Vue.js 组件开发

使用组件扩展 HTML 元素，封装可重用的代码

Content

- 组件 Component
- 注册全局，局部组件
- 组件命名约定
- is 属性
- 组件 data 函数
- 父子组件数据传递
- prop 验证
- 自定义事件：$on, $emit
- 原生事件监听 .native

Content

- 非父子组件通信
- Slot 分发内容
- 单个 Slot，匿名 Slot，具名 Slot
- 作用域插槽
- 动态组件
- keep-alive
- 可复用组件
- 子组件索引 $refs
- 异步组件 resolve reject
- v-once

组件 Component

- **组件（Component）**是 Vue.js 最强大的功能之一。
- 组件可以扩展 HTML 元素，封装可重用的代码。
- 在较高层面上，组件是自定义元素， Vue.js 的编译器为它添加特殊功能。
- 在有些情况下，组件也可以是原生 HTML 元素的形式，以 is 特性扩展。

注册全局组件

- 注册全局组件
  - Vue.component(tagName, options)
  - 全局组件可以在任意 Vue 实例下使用
  - 组件在注册之后，以自定义元素 <tagName></ tagName > 的形式使用

```
// 注册
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
// 创建根实例
new Vue({
  el: '#example'
})
```

```
// 使用
<div id="example">
  <my-component></my-component>
</div>
```

注册局部组件

- 注册局部组件
  - 仅在另一个实例/组件的作用域中可用

```
var Child = {
    template: '<div>A custom component!</div>'
}
new Vue({
    // ...
    el: '#example',
    components: {
        // <my-component> 将只在父模板可用
        'my-component': Child
    }
})
```

组件命名约定

- 当注册组件（或者 props）时，可以使用 kebab-case ，camelCase ，或 TitleCase 。
- 在 HTML 模版中，必须使用 kebab-case 形式。

Vue 实例 DOM 模板

- 如果使用 DOM 模板，Vue 只有在浏览器解析和标准化 HTML 后才能获取模版内容，某些受限制被包裹内容的元素（<ul>，<ol>，<table>，<select>）在自定义组件中使用时会导致一些自定义组件被认为是无效的内容

```
<select>
    <app-layout></app-layout> <!-- 模板返回的 option 无法生效 -->
</select>
```

is 属性

- 变通的方案是使用原生 HTML，然后使用特殊的 is 属性：

```
<select>
    <app-layout></app-layout> <!-- 无效 -->
    <option is="app-layout"></option> <!-- 有效，使用 this 属性 -->
</select>
```

组件 data 函数

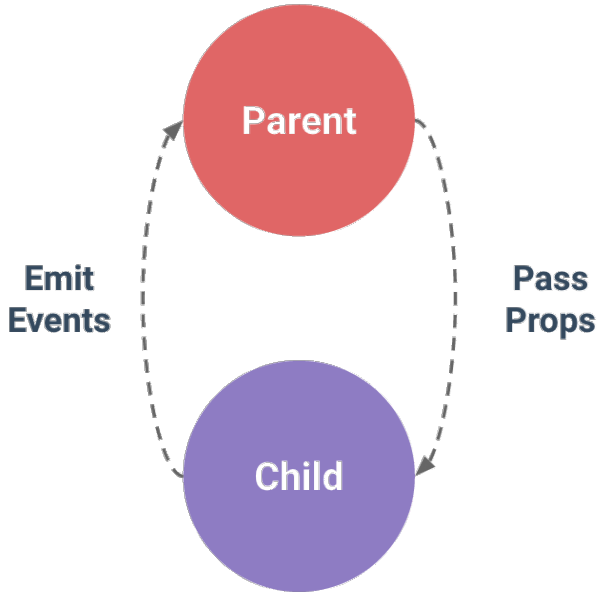
- data 属性要求声明为函数，以便防止多个 Vue 组件共享数据导致的修改影响

```
Vue.component('my-component', {
    template: '<button @click="counter += 1">{{counter}}</button>',
    data: function () {
      return {
        counter: 0
      }
    }
})
```

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

父子组件数据传递

- 在 Vue.js 中，父子组件的关系可以总结为 **props down, events up** 。父组件通过 props 向下传递数据给子组件，子组件通过 events 给父组件发送消息。



