day03

组件

- 组件 (Component) 是 Vue.js 最强大的功能之一
- 组件可以扩展 HTML 元素，封装可重用的代

组件注册

全局注册

- Vue.component('组件名称', { }) 第1个参数是标签名称，第2个参数是一个选项对象
- **全局组件**注册后，任何**vue实例**都可以用

组件基础用

```
<div id="example">
  <!-- 2、 组件使用 组件名称 是以HTML标签的形式使用  -->
  <my-component></my-component>
</div>
<script>
    //   注册组件
    // 1、 my-component 就是组件中自定义的标签名
    Vue.component('my-component', {
      template: '<div>A custom component!</div>'
    })

    // 创建根实例
    new Vue({
      el: '#example'
    })

</script>
```

组件注意事项

- 组件参数的data值必须是函数同时这个函数要求返回一个对象
- 组件模板必须是单个根元素
- 组件模板的内容可以是模板字符串

```
  <div id="app">
     <!--
        4、   组件可以重复使用多次
          因为data中返回的是一个对象所以每个组件中的数据是私有的
          即每个实例可以维护一份被返回对象的独立的拷贝
    -->
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
```

```
      <!-- 8、必须使用短横线的方式使用组件 -->
     <hello-world></hello-world>
  </div>

<script type="text/javascript">
    //5  如果使用驼峰式命名组件，那么在使用组件的时候，只能在字符串模板中用驼峰的方式使用组件，
    // 7、但是在普通的标签模板中，必须使用短横线的方式使用组件
     Vue.component('HelloWorld', {
      data: function(){
        return {
          msg: 'HelloWorld'
        }
      },
      template: '<div>{{msg}}</div>'
    });



    Vue.component('button-counter', {
      // 1、组件参数的data值必须是函数
      // 同时这个函数要求返回一个对象
      data: function(){
        return {
          count: 0
        }
      },
      //  2、组件模板必须是单个根元素
      //  3、组件模板的内容可以是模板字符串
      template: `
        <div>
          <button @click="handle">点击了{{count}}次</button>
          <button>测试123</button>
            #  6 在字符串模板中可以使用驼峰的方式使用组件
           <HelloWorld></HelloWorld>
        </div>
      `,
      methods: {
        handle: function(){
          this.count += 2;
        }
      }
    })
    var vm = new Vue({
      el: '#app',
      data: {

      }
    });
  </script>
```

局部注册

- 只能在当前注册它的vue实例中使用

```
  <div id="app">
      <my-component></my-component>
  </div>


<script>
    // 定义组件的模板
    var Child = {
      template: '<div>A custom component!</div>'
    }
    new Vue({
      //局部注册组件
      components: {
        // <my-component> 将只在父模板可用  一定要在实例上注册了才能在html文件中使用
        'my-component': Child
      }
    })
 </script>
```

Vue 调试工具

Vue组件之间传值

父组件向子组件传值

- 父组件发送的形式是以属性的形式绑定值到子组件身上。
- 然后子组件用属性props接收
- 在props中使用驼峰形式，模板中需要使用短横线的形式字符串形式的模板中没有这个限制

```
  <div id="app">
    <div>{{pmsg}}</div>
     <!--1、menu-item  在 APP中嵌套着 故 menu-item   为  子组件       -->
     <!-- 给子组件传入一个静态的值 -->
    <menu-item title='来自父组件的值'></menu-item>
    <!-- 2、 需要动态的数据的时候 需要属性绑定的形式设置 此时 ptitle  来自父组件data 中的数据 .
          传的值可以是数字、对象、数组等等
    -->
    <menu-item :title='ptitle' content='hello'></menu-item>
  </div>

  <script type="text/javascript">
    Vue.component('menu-item', {
      // 3、 子组件用属性props接收父组件传递过来的数据
      props: ['title', 'content'],
```

```
    data: function() {
      return {
        msg: '子组件本身的数据'
      }
    },
    template: '<div>{{msg + "----" + title + "-----" + content}}</div>'
  });
  var vm = new Vue({
    el: '#app',
    data: {
      pmsg: '父组件中内容',
      ptitle: '动态绑定属性'
    }
  });
</script>
```

