.active {
    background-color: #e01222;
    color: #fff;
  }
</style>
```

3、思路

1. 基于数据，动态渲染tab（v-for）
2. 准备一个下标 记录高亮哪一个 tab
3. 基于下标动态切换class的类名

4、完整代码

```vue
<script setup>
  import { ref } from 'vue'

  // tabs 列表
  const tabs = [
    { id: 1, name: '京东秒杀' },
    { id: 2, name: '每日特价' },
    { id: 3, name: '品类秒杀' }
  ]
  // 当前高亮的下标，默认是第一个，下标为 0
  const currentIndex = ref(0)

</script>

<template>
  <div>
    <ul>
      <li
        v-for="(item, index) in tabs"
        :key="item.id"
        @click="currentIndex = index">
        <a
          :class="{ active: currentIndex === index }"
          href="#"
          >{{ item.name }}</a
        >
      </li>
    </ul>
  </div>
</template>

<style>
  * {
    margin: 0;
    padding: 0;
  }
  ul {
```

```
    display: flex;
    border-bottom: 2px solid #e01222;
    padding: 0 10px;
  }
  li {
    width: 100px;
    height: 50px;
    line-height: 50px;
    list-style: none;
    text-align: center;
  }
  li a {
    display: block;
    text-decoration: none;
    font-weight: bold;
    color: #333333;
  }
  li a.active {
    background-color: #e01222;
    color: #fff;
  }
</style>

```

4、操作style

1、语法

```
<div class="box" :style="{ CSS属性名1: CSS属性值, CSS属性名2: CSS属性值 }"></div>
```

2、代码演示

```
<script setup>
  import { reactive } from 'vue'

  // 行内样式对象
  const styleObj = reactive({
    color: '#fff',
    backgroundColor: 'purple'
  })
</script>
<template>
  <div>
```

```
    <p :style="styleObj">Some Text...</p>
  </div>
</template>
<style></style>
```

3、进度条案例

```
<script setup></script>

<template>
  <div class="progress">
    <div class="inner" style="width: 50%">
      <span>50%</span>
    </div>
  </div>
  <button>设置25%</button>
  <button>设置50%</button>
  <button>设置75%</button>
  <button>设置100%</button>
</template>

<style>
  .progress {
    height: 25px;
    width: 400px;
    border-radius: 15px;
    background-color: #272425;
    border: 3px solid #272425;
    box-sizing: border-box;
    margin-bottom: 30px;
  }
  .inner {
    height: 20px;
    border-radius: 10px;
    text-align: right;
    position: relative;
    background-color: #409eff;
    background-size: 20px 20px;
    box-sizing: border-box;
    transition: all 1s;
  }
  .inner span {
    position: absolute;
    right: -25px;
    bottom: -25px;
```

```
  }
</style>
```

4、完整代码

```
<script setup>
  import { ref } from 'vue'

  // 份数
  const x = ref(0)
</script>

<template>
  <div class="progress">
    <div
      class="inner"
      :style="{ width: `${(x / 4) * 100}%` }">
      <span>{{ (x / 4) * 100 }}%</span>
    </div>
  </div>
  <button @click="x = 1">设置25%</button>
  <button @click="x = 2">设置50%</button>
  <button @click="x = 3">设置75%</button>
  <button @click="x = 4">设置100%</button>
</template>

<style>
  .progress {
    height: 25px;
    width: 400px;
    border-radius: 15px;
    background-color: #272425;
    border: 3px solid #272425;
    box-sizing: border-box;
    margin-bottom: 30px;
  }
  .inner {
    height: 20px;
    border-radius: 10px;
    text-align: right;
    position: relative;
    background-color: #409eff;
    background-size: 20px 20px;
    box-sizing: border-box;
    transition: all 1s;
```

```
  }
  .inner span {
    position: absolute;
    right: -25px;
    bottom: -25px;
  }
</style>
```

5、总结

1. 如何给元素动态绑定style?

   答: `:style="{ 属性名1: 表达式1, 属性名2: 表达式2, ... }"`

三、计算属性

1. 基本使用

1.1 概念

基于**现有的数据**，计算出来的**新数据; 当现有的数据变化，会自动**重新计算。

1.2 语法

1. 使用 **computed** 函数，计算得到一个新数据进行展示

```
const 新数据 = computed(() => {
  // some code ...
  return 结果
})