使用 Prop 传递数据

- 父子组件之间，可以使用 props 把数据传给子组件。
  - prop 的名字形式会从 camelCase 转为 kebab-case（短横线隔开）

```
Vue.component('child', {
    // 声明 props
    props: ['myMessage'],
    template: '<span>{{ myMessage }}</span>'
})

<!-- 直接为组件传递字符串 -->
<child my-message="hello!"></child>
<!-- 绑定父组件的属性，向子组件传递数据 -->
<child :my-message="vmMsg"></child>
```

Prop 验证

- 组件可以为 props 指定验证要求。当 prop 验证失败了，如果使用的是开发版本会抛出一条警告。
- prop 是一个对象而不是字符串数组时，它包含属性名和对应的验证要求。

Prop 验证

```
props: {
  // 基础类型检测 （`null` 意思是任何类型都可以）
  propA: Number,
  // 多种类型
  propB: [String, Number],
  // 必传且是字符串
  propC: {
    type: String,  required: true
  },
  // 数字，有默认值
  propD: {
    type: Number, default: 100
  },
  // 数组 / 对象的默认值应当由一个工厂函数返回
  propE: {
    type: Object,
    default: function () {
      return { message: 'hello' }
    }
  },
  // 自定义验证函数
  propF: {
    validator: function (value) {
      return value > 10
    }
  }
}
```

Prop 验证

- type 可以是下面原生构造器：
  - String
  - Number
  - Boolean
  - Function
  - Object
  - Array
- type 也可以是一个自定义构造器，使用 instanceof 检测。

自定义事件

- props 用于传递数据给子组件；自定义事件可从子组件要把数据传递回父组件。
- 事件接口
  - **$on(eventName, callback)** 监听事件
  - **$emit(eventName, […args])** 触发事件
- **v-on**

父组件可以在使用子组件的地方直接监听子组件触发的事件

自定义事件

- 使用 v-on 监听子组件通过 $emit 引发的事件

```
<div id="counter-event-example">
    <p>{{ total }}</p>
    <!-- 使用 v-on 监听子组件触发的事件，调用父组件方法 -->
    <button-counter v-on:increment="incrementTotal"></button-counter>
    <button-counter @increment="incrementTotal"></button-counter>
</div>
```

自定义事件

- 引发事件

```
Vue.component('button-counter', {
    template: '<button @click="increment">{{ counter }}</button>',
    data: function() { return { counter: 0 }},
    methods: {
        increment: function() {
            this.counter += 1;
            this.$emit('increment'); // 触发父组件关心的 increment 事件
        }
    }
})
var vm=new Vue({
    el: '#counter-event-example',
    data: { total: 0 },
    methods: {
        incrementTotal: function() {
            this.total += 1
        }
    }
})
```

原生事件监听

- 使用 **.native** 修饰 v-on

```
<button-counter  v-on:click.native="console.info(111);"></button-counter>
```

非父子组件通信

- 使用一个空的 Vue 实例作为中央事件总线

```
var bus = new Vue()

// 在组件 A 中监听事件
bus.$on('show', function(id, name, sex) {
    console.info(id);
    console.info(name);
    console.info(sex);
    vm.total=100;
})

// 在 B 中触发组件 A 中的事件
bus.$emit('show', 1, 'Jay', 'F')
```

Slot 分发内容

- Slot 槽位可以用来动态分发内容
- **单个 Slot**
  - 匿名Slot
- **具名 Slot**
  - 包含 name 名字属性，匹配内容片段中有对应 slot 特性的元素
  - 可以有一个匿名 slot，它是默认 slot，作为找不到匹配的内容片段的备用插槽。如果没有默认的 slot，这些找不到匹配的内容片段将被抛弃。

单个 Slot

- 单个 Slot

```
<script type="text/x-template"
id="headerComponentTmpl">
    <div>
        <h2>子组件</h2>
        <slot> 没有分发内容时显示 </slot>
    </div>
</script>

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

```
<script type="text/javascript">
    Vue.component("header-component", {
        template: "#headerComponentTmpl"
    })

    new Vue({
        el: "#app"
    });
</script>
```

单个 Slot

- 在子组件标签内可以指定单个 Solt 的分发内容

```
<div id="app">
    <h1>父组件</h1>
    <header-component>
        分发内容，父组件数据 {{message}}
    </header-component>
</div>
```

具名 Slot

- 可以有一个默认 slot