子组件向父组件传值

- 子组件用 `$emit()` 触发事件
- `$emit()` 第一个参数为 自定义的事件名称 第二个参数为需要传递的数据
- 父组件用v-on 监听子组件的事件

```
<div id="app">
  <div :style='{fontSize: fontSize + "px"}'>{{pmsg}}</div>
   <!-- 2 父组件用v-on 监听子组件的事件
      这里 enlarge-text  是从 $emit 中的第一个参数对应   handle 为对应的事件处理函数
  -->
  <menu-item :parr='parr' @enlarge-text='handle($event)'></menu-item>
</div>
<script type="text/javascript" src="js/vue.js"></script>
<script type="text/javascript">
  /*
    子组件向父组件传值-携带参数
  */

  Vue.component('menu-item', {
    props: ['parr'],
    template: `
      <div>
        <ul>
          <li :key='index' v-for='(item,index) in parr'>{{item}}</li>
        </ul>
          ###  1、子组件用$emit()触发事件
          ### 第一个参数为 自定义的事件名称    第二个参数为需要传递的数据
        <button @click='$emit("enlarge-text", 5)'>扩大父组件中字体大小</button>
        <button @click='$emit("enlarge-text", 10)'>扩大父组件中字体大小</button>
      </div>
    `
  });
  var vm = new Vue({
    el: '#app',
    data: {
```

```
        pmsg: '父组件中内容',
        parr: ['apple','orange','banana'],
        fontSize: 10
      },
      methods: {
        handle: function(val){
          // 扩大字体大小
          this.fontSize += val;
        }
      }
    });
  </script>
```

兄弟之间的传递

- 兄弟之间传递数据需要借助于事件中心，通过事件中心传递数据
  - 提供事件中心 var hub = new Vue()
- 传递数据方，通过一个事件触发hub.$emit(方法名，传递的数据)
- 接收数据方，通过mounted(){} 钩子中 触发hub.$on()方法名
- 销毁事件 通过hub.$off()方法名销毁之后无法进行传递数据

```
<div id="app">
    <div>父组件</div>
    <div>
      <button @click='handle'>销毁事件</button>
    </div>
    <test-tom></test-tom>
    <test-jerry></test-jerry>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      兄弟组件之间数据传递
    */
    //1、 提供事件中心
    var hub = new Vue();

    Vue.component('test-tom', {
      data: function(){
        return {
          num: 0
        }
      },
      template: `
        <div>
          <div>TOM:{{num}}</div>
          <div>
            <button @click='handle'>点击</button>
          </div>
        </div>
```

```
    `,
    methods: {
      handle: function(){
        //2、传递数据方，通过一个事件触发hub.$emit(方法名，传递的数据)    触发兄弟组件的事件
        hub.$emit('jerry-event', 2);
      }
    },
    mounted: function() {
     // 3、接收数据方，通过mounted(){} 钩子中   触发hub.$on(方法名
      hub.$on('tom-event', (val) => {
        this.num += val;
      });
    }
  });
  Vue.component('test-jerry', {
    data: function(){
      return {
        num: 0
      }
    },
    template: `
      <div>
        <div>JERRY:{{num}}</div>
        <div>
          <button @click='handle'>点击</button>
        </div>
      </div>
    `,
    methods: {
      handle: function(){
        //2、传递数据方，通过一个事件触发hub.$emit(方法名，传递的数据)    触发兄弟组件的事件
        hub.$emit('tom-event', 1);
      }
    },
    mounted: function() {
      // 3、接收数据方，通过mounted(){} 钩子中   触发hub.$on()方法名
      hub.$on('jerry-event', (val) => {
        this.num += val;
      });
    }
  });
  var vm = new Vue({
    el: '#app',
    data: {

    },
    methods: {
      handle: function(){
        //4、销毁事件 通过hub.$off()方法名销毁之后无法进行传递数据
        hub.$off('tom-event');
        hub.$off('jerry-event');
      }
    }
```

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

组件插槽

- 组件的最大特性就是复用性，而用好插槽能大大提高组件的可复用能力

匿名插槽

```
<div id="app">
  <!-- 这里的所有组件标签中嵌套的内容会替换掉slot  如果不传值 则使用 slot 中的默认值  -->
  <alert-box>有bug发生</alert-box>
  <alert-box>有一个警告</alert-box>
  <alert-box></alert-box>
</div>

<script type="text/javascript">
  /*
    组件插槽：父组件向子组件传递内容
  */
  Vue.component('alert-box', {
    template: `
      <div>
        <strong>ERROR:</strong>
      # 当组件渲染的时候，这个 <slot> 元素将会被替换为“组件标签中嵌套的内容”。
      # 插槽内可以包含任何模板代码，包括 HTML
        <slot>默认内容</slot>
      </div>
    `
  });
  var vm = new Vue({
    el: '#app',
    data: {

    }
  });
</script>
</body>
</html>
```

