
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
Vue全家桶之组件化开发
GO

目录 Contents

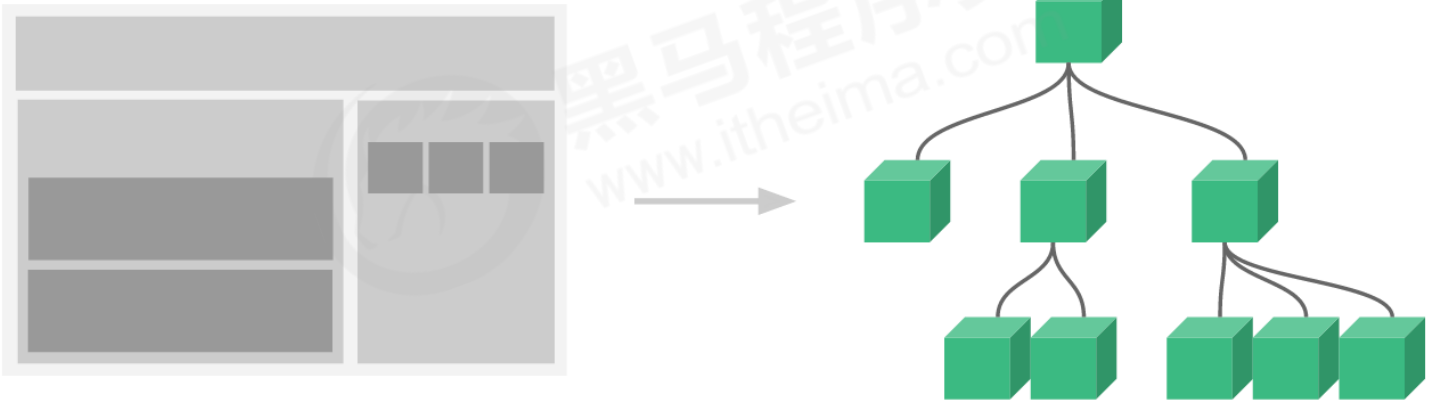
1. 组件化开发思想

1.1 现实中的组件化思想体现

- 标准
- 分治
- 重用
- 组合

1. 组件化开发思想

1.2 编程中的组件化思想体现



1.3 组件化规范: Web Components

- 我们希望尽可能多的重用代码
- 自定义组件的方式不太容易（html、css和js）
- 多次使用组件可能导致冲突

Web Components 通过创建封装好功能的**定制元素**解决上述问题

官网：https://developer.mozilla.org/zh-CN/docs/Web/Web_Components

Vue部分实现了上述规范

目录 Contents

2. 组件注册

2.1 全局组件注册语法

```
Vue.component(组件名称, {
  data: 组件数据,
  template: 组件模板内容
})
```

```
// 注册一个名为 button-counter 的新组件
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">点击了{{ count }}次.</button>'
})
```

2. 组件注册

2.2 组件用法

```
<div id="app">
    <button-counter></button-counter>
</div>
```

```
<div id="app">
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
</div>
```

2.3 组件注册注意事项

1. data必须是一个函数

- 分析函数与普通对象的对比

2. 组件模板内容必须是单个跟元素

- 分析演示实际的效果

3. 组件模板内容可以是模板字符串

- 模板字符串需要浏览器提供支持（ES6语法）

2.3 组件注册注意事项

4. 组件命名方式

- 短横线方式

```
Vue.component('my-component', { /* ... */ })
```

- 驼峰方式

```
Vue.component('MyComponent', { /* ... */ })
```

2.4 局部组件注册

```
var ComponentA = { /* ... */ }
var ComponentB = { /* ... */ }
var ComponentC = { /* ... */ }
new Vue({
  el: '#app'
  components: {
    'component-a': ComponentA,
    'component-b': ComponentB,
    'component-c': ComponentC,
  }
})
```