```
<script type="text/x-template"
id="appLayoutTmpl">
    <div class="container">
        <header>
            <slot name="header"></slot>
        </header>
        <main>
            <slot>默认 Slot</slot>
        </main>
        <footer>
            <slot name="footer"></slot>
        </footer>
    </div>
</script>
```

```
<div id="app">
    <h1>父组件</h1>
    <app-layout>
        <h1 slot="header"> Header 标题 </h1>
        <p>默认 slot 接收内容。</p>
        <div slot="footer">
          <a href="#">联系我们</a>
        </div>
    </app-layout>
</div>
```

作用域插槽

- props 可以将子组件在插槽上的数据传递到父组件使用

```
<script type="text/x-template" id="appLayoutTmpl">
    <div>
        <slot text="信息1" text2="信息2"></slot>
    </div>
</script>
<div id="app">
    <h1>父组件</h1>
    <app-layout>
        <template scope="props">
          <h1>子组件标题</h1>
          <p>{{ props.text }}</p>
          <p>{{ props.text2 }}</p>
         </template>
    </app-layout>
</div>
```

动态组件

- 多个组件可以使用同一个挂载点，然后动态地在它们之间切换。
- 使用保留的 <component> 元素，动态地绑定到它的 is 特性：

```
new Vue({
    el: "#app",
    data: {
        message: "parent Message",
        currentComponent: 'myh1'
    },
    components:{
        "myh1":{
            template: "#h1Tmpl"
        },
        "myh2":{
            template: "#h2Tmpl"
        }
    }
});
```

```
<script type="text/x-template" id="h1Tmpl">
    <h1><slot></slot></h1>
</script>
<script type="text/x-template" id="h2Tmpl">
    <h2><slot></slot></h2>
</script>
<div id="app">
    <h1>父组件</h1>
    <div v-bind:is="currentComponent"> 动态组件
    {{currentComponent}}
    </div >
</div>
```

Keep-alive

- 如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个 keep-alive 指令参数：

```
<div id="app">
    <h1>父组件</h1>
    <keep-alive>
        <div v-bind:is="currentComponent">
        动态组件 {{currentComponent}}
        </div>
    </keep-alive>
</div>
```

可复用组件

- Vue 组件的 API 来自三部分 - props, events 和 slots：
  - **Props** 允许外部环境传递数据给组件
  - **Events** 允许组件触发外部环境的调用
  - **Slots** 允许外部环境将额外的内容组合在组件中

子组件索引

- 尽管有 props 和 events，但是有时仍然需要在 JavaScript 中直接访问子组件。为此可以使用 ref 为子组件指定一个索引 ID 。
  - $refs 只在组件渲染完成后才填充，仅仅作为一个直接访问子组件的应急方案——应当避免在模版或计算属性中使用 $refs 。

```
<div id="parent">
  <user-profile ref="profile"></user-profile>
</div>
```

```
var parent = new Vue({ el: '#parent' })
// 访问子组件
var child = parent.$refs.profile
```

异步组件

- 在大型应用中，可能需要将应用拆分为多个小模块，按需从服务器下载。
- 为了让事情更简单， Vue.js 允许将组件定义为一个工厂函数，动态地解析组件的定义。Vue.js 只在组件需要渲染时触发工厂函数，并且把结果缓存起来，用于后面的再次渲染。

异步组件

- 异步组件工厂函数
  - resolve 回调，在收到从服务器下载的组件定义时调用
  - reject(reason) 指示加载失败

```
Vue.component('app-layout', function (resolve, reject) {
  setTimeout(function () {
    resolve({
      template: '<div>I am async!</div>'
    })
  }, 2000)
})
```

v-once

- 当组件中包含大量静态内容时，可以考虑使用 **v-once** 将渲染结果缓存起来