具名插槽

- 具有名字的插槽
- 使用 中的 "name" 属性绑定元素

```
<div id="app">
  <base-layout>
     <!-- 2、 通过slot属性来指定，这个slot的值必须和下面slot组件得name值对应上
```

```
                如果没有匹配到 则放到匿名的插槽中    -->
      <p slot='header'>标题信息</p>
      <p>主要内容1</p>
      <p>主要内容2</p>
      <p slot='footer'>底部信息信息</p>
    </base-layout>

    <base-layout>
      <!-- 注意点：template临时的包裹标签最终不会渲染到页面上      -->
      <template slot='header'>
        <p>标题信息1</p>
        <p>标题信息2</p>
      </template>
      <p>主要内容1</p>
      <p>主要内容2</p>
      <template slot='footer'>
        <p>底部信息信息1</p>
        <p>底部信息信息2</p>
      </template>
    </base-layout>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
    /*
      具名插槽
    */
    Vue.component('base-layout', {
      template: `
        <div>
          <header>
            ### 1、 使用 <slot> 中的 "name" 属性绑定元素 指定当前插槽的名字
            <slot name='header'></slot>
          </header>
          <main>
            <slot></slot>
          </main>
          <footer>
            ###  注意点:
            ###  具名插槽的渲染顺序，完全取决于模板，而不是取决于父组件中元素的顺序
            <slot name='footer'></slot>
          </footer>
        </div>
      `
    });
    var vm = new Vue({
      el: '#app',
      data: {

      }
    });
  </script>
</body>
</html>
```

作用域插槽

- 父组件对子组件加工处理
- 既可以复用子组件的slot，又可以使slot内容不一致

```
<div id="app">
  <!--
      1、当我们希望li 的样式由外部使用组件的地方定义，因为可能有多种地方要使用该组件，
      但样式希望不一样 这个时候我们需要使用作用域插槽

  -->
  <fruit-list :list='list'>
     <!-- 2、 父组件中使用了<template>元素,而且包含scope="slotProps",
          slotProps在这里只是临时变量
      --->
    <template slot-scope='slotProps'>
      <strong v-if='slotProps.info.id==3' class="current">
          {{slotProps.info.name}}
       </strong>
      <span v-else>{{slotProps.info.name}}</span>
    </template>
  </fruit-list>
</div>
<script type="text/javascript" src="js/vue.js"></script>
<script type="text/javascript">
  /*
    作用域插槽
  */
  Vue.component('fruit-list', {
    props: ['list'],
    template: `
      <div>
        <li :key='item.id' v-for='item in list'>
          ###  3、 在子组件模板中,<slot>元素上有一个类似props传递数据给组件的写法msg="xxx",
          ###    插槽可以提供一个默认内容，如果如果父组件没有为这个插槽提供了内容，会显示默认的内容。
                    如果父组件为这个插槽提供了内容，则默认的内容会被替换掉
          <slot :info='item'>{{item.name}}</slot>
        </li>
      </div>
    `
  });
  var vm = new Vue({
    el: '#app',
    data: {
      list: [{
        id: 1,
        name: 'apple'
      },{
        id: 2,
        name: 'orange'
      },{
```

```
          id: 3,
          name: 'banana'
        }]
      }
    });
  </script>
</body>
</html>
```