目录 Contents

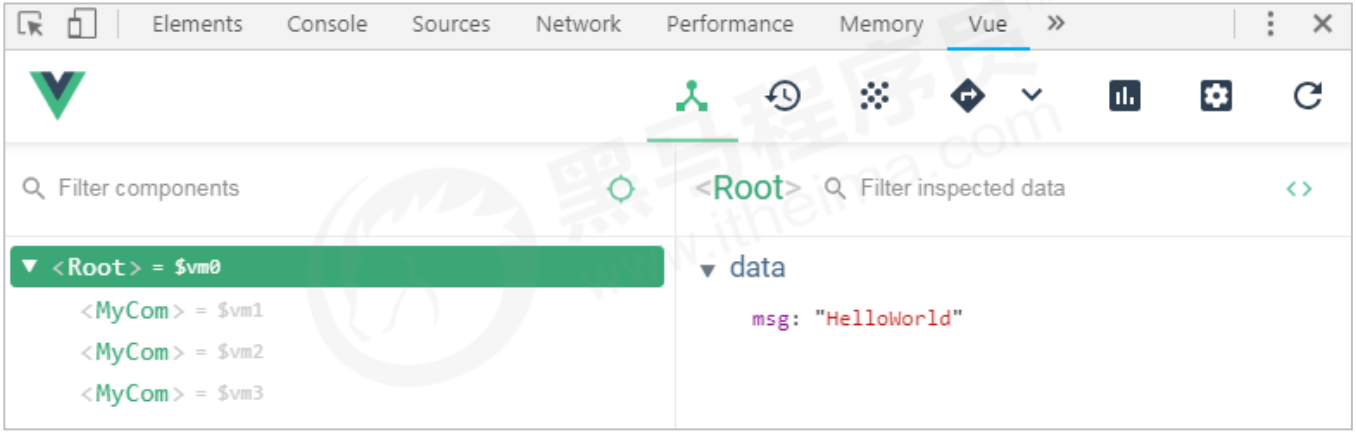
3. Vue调试工具

3.1 调试工具安装

① 克隆仓库
② 安装依赖包
③ 构建
④ 打开Chrome扩展页面
⑤ 选中开发者模式
⑥ 加载已解压的扩展，选择shells/chrome

3. Vue调试工具

3.2 调试工具用法



目录 Contents

4. 组件间数据交互

4.1 父组件向子组件传值

1. 组件内部通过props接收传递过来的值

```
Vue.component('menu-item', {
  props: ['title'],
  template: '<div>{{ title }}</div>'
})
```

2. 父组件通过属性将值传递给子组件

```
<menu-item title="来自父组件的数据"></menu-item>
<menu-item :title="title"></menu-item>
```

4. 组件间数据交互

4.1 父组件向子组件传值

3. props属性名规则

- 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

```
Vue.component('menu-item', {
    // 在 JavaScript 中是驼峰式的
    props: ['menuTitle'],
    template: '<div>{{ menuTitle }}</div>'
})
<!- 在html中是短横线方式的 -->
<menu-item menu-title="nihao"></menu-item>
```

4. 组件间数据交互

4.1 父组件向子组件传值

4. props属性值类型

- 字符串 String
- 数值 Number
- 布尔值 Boolean
- 数组 Array
- 对象 Object

4. 组件间数据交互

4.2 子组件向父组件传值

1. 子组件通过自定义事件向父组件传递信息

```
<button v-on:click='$emit("enlarge-text") '>扩大字体</button>
```

2. 父组件监听子组件的事件

```
<menu-item v-on:enlarge-text='fontSize += 0.1'></menu-item>
```

4. 组件间数据交互

4.2 子组件向父组件传值

3. 子组件通过自定义事件向父组件传递信息

```
<button v-on:click='$emit("enlarge-text", 0.1) '>扩大字体</button>
```

4. 父组件监听子组件的事件

```
<menu-item v-on:enlarge-text='fontSize += $event'></menu-item>
```

4. 组件间数据交互

4.3 非父子组件间传值

1. 单独的事件中心管理组件间的通信

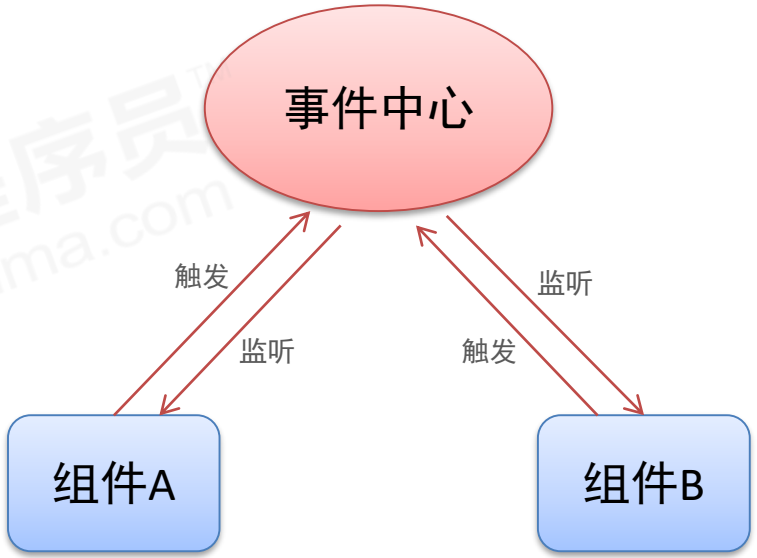
```
var eventHub = new Vue()
```

2. 监听事件与销毁事件

```
eventHub.$on('add-todo', addTodo)
eventHub.$off('add-todo')
```