// 商品列表(原始数据)
const goodsList = ref([
  { id: 1, name: '篮球', num: 1 },
  { id: 2, name: '玩具', num: 3 },
  { id: 3, name: '书籍', num: 2 }
])
```

1.3 案例

比如我们可以使用计算属性实现下面这个业务场景

1.4 静态代码(可复制)

```vue
<script setup>

  import { ref } from 'vue'

  // 商品列表(原始数据)
  const goodsList = ref([
    { id: 1, name: '篮球', num: 1 },
    { id: 2, name: '玩具', num: 3 },
    { id: 3, name: '书籍', num: 2 }
  ])

</script>
<template>
  <h3>比特人的礼物清单</h3>
  <table>
    <tr>
      <th>名字</th>
      <th>数量</th>
    </tr>
    <tr
      v-for="item in goodsList"
      :key="item.id">
      <td>{{ item.name }}</td>
      <td>{{ item.num }}个</td>
    </tr>
   </table>
  <!-- 目标：统计求和，展示礼物总数 -->
  <p>礼物总数：？个</p>
</template>

<style>
  table {
    width: 350px;
    border: 1px solid #333;
  }
  table th,
  table td {
    border: 1px solid #333;
  }
  table td {
    text-align: center;
  }
</style>
```

1.5 完整代码

```vue
<script setup>
  import { computed, ref } from 'vue'
  // 商品列表(原始数据)
  const goodsList = ref([
    { id: 1, name: '篮球', num: 1 },
    { id: 2, name: '玩具', num: 3 },
    { id: 3, name: '书籍', num: 2 }
  ])

  // 由于这里只有原始数据，没有直接给我们提供商品总数量
  // 但是我们可以基于现有的 goodsList 计算得到总数量
  // 那么推荐把总数量声明为 计算属性
  const totalNum = computed(() => {
    return goodsList.value.reduce((prev, item) => prev + item.num, 0)
  })
</script>

<template>
  <h3>比特人的礼物清单</h3>
  <table>
    <thead>
      <tr>
        <th>名字</th>
        <th>数量</th>
      </tr>
    </thead>
    <tbody>
      <tr
        v-for="item in goodsList"
        :key="item.id">
        <td>{{ item.name }}</td>
        <td>{{ item.num }}</td>
      </tr>
    </tbody>
  </table>
  <p>礼物总数: {{ totalNum }} 个</p>
</template>

<style>
  table {
    width: 350px;
    border: 1px solid #333;
  }
  table th,
```

```
45    table td {
46      border: 1px solid #333;
47    }
48    table td {
49      text-align: center;
50    }
51  </style>
```

1.6 注意

1. 计算属性必须 `有返回值`
2. 使用和 ref/reactive 数据一样 , 可用于 `插值` , 也可 `配合指令`

1.7 总结

1. 何时用计算属性?

   答：基于现在数据计算得到新数据的时候

2. 语法?

```
1  const 新数据 = computed(() => {
2    return 计算结果
3  })
```

2. 计算属性 VS 普通函数

2.1 计算属性

2.1.1 作用

封装了一段对于基于现有数据，计算求得一个**新数据**

2.1.2 语法

1. 写在computed函数中，必须返回
2. 作为属性，直接使用
   - js中获取计算属性： 计算属性.value
   - 模板中使用计算属性：{{ 计算属性 }} 或 配合指令

2.2 普通函数

2.2.1 作用

封装一段js代码，用于调用以**处理业务逻辑**。

2.2.2 语法

1. 直接在script下定义
2. 作为函数调用
   - js中调用：函数名(实参列表)
   - 模板中调用 {{ 函数名 (实参列表) }} 或者 @事件名=“方法名(实参列表)”
3. 计算属性的优势
   - 缓存特性(提升性能)， 函数没有缓存特性
   - 计算属性会对计算出来的结果缓存, 再次使用直接读取缓存
   - 只有依赖项变了, 才会自动重新计算, 并再次缓存新数据

```vue
<script setup>
  import { ref, computed } from 'vue'

  // 商品列表(原始数据)
  const goodsList = ref([
    { id: 1, name: '篮球', num: 1 },
    { id: 2, name: '玩具', num: 3 },
    { id: 3, name: '书籍', num: 2 }
  ])

  // 计算总数
  const totalNum = computed(() => {
    return goodsList.value.reduce((prev, item) => prev+ item.num, 0)
  })
</script>
<template>
  <div id="app">
    <h3>比特人的礼物清单</h3>
    <table>
      <tr>
        <th>名字</th>
        <th>数量</th>
      </tr>
      <tr
        v-for="item in goodsList"
        :key="item.id">
        <td>{{ item.name }}</td>
        <td>{{ item.num }}个</td>
      </tr>
    </table>