购物车案例

1. 实现组件化布局

- 把静态页面转换成组件化模式
- 把组件渲染到页面上

```
<div id="app">
   <div class="container">
     <!-- 2、把组件渲染到页面上 -->
     <my-cart></my-cart>
   </div>
 </div>
 <script type="text/javascript" src="js/vue.js"></script>
 <script type="text/javascript">
   # 1、 把静态页面转换成组件化模式
   # 1.1  标题组件
   var CartTitle = {
     template: `
       <div class="title">我的商品</div>
     `
   }
   # 1.2  商品列表组件
   var CartList = {
     #  注意点 :   组件模板必须是单个根元素
     template: `
       <div>
         <div class="item">
           <img src="img/a.jpg"/>
           <div class="name"></div>
           <div class="change">
             <a href=""> - </a>
             <input type="text" class="num" />
             <a href=""> + </a>
           </div>
           <div class="del">x</div>
         </div>
         <div class="item">
           <img src="img/b.jpg"/>
           <div class="name"></div>
           <div class="change">
```

```
            <a href="">-</a>
            <input type="text" class="num" />
            <a href="">+</a>
          </div>
          <div class="del">x</div>
        </div>
        <div class="item">
          <img src="img/c.jpg"/>
          <div class="name"></div>
          <div class="change">
            <a href="">-</a>
            <input type="text" class="num" />
            <a href="">+</a>
          </div>
          <div class="del">x</div>
        </div>
        <div class="item">
          <img src="img/d.jpg"/>
          <div class="name"></div>
          <div class="change">
            <a href="">-</a>
            <input type="text" class="num" />
            <a href="">+</a>
          </div>
          <div class="del">x</div>
        </div>
        <div class="item">
          <img src="img/e.jpg"/>
          <div class="name"></div>
          <div class="change">
            <a href="">-</a>
            <input type="text" class="num" />
            <a href="">+</a>
          </div>
          <div class="del">x</div>
        </div>
      </div>
    `
}
# 1.3  商品结算组件
var CartTotal = {
  template: `
    <div class="total">
      <span>总价: 123</span>
      <button>结算</button>
    </div>
  `
}
## 1.4  定义一个全局组件 my-cart
Vue.component('my-cart',{
  ##  1.6 引入子组件
  template: `
    <div class='cart'>
```

```
          <cart-title></cart-title>
          <cart-list></cart-list>
          <cart-total></cart-total>
        </div>
      `,
      # 1.5  注册子组件
      components: {
        'cart-title': CartTitle,
        'cart-list': CartList,
        'cart-total': CartTotal
      }
    });
    var vm = new Vue({
      el: '#app',
      data: {

      }
    });

  </script>
```

2、 实现 标题和结算功能组件

- 标题组件实现动态渲染
  - 从父组件把标题数据传递过来 即 父向子组件传值
  - 把传递过来的数据渲染到页面上
- 结算功能组件
  - 从父组件把商品列表list 数据传递过来 即 父向子组件传值
  - 把传递过来的数据计算最终价格渲染到页面上

```
<div id="app">
    <div class="container">
      <my-cart></my-cart>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">
     # 2.2  标题组件      子组件通过props形式接收父组件传递过来的uname数据
    var CartTitle = {
      props: ['uname'],
      template: `
        <div class="title">{{uname}}的商品</div>
      `
    }
    # 2.3  商品结算组件  子组件通过props形式接收父组件传递过来的list数据
    var CartTotal = {
      props: ['list'],
      template: `
        <div class="total">
```

```
      <span>总价: {{total}}</span>
      <button>结算</button>
    </div>
  `,
  computed: {
    # 2.4    计算商品的总价   并渲染到页面上
    total: function() {
      var t = 0;
      this.list.forEach(item => {
        t += item.price * item.num;
      });
      return t;
    }
  }
}
Vue.component('my-cart',{
  data: function() {
    return {
      uname: '张三',
      list: [{
        id: 1,
        name: 'TCL彩电',
        price: 1000,
        num: 1,
        img: 'img/a.jpg'
      },{
        id: 2,
        name: '机顶盒',
        price: 1000,
        num: 1,
        img: 'img/b.jpg'
      },{
        id: 3,
        name: '海尔冰箱',
        price: 1000,
        num: 1,
        img: 'img/c.jpg'
      },{
        id: 4,
        name: '小米手机',
        price: 1000,
        num: 1,
        img: 'img/d.jpg'
      },{
        id: 5,
        name: 'PPTV电视',
        price: 1000,
        num: 2,
        img: 'img/e.jpg'
      }]
    }
  },
  #  2.1  父组件向子组件以属性传递的形式 传递数据
```

```
    #   向 标题组件传递 uname 属性    向 商品结算组件传递 list  属性
    template: `
      <div class='cart'>
        <cart-title :uname='uname'></cart-title>
        <cart-list></cart-list>
        <cart-total :list='list'></cart-total>
      </div>
    `,
    components: {
      'cart-title': CartTitle,
      'cart-list': CartList,
      'cart-total': CartTotal
    }
  });
  var vm = new Vue({
    el: '#app',
    data: {

    }
  });

</script>
```