3. 触发事件

```
eventHub.$emit('add-todo', id)
```

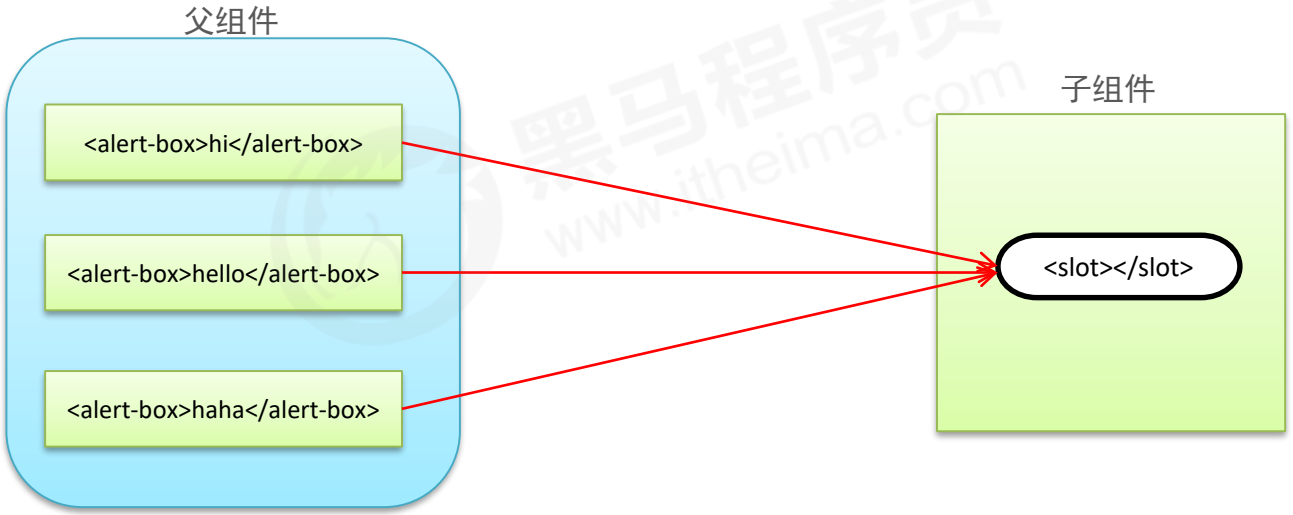


目录 Contents

4. 组件间数据交互

5.1 组件插槽的作用

- 父组件向子组件传递内容



5.2 组件插槽基本用法

1. 插槽位置

```
Vue.component('alert-box', {
  template: `
    <div class="demo-alert-box">
      <strong>Error!</strong>
      <slot></slot>
    </div>
  `
})
```

2. 插槽内容

```
<alert-box>Something bad happened.</alert-box>
```

5.3 具名插槽用法

1. 插槽定义

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

2. 插槽内容

```
<base-layout>
  <h1 slot="header">标题内容</h1>

  <p>主要内容1</p>
  <p>主要内容2</p>

  <p slot="footer">底部内容</p>
</base-layout>
```

5. 组件插槽

5.4 作用域插槽

- 应用场景：**父组件对子组件的内容进行加工处理**

1. 插槽定义

```
<ul>
  <li v-for= "item in list" v-bind:key= "item.id" >
     <slot v-bind:item="item">
       {{item.name}}
     </slot>
  </li>
</ul>
```

2. 插槽内容

```
<fruit-list v-bind:list= "list">
  <template slot-scope="slotProps">
    <strong v-if="slotProps.item.current">
       {{ slotProps.item.text }}
    </strong>
  </template>
</fruit-list>
```

目录 Contents

6. 基于组件的案例

案例：购物车

6. 基于组件的案例

案例：需求分析

1. 按照组件化方式实现业务需求

- 根据业务功能进行组件化划分
  - ① 标题组件（展示文本）
  - ② 列表组件（列表展示、商品数量变更、商品删除）
  - ③ 结算组件（计算商品总额）

6. 基于组件的案例

案例：实现步骤

1. 功能实现步骤

- 实现整体布局和样式效果
- 划分独立的功能组件
- 组合所有的子组件形成整体结构
- 逐个实现各个组件功能
  - 标题组件
  - 列表组件
  - 结算组件

传智播客旗下高端IT教育品牌