```
Vue.component('app-layout', {
  template: '\
    <div v-once>\
      <h1>Terms of Service</h1>\
      ... a lot of static content ...\
    </div>\
  '
})
```

End

Thanks!

任务

练习

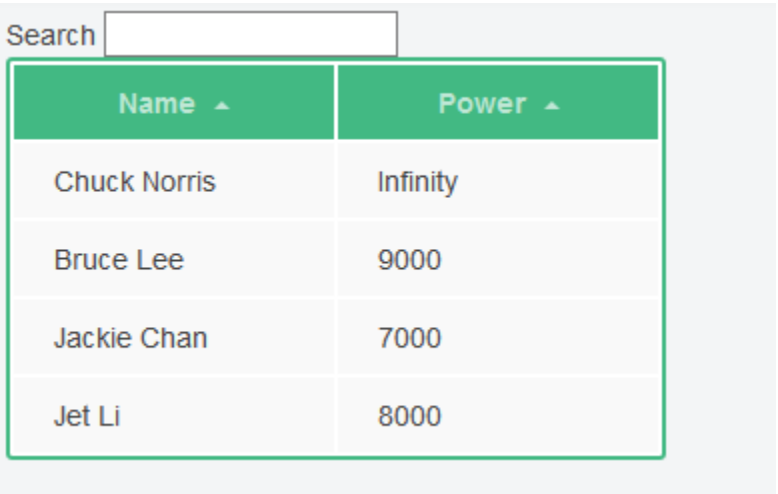
任务一

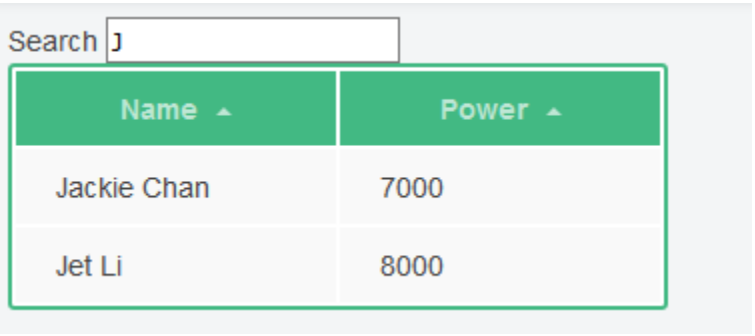
- 使用自定义组件实现如下备忘录

- 提交后文本框清空
- 点击删除，删除对应事件

任务二

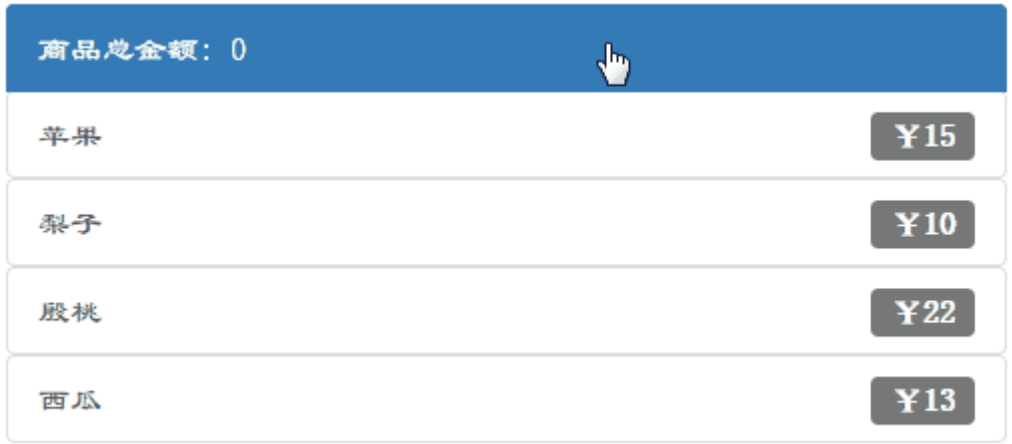
- 网格即时搜索，输入字符，立即显示相关网格数据





任务三

- 商品金额统计，点击选择，总额自动更新

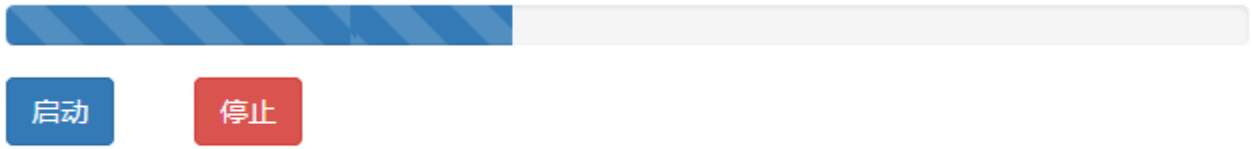


任务四

- 父子组件数据传递，更新父组件或子组件时联动更新

任务五

- 使用 Vue 制作一个可控制的进度条



任务六

- 制作如下简易留言板