3. 实现列表组件删除功能

- 从父组件把商品列表list 数据传递过来 即 父向子组件传值
- 把传递过来的数据渲染到页面上
- 点击删除按钮的时候删除对应的数据
  - 给按钮添加点击事件把需要删除的id传递过来
    - 子组件中不推荐操作父组件的数据有可能多个子组件使用父组件的数据 我们需要把数据传递给父组件让父组件操作数据
    - 父组件删除对应的数据

```
<div id="app">
  <div class="container">
    <my-cart></my-cart>
  </div>
</div>
<script type="text/javascript" src="js/vue.js"></script>
<script type="text/javascript">

  var CartTitle = {
    props: ['uname'],
    template: `
      <div class="title">{{uname}}的商品</div>
    `
  }
  #  3.2 把列表数据动态渲染到页面上
  var CartList = {
    props: ['list'],
```

```
  template: `
    <div>
      <div :key='item.id' v-for='item in list' class="item">
        <img :src="item.img"/>
        <div class="name">{{item.name}}</div>
        <div class="change">
          <a href="">-</a>
          <input type="text" class="num" />
          <a href="">+</a>
        </div>
        # 3.3  给按钮添加点击事件把需要删除的id传递过来
        <div class="del" @click='del(item.id)'>x</div>
      </div>
    </div>
  `,
  methods: {
    del: function(id){
       # 3.4 子组件中不推荐操作父组件的数据有可能多个子组件使用父组件的数据
      #       我们需要把数据传递给父组件 让父组件操作数据
      this.$emit('cart-del', id);
    }
  }
}
var CartTotal = {
  props: ['list'],
  template: `
    <div class="total">
      <span>总价: {{total}}</span>
      <button>结算</button>
    </div>
  `,
  computed: {
    total: function() {
      // 计算商品的总价
      var t = 0;
      this.list.forEach(item => {
        t += item.price * item.num;
      });
      return t;
    }
  }
}
Vue.component('my-cart',{
  data: function() {
    return {
      uname: '张三',
      list: [{
        id: 1,
        name: 'TCL彩电',
        price: 1000,
        num: 1,
        img: 'img/a.jpg'
      },{
```

```
          id: 2,
          name: '机顶盒',
          price: 1000,
          num: 1,
          img: 'img/b.jpg'
        },{
          id: 3,
          name: '海尔冰箱',
          price: 1000,
          num: 1,
          img: 'img/c.jpg'
        },{
          id: 4,
          name: '小米手机',
          price: 1000,
          num: 1,
          img: 'img/d.jpg'
        },{
          id: 5,
          name: 'PPTV电视',
          price: 1000,
          num: 2,
          img: 'img/e.jpg'
        }]
      }
    },
    # 3.1 从父组件把商品列表list 数据传递过来 即 父向子组件传值
    template: `
      <div class='cart'>
        <cart-title :uname='uname'></cart-title>
        #  3.5  父组件通过事件绑定 接收子组件传递过来的数据
        <cart-list :list='list' @cart-del='delCart($event)'></cart-list>
        <cart-total :list='list'></cart-total>
      </div>
    `,
    components: {
      'cart-title': CartTitle,
      'cart-list': CartList,
      'cart-total': CartTotal
    },
    methods: {
      # 3.6    根据id删除list中对应的数据
      delCart: function(id) {
        // 1、找到id所对应数据的索引
        var index = this.list.findIndex(item=>{
          return item.id == id;
        });
        // 2、根据索引删除对应数据
        this.list.splice(index, 1);
      }
    }
  });
  var vm = new Vue({
```

```
      el: '#app',
      data: {

      }
    });

  </script>
</body>
</html>
```