    <!-- 目标：统计求和，求得礼物总数 -->
```

```
    <p>礼物总数：{{ totalNum }} 个</p>
  </div>
</template>

<style>
  #app {
    width: 240px;
    margin: 100px auto;
  }
  table {
    border: 1px solid #000;
    text-align: center;
    width: 240px;
  }
  th,
  td {
    border: 1px solid #000;
  }
  h3 {
    position: relative;
  }
</style>
```

2.3 总结

1. 计算属性与普通函数的区别？

   答： 1、计算属性基于依赖 **有缓存特性**，普通函数 **没有缓存**

   2、当基于现有的数据计算得到新数据，推荐使用计算属性

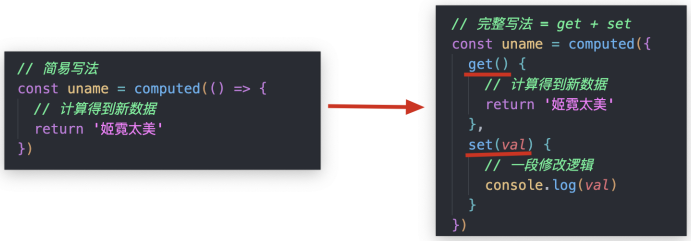
   3、当处理业务逻辑时，推荐使用函数，比如点击事件的处理函数

3. 计算属性的完整写法

3.1 思考

既然计算属性也是属性，能访问，应该也能修改了？

1. 计算属性默认的写法，只能读(展示数据)，不能 "改"
2. 如果要 "改" → 需要用计算属性的完整写法



3.2 代码演示

```vue
<script setup>
  import { computed } from 'vue'

  // 完整写法 = get + set
  const uname = computed({
    // 使用计算属性的时候，自动触发get，get内部必须 `return 计算结果`
    get() {
      // 计算得到新数据
      return '姬霓太美'
    },
    // 给计算属性赋值的时候，自动触发set，并接收赋予的新值val
    set(val) {
      // 一段修改逻辑
      console.log(val)
    }
  })
</script>
<template>
  <input type="text" v-model="uname" />
</template>
```


3.3 总结

1. 何时使用计算属性完整写法?

   答：当 `修改计算属性` 时, 也就是当计算属性配合v-model的时候

2. 完整写法的是?

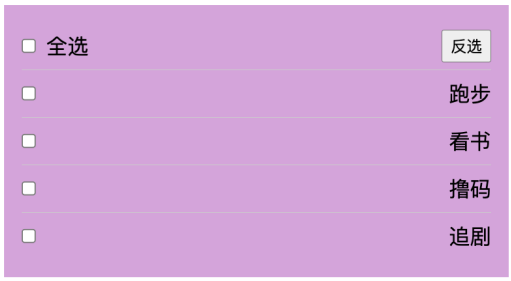
   答：`get + set`

```vue
const xxx = computed({
  get() {
    return 计算结果
  },
  set(val) {
```

```
6
7   }
8 })
```

4. 案例 - 全返反选

4.1 效果



4.2 静态代码(可复制)

```
1 <script setup>
2   import { ref } from 'vue'
3
4   // 计划列表
5   const planList = ref([
6     { id: 12, name: '跑步', done: false },
7     { id: 76, name: '看书', done: false },
8     { id: 31, name: '撸码', done: false },
9     { id: 49, name: '追剧', done: false }
10  ])
11
12 </script>
13
14 <template>
15   <p>
16     <span>
17       <input type="checkbox" id="all" />
18       <label for="all">全选</label>
19     </span>
20     <button>反选</button>
21   </p>
22   <ul>
23     <li>
24       <input type="checkbox" />
25       <span>xxx</span>
26     </li>
27   </ul>
```