4. 实现组件更新数据功能 上

- 1. 将输入框中的默认数据动态渲染出来
- 2. 输入框失去焦点的时候 更改商品的数量
- 3 子组件中不推荐操作数据 把这些数据传递给父组件 让父组件处理这些数据
- 4 父组件中接收子组件传递过来的数据并处理

```
 <div id="app">
    <div class="container">
      <my-cart></my-cart>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">

    var CartTitle = {
      props: ['uname'],
      template: `
        <div class="title">{{uname}}的商品</div>
      `
    }
    var CartList = {
      props: ['list'],
      template: `
        <div>
          <div :key='item.id' v-for='item in list' class="item">
            <img :src="item.img"/>
            <div class="name">{{item.name}}</div>
            <div class="change">
              <a href="">-</a>
                # 1. 将输入框中的默认数据动态渲染出来
                # 2. 输入框失去焦点的时候 更改商品的数量  需要将当前商品的id 传递过来
              <input type="text" class="num" :value='item.num' @blur='changeNum(item.id,
$event)'/>
              <a href="">+</a>
            </div>
            <div class="del" @click='del(item.id)'>x</div>
          </div>
        </div>
      `,
      methods: {
        changeNum: function(id, event){
```

```
          # 3 子组件中不推荐操作数据  因为别的组件可能也引用了这些数据
          #  把这些数据传递给父组件 让父组件处理这些数据
          this.$emit('change-num', {
            id: id,
            num: event.target.value
          });
        },
        del: function(id){
          // 把id传递给父组件
          this.$emit('cart-del', id);
        }
      }
    }
    var CartTotal = {
      props: ['list'],
      template: `
        <div class="total">
          <span>总价: {{total}}</span>
          <button>结算</button>
        </div>
      `,
      computed: {
        total: function() {
          // 计算商品的总价
          var t = 0;
          this.list.forEach(item => {
            t += item.price * item.num;
          });
          return t;
        }
      }
    }
    Vue.component('my-cart',{
      data: function() {
        return {
          uname: '张三',
          list: [{
            id: 1,
            name: 'TCL彩电',
            price: 1000,
            num: 1,
            img: 'img/a.jpg'
          }]
      },
      template: `
        <div class='cart'>
          <cart-title :uname='uname'></cart-title>
            # 4  父组件中接收子组件传递过来的数据
          <cart-list :list='list' @change-num='changeNum($event)' @cart-
del='delCart($event)'></cart-list>
          <cart-total :list='list'></cart-total>
        </div>
      `,
```

```
    components: {
      'cart-title': CartTitle,
      'cart-list': CartList,
      'cart-total': CartTotal
    },
    methods: {
      changeNum: function(val) {
        //4.1 根据子组件传递过来的数据，跟新list中对应的数据
        this.list.some(item=>{
          if(item.id == val.id) {
            item.num = val.num;
            // 终止遍历
            return true;
          }
        });
      },
      delCart: function(id) {
        // 根据id删除list中对应的数据
        // 1、找到id所对应数据的索引
        var index = this.list.findIndex(item=>{
          return item.id == id;
        });
        // 2、根据索引删除对应数据
        this.list.splice(index, 1);
      }
    }
  });
  var vm = new Vue({
    el: '#app',
    data: {

    }
  });

</script>
```

5. 实现组件更新数据功能 下

- 1

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <style type="text/css">
    .container {
    }
    .container .cart {
      width: 300px;
      margin: auto;
    }
    .container .title {
```

```
  background-color: lightblue;
  height: 40px;
  line-height: 40px;
  text-align: center;
  /*color: #fff;*/
}
.container .total {
  background-color: #FFCE46;
  height: 50px;
  line-height: 50px;
  text-align: right;
}
.container .total button {
  margin: 0 10px;
  background-color: #DC4C40;
  height: 35px;
  width: 80px;
  border: 0;
}
.container .total span {
  color: red;
  font-weight: bold;
}
.container .item {
  height: 55px;
  line-height: 55px;
  position: relative;
  border-top: 1px solid #ADD8E6;
}
.container .item img {
  width: 45px;
  height: 45px;
  margin: 5px;
}
.container .item .name {
  position: absolute;
  width: 90px;
  top: 0;left: 55px;
  font-size: 16px;
}

.container .item .change {
  width: 100px;
  position: absolute;
  top: 0;
  right: 50px;
}
.container .item .change a {
  font-size: 20px;
  width: 30px;
  text-decoration:none;
  background-color: lightgray;
  vertical-align: middle;
```

```
    }
    .container .item .change .num {
      width: 40px;
      height: 25px;
    }
    .container .item .del {
      position: absolute;
      top: 0;
      right: 0px;
      width: 40px;
      text-align: center;
      font-size: 40px;
      cursor: pointer;
      color: red;
    }
    .container .item .del:hover {
      background-color: orange;
    }
  </style>
</head>
<body>
  <div id="app">
    <div class="container">
      <my-cart></my-cart>
    </div>
  </div>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript">

    var CartTitle = {
      props: ['uname'],
      template: `
        <div class="title">{{uname}}的商品</div>
      `
    }
    var CartList = {
      props: ['list'],
      template: `
        <div>
          <div :key='item.id' v-for='item in list' class="item">
            <img :src="item.img"/>
            <div class="name">{{item.name}}</div>
            <div class="change">
              # 1.  + - 按钮绑定事件
              <a href="" @click.prevent='sub(item.id)'> - </a>
              <input type="text" class="num" :value='item.num' @blur='changeNum(item.id,
$event)'/>
              <a href="" @click.prevent='add(item.id)'> + </a>
            </div>
            <div class="del" @click='del(item.id)'>x</div>
          </div>
        </div>
      `,
```

```
      methods: {
        changeNum: function(id, event){
          this.$emit('change-num', {
            id: id,
            type: 'change',
            num: event.target.value
          });
        },
        sub: function(id){
          # 2 数量的增加和减少通过父组件来计算    每次都是加1 和 减1 不需要传递数量    父组件需要一个类型来判断 是 加一 还是减1  以及是输入框输入的数据  我们通过type 标识符来标记 不同的操作
          this.$emit('change-num', {
            id: id,
            type: 'sub'
          });
        },
        add: function(id){
          # 2 数量的增加和减少通过父组件来计算    每次都是加1 和 减1 不需要传递数量    父组件需要一个类型来判断 是 加一 还是减1  以及是输入框输入的数据  我们通过type 标识符来标记 不同的操作
          this.$emit('change-num', {
            id: id,
            type: 'add'
          });
        },
        del: function(id){
          // 把id传递给父组件
          this.$emit('cart-del', id);
        }
      }
    }
    var CartTotal = {
      props: ['list'],
      template: `
        <div class="total">
          <span>总价: {{total}}</span>
          <button>结算</button>
        </div>
      `,
      computed: {
        total: function() {
          // 计算商品的总价
          var t = 0;
          this.list.forEach(item => {
            t += item.price * item.num;
          });
          return t;
        }
      }
    }
    Vue.component('my-cart',{
      data: function() {
        return {
          uname: '张三',
```

```
          list: [{
            id: 1,
            name: 'TCL彩电',
            price: 1000,
            num: 1,
            img: 'img/a.jpg'
          },{
            id: 2,
            name: '机顶盒',
            price: 1000,
            num: 1,
            img: 'img/b.jpg'
          },{
            id: 3,
            name: '海尔冰箱',
            price: 1000,
            num: 1,
            img: 'img/c.jpg'
          },{
            id: 4,
            name: '小米手机',
            price: 1000,
            num: 1,
            img: 'img/d.jpg'
          },{
            id: 5,
            name: 'PPTV电视',
            price: 1000,
            num: 2,
            img: 'img/e.jpg'
          }]
        }
      },
      template: `
        <div class='cart'>
          <cart-title :uname='uname'></cart-title>
        # 3 父组件通过事件监听    接收子组件的数据
          <cart-list :list='list' @change-num='changeNum($event)' @cart-
del='delCart($event)'></cart-list>
          <cart-total :list='list'></cart-total>
        </div>
      `,
      components: {
        'cart-title': CartTitle,
        'cart-list': CartList,
        'cart-total': CartTotal
      },
      methods: {
        changeNum: function(val) {
          #4 分为三种情况：输入框变更、加号变更、减号变更
          if(val.type=='change') {
            // 根据子组件传递过来的数据，跟新list中对应的数据
            this.list.some(item=>{
```

```
              if(item.id == val.id) {
                item.num = val.num;
                // 终止遍历
                return true;
              }
            });
          }else if(val.type=='sub'){
            // 减一操作
            this.list.some(item=>{
              if(item.id == val.id) {
                item.num -= 1;
                // 终止遍历
                return true;
              }
            });
          }else if(val.type=='add'){
            // 加一操作
            this.list.some(item=>{
              if(item.id == val.id) {
                item.num += 1;
                // 终止遍历
                return true;
              }
            });
          }
        }
      }
    });
    var vm = new Vue({
      el: '#app',
      data: {

      }
    });

  </script>
</body>
</html>
```