```vue
</template>

<style lang="scss">
  * {
    margin: 0;
    padding: 0;
  }
  div {
    width: 400px;
    margin: 100px auto;
    padding: 15px;
    font-size: 18px;
    background: plum;
    p {
      display: flex;
      justify-content: space-between;
      align-items: center;
      height: 40px;
      button {
        padding: 3px 6px;
      }
    }
  }

  ul {
    list-style: none;
    li {
      display: flex;
      align-items: center;
      justify-content: space-between;
      height: 40px;
      border-top: 1px solid #ccc;
      span.completed {
        color: #ddd;
        text-decoration: line-through;
      }
    }
  }

  input {
    margin-right: 8px;
  }
</style>
```


4.3 完整代码

```vue
<script setup>
  import { computed, ref } from 'vue'
  // 计划列表
  const planList = ref([
    { id: 12, name: '跑步', done: false },
    { id: 76, name: '看书', done: false },
    { id: 31, name: '撸码', done: false },
    { id: 49, name: '追剧', done: false }
  ])
  // 是否全部选中
  const isAll = computed({
    // 使用计算属性自动触发 get
    get() {
      // every: 检测每一个
      return planList.value.every((item) => item.done)
    },
    // 修改计算属性自动触发 set
    set(val) {
      // val: 给计算属性赋予的新值，在这里就是表全选复选框的状态

      // 遍历 planList 数组，把每个小选的 done 属性与 val 保持一致即可
      planList.value.forEach((item) => (item.done = val))
    }
  })

  // 反选
  const onToggle = () => {
    // 遍历 planList 数组，对每个对象的 done 属性取反即可
    planList.value.forEach((item) => (item.done = !item.done))
  }
</script>
<template>
  <p>
    <span>
      <input
        v-model="isAll"
        type="checkbox"
        id="all" />
      <label for="all">全选</label>
    </span>
    <button @click="onToggle">反选</button>
  </p>
  <ul>
    <li
      v-for="item in planList"
      :key="item.id">
      <input
```

```
48          type="checkbox"
49          v-model="item.done" />
50        <span :class="{ completed: item.done }">{{ item.name }}</span>
51      </li>
52    </ul>
53  </template>
54
55  <style lang="scss">
56    // npm i sass -D
57    // npm run dev
58    * {
59      margin: 0;
60      padding: 0;
61    }
62    ul {
63      list-style: none;
64    }
65    #app {
66      width: 400px;
67      margin: 100px auto;
68      padding: 15px 18px;
69      background: plum;
70      p {
71        display: flex;
72        justify-content: space-between;
73        align-items: center;
74        height: 40px;
75        border-bottom: 1px solid #ccc;
76        button {
77          padding: 3px 6px;
78        }
79        input {
80          margin-right: 8px;
81        }
82      }
83      ul {
84        li {
85          display: flex;
86          justify-content: space-between;
87          align-items: center;
88          height: 40px;
89          border-bottom: 1px solid #ccc;
90          span.completed {
91            color: #ddd;
92            text-decoration: line-through;
93          }
94        }
```

```
    }
  }
</style>
```

四、侦听器

1. 作用

监视数据变化, 执行DOM操作或异步操作

2. 语法

1. 监视简单类型

```
<script setup>
  import { ref, watch } from 'vue'

  // 搜索框关键字
  const keyword = ref('')

  // 监视 keyword，只要 keyword 的值变了，就会执行回调函数
  watch(keyword,(newVal, oldVal) => {
    // newVal: 新值
    // oldVal: 旧值
  })

</script>
<template>
  <div>
    <input
      type="text"
      v-model="keyword"
      placeholder="请输入关键字" />
  </div>
</template>
```

2. 监视复杂类型

```
<script setup>
  import { reactive, watch } from 'vue'

  // 表单对象
  const obj = reactive({
```

```
    username: '',
    password: ''
  })

  // 监视 obj 的变化
  watch(obj, (newVal, oldVal) => {
    console.log(newVal, oldVal)
  })
</script>
<template>
  <div>
    <input
      type="text"
      v-model="obj.username"
      placeholder="请输入用户名" />
    <br/>
    <br/>
    <input
      type="text"
      v-model="obj.password"
      placeholder="请输入密码" />
  </div>
</template>

```

3. 总结

1. watch的作用是?

   答: 监视响应式数据的变化, 当数据变了, 针对性的DOM操作或异步操作

五、案例成绩管理

1. 效果

2. 静态代码(可复制)

```
<script setup>
  import { ref } from 'vue'

```

```vue
  // 成绩列表
  const scoreList = ref([
    { id: 19, subject: '语文', score: 94 },
    { id: 27, subject: '数学', score: 59 },
    { id: 12, subject: '英语', score: 92 }
  ])
</script>

<template>
  <div class="score-case">
    <div class="table">
      <table>
        <thead>
          <tr>
            <th>编号</th>
            <th>科目</th>
            <th>成绩</th>
            <th>操作</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>1</td>
            <td>语文</td>
            <td class="red">46</td>
            <td><a href="#">删除</a></td>
          </tr>
          <tr>
            <td>2</td>
            <td>英语</td>
            <td>80</td>
            <td><a href="#">删除</a></td>
          </tr>
        </tbody>
        <tbody>
          <tr>
            <td colspan="5">
              <span class="none">暂无数据</span>
            </td>
          </tr>
        </tbody>

        <tfoot>
          <tr>
            <td colspan="5">
              <span>总分: 246</span>
              <span style="margin-left: 50px">平均分: 79</span>
```

```
            </td>
          </tr>
        </tfoot>
      </table>
    </div>
    <form class="form">
      <div class="form-item">
        <div class="label">科目: </div>
        <div class="input">
          <input
            type="text"
            placeholder="请输入科目" />
        </div>
      </div>
      <div class="form-item">
        <div class="label">分数: </div>
        <div class="input">
          <input
            type="text"
            placeholder="请输入分数" />
        </div>
      </div>
      <div class="form-item">
        <div class="label"></div>
        <div class="input">
          <button class="submit">添加</button>
        </div>
      </div>
    </form>
  </div>
</template>
<style>
  .score-case {
    width: 1000px;
    margin: 50px auto;
    display: flex;
  }
  .score-case .table {
    flex: 4;
  }
  .score-case .table table {
    width: 100%;
    border-spacing: 0;
    border-top: 1px solid #ccc;
    border-left: 1px solid #ccc;
  }
  .score-case .table table th {
```

```
    background: #f5f5f5;
  }
  .score-case .table table tr:hover td {
    background: #f5f5f5;
  }
  .score-case .table table td,
  .score-case .table table th {
    border-bottom: 1px solid #ccc;
    border-right: 1px solid #ccc;
    text-align: center;
    padding: 10px;
  }
  .score-case .table table td.red,
  .score-case .table table th.red {
    color: red;
  }
  .score-case .table .none {
    height: 100px;
    line-height: 100px;
    color: #999;
  }
  .score-case .form {
    flex: 1;
    padding: 20px;
  }
  .score-case .form .form-item {
    display: flex;
    margin-bottom: 20px;
    align-items: center;
  }
  .score-case .form .form-item .label {
    width: 60px;
    text-align: right;
    font-size: 14px;
  }
  .score-case .form .form-item .input {
    flex: 1;
  }
  .score-case .form .form-item input,
  .score-case .form .form-item select {
    appearance: none;
    outline: none;
    border: 1px solid #ccc;
    width: 200px;
    height: 40px;
    box-sizing: border-box;
    padding: 10px;
```

```
    color: #666;
  }
  .score-case .form .form-item input::placeholder {
    color: #666;
  }
  .score-case .form .form-item .cancel,
  .score-case .form .form-item .submit {
    appearance: none;
    outline: none;
    border: 1px solid #ccc;
    border-radius: 4px;
    padding: 4px 10px;
    margin-right: 10px;
    font-size: 12px;
    background: #ccc;
  }
  .score-case .form .form-item .submit {
    border-color: #069;
    background: #069;
    color: #fff;
  }
</style>
```

3. 功能描述

3.1 渲染功能

3.2 添加功能

3.3 删除功能

3.4 统计总分，求平均分

3.5 数据持久化

4. 思路分析

- 渲染功能 v-for :key v-bind:动态绑定class的样式
- 删除功能 v-on绑定事件， 阻止a标签的默认行为
- v-model的修饰符 .trim、 .number、 判断数据是否为空后 再添加、添加后清空文本框的数据
- 使用计算属性computed 计算总分和平均分的值

5. 完整代码

```vue
<script setup>
  import { computed, reactive, ref, watch } from 'vue'

  // 本地存储 key
  const SCORE_LIST_KEY = 'score-list-key'

  // 成绩列表
  const scoreList = ref(
    // 从本地取值
    // 有值就用，否则用默认的成绩列表
    JSON.parse(localStorage.getItem(SCORE_LIST_KEY)) || [
      { id: 19, subject: '语文', score: 94 },
      { id: 27, subject: '数学', score: 57 },
      { id: 12, subject: '英语', score: 92 }
    ]
  )

  // 成绩表单
  const scoreForm = reactive({
    subject: '', // 科目
    score: '' // 分数
  })

  // 总分
  const totalScore = computed(() => {
    // reduce 求和
    return scoreList.value.reduce((prev, item) => prev + item.score, 0)
  })
  // 平均分
  const avgScore = computed(() => {
    return scoreList.value.length
      ? totalScore.value / scoreList.value.length
      : 0
  })

  // 删除
  const onDel = (index) => {
    // 弹窗确认
    if (window.confirm('确认删除么?')) {
      scoreList.value.splice(index, 1)
    }
  }

  // 添加
  const onAdd = () => {
    // 非空校验
```

```
    if (!scoreForm.subject || !scoreForm.score)
      return alert('科目或分数不能为空')

    // 添加至数组
    scoreList.value.push({
      ...scoreForm,
      id: Date.now()
    })
    // 清空
    scoreForm.subject = scoreForm.score = ''
  }

  // 监听 scoreList 的变化
  watch(
    scoreList,
    (newVal) => {
      // 存本地
      localStorage.setItem(SCORE_LIST_KEY, JSON.stringify(newVal))
    },
    {
      // 深度监视
      deep: true
    }
  )
</script>

<template>
  <div class="score-case">
    <div class="table">
      <table>
        <thead>
          <tr>
            <th>编号</th>
            <th>科目</th>
            <th>成绩</th>
            <th>操作</th>
          </tr>
        </thead>
        <tbody v-if="scoreList.length">
          <tr
            v-for="(item, index) in scoreList"
            :key="item.id">
            <td>{{ index + 1 }}</td>
            <td>{{ item.subject }}</td>
            <td :class="{ red: item.score < 60 }">{{ item.score }}</td>
            <td>
              <a
```

```
                href="#"
                @click="onDel(index)"
                >删除</a
              >
            </td>
          </tr>
        </tbody>
        <tbody v-else>
          <tr>
            <td colspan="5">
              <span class="none">暂无数据</span>
            </td>
          </tr>
        </tbody>

        <tfoot>
          <tr>
            <td colspan="5">
              <span>总分: {{ totalScore }}</span>
              <span style="margin-left: 50px">平均分: {{ avgScore }}</span>
            </td>
          </tr>
        </tfoot>
      </table>
    </div>
    <form class="form">
      <div class="form-item">
        <div class="label">科目: </div>
        <div class="input">
          <input
            type="text"
            placeholder="请输入科目"
            v-model.trim="scoreForm.subject" />
        </div>
      </div>
      <div class="form-item">
        <div class="label">分数: </div>
        <div class="input">
          <input
            type="number"
            placeholder="请输入分数"
            v-model.number="scoreForm.score" />
        </div>
      </div>
      <div class="form-item">
        <div class="label"></div>
        <div class="input">
```

```
          <button
            class="submit"
            @click.prevent="onAdd">
            添加
          </button>
        </div>
      </div>
    </form>
  </div>
</template>
<style>
  .score-case {
    width: 1000px;
    margin: 50px auto;
    display: flex;
  }
  .score-case .table {
    flex: 4;
  }
  .score-case .table table {
    width: 100%;
    border-spacing: 0;
    border-top: 1px solid #ccc;
    border-left: 1px solid #ccc;
  }
  .score-case .table table th {
    background: #f5f5f5;
  }
  .score-case .table table tr:hover td {
    background: #f5f5f5;
  }
  .score-case .table table td,
  .score-case .table table th {
    border-bottom: 1px solid #ccc;
    border-right: 1px solid #ccc;
    text-align: center;
    padding: 10px;
  }
  .score-case .table table td.red,
  .score-case .table table th.red {
    color: red;
  }
  .score-case .table .none {
    height: 100px;
    line-height: 100px;
    color: #999;
  }
```

```
.score-case .form {
  flex: 1;
  padding: 20px;
}
.score-case .form .form-item {
  display: flex;
  margin-bottom: 20px;
  align-items: center;
}
.score-case .form .form-item .label {
  width: 60px;
  text-align: right;
  font-size: 14px;
}
.score-case .form .form-item .input {
  flex: 1;
}
.score-case .form .form-item input,
.score-case .form .form-item select {
  appearance: none;
  outline: none;
  border: 1px solid #ccc;
  width: 200px;
  height: 40px;
  box-sizing: border-box;
  padding: 10px;
  color: #666;
}
.score-case .form .form-item input::placeholder {
  color: #666;
}
.score-case .form .form-item .cancel,
.score-case .form .form-item .submit {
  appearance: none;
  outline: none;
  border: 1px solid #ccc;
  border-radius: 4px;
  padding: 4px 10px;
  margin-right: 10px;
  font-size: 12px;
  background: #ccc;
}
.score-case .form .form-item .submit {
  border-color: #069;
  background: #069;
  color: #fff;
}
```

```
235 </style>
```