

Vue 核心技术与实战

day04



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



目录

Contents

◆ 组件的三大组成部分 (结构/样式/逻辑)

scoped样式冲突 / data是一个函数

◆ 组件通信

组件通信语法 / 父传子 / 子传父 / 非父子 (扩展)

◆ 综合案例：小黑记事本 (组件版)

拆分组件 / 渲染 / 添加 / 删除 / 统计 / 清空 / 持久化

◆ 进阶语法

v-model原理 / v-model应用于组件 / sync修饰符 / ref 和 \$refs / \$nextTick

综合案例：

小黑记事本

添加任务

1. 跑步一公里

2. 游泳100米

合计: 2

清空任务



目录

Contents

◆ 组件的三大组成部分

scoped样式冲突 / data是一个函数

◆ 组件通信

组件通信语法 / 什么是prop / props校验 / props和data区别 & 单向数据流

◆ 综合案例：小黑记事本 (组件版)

拆分组件 / 渲染 / 添加 / 删除 / 统计 / 清空 / 持久化

◆ 进阶语法

v-model原理 / v-model应用于组件 / sync修饰符 / ref 和 \$refs / \$nextTick

组件的三大组成部分 - 注意点说明

结构
<template>

只能有一个根元素

样式
<style>

全局样式(默认): 影响所有组件
局部样式: **scoped** 下样式, 只作用于
当前组件

逻辑
<script>

el 根实例独有, **data** 是一个函数,
其他配置项一致

组件的样式冲突 scoped

默认情况：写在组件中的样式会 **全局生效** → 因此很容易造成多个组件之间的样式冲突问题。

1. **全局样式：**默认组件中的样式会作用到全局
2. **局部样式：**可以给组件加上 **scoped** 属性, 可以让样式只作用于当前组件

scoped原理?

1. 当前组件内标签都被添加 **data-v-hash值** 的属性
2. css选择器都被添加 **[data-v-hash值]** 的属性选择器

最终效果：必须是当前组件的元素, 才会有这个自定义属性, 才会被这个样式作用到

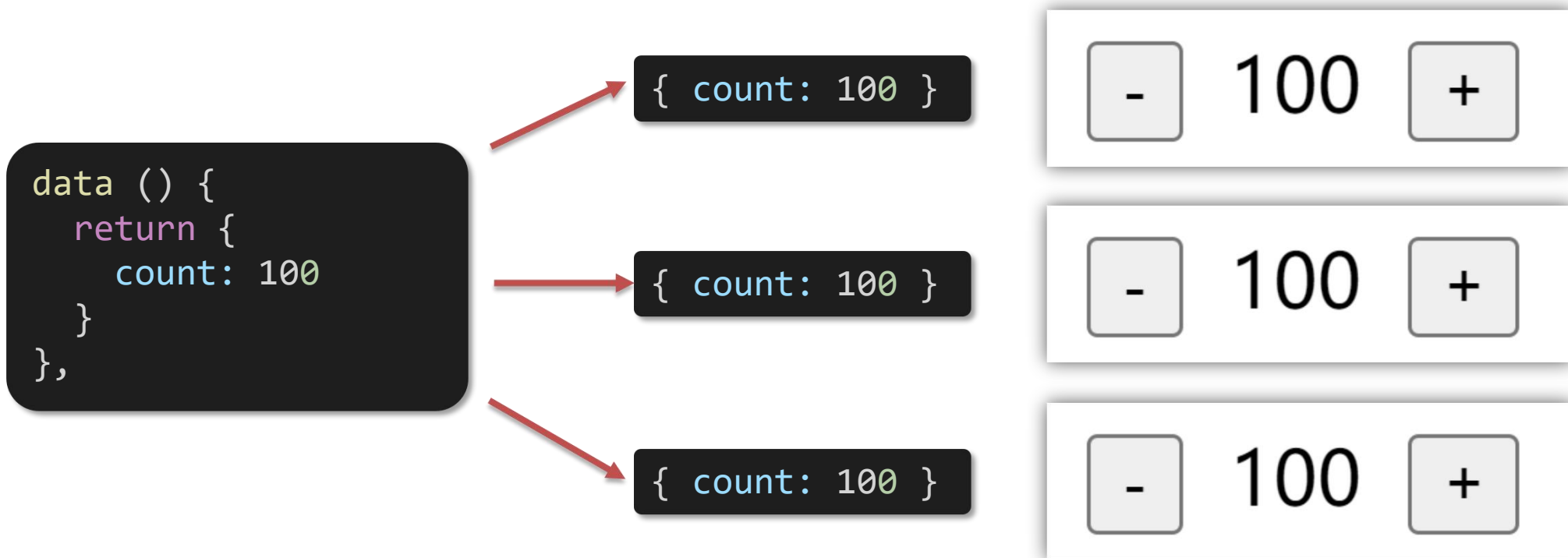
```
▼ <div data-v-cbe7c9bc> == $0
  <p data-v-cbe7c9bc>我是hm-header</p>
</div>
<div>我是普通盒子</div>
```

```
div[data-v-cbe7c9bc] {
  border: 1px solid #000;
  margin: 10px 0;
}
```

data 是一个函数

一个组件的 **data** 选项必须是一个**函数**。→ 保证每个组件实例，维护**独立**的一份数据对象。

每次创建新的组件实例，都会新执行一次 data 函数，得到一个新对象。






总结

组件三大组成部分的注意点：

1. 结构：有且只能一个根元素
2. 样式：默认全局样式，加上 **scoped** 局部样式
3. 逻辑：**data**是一个函数，保证数据独立。



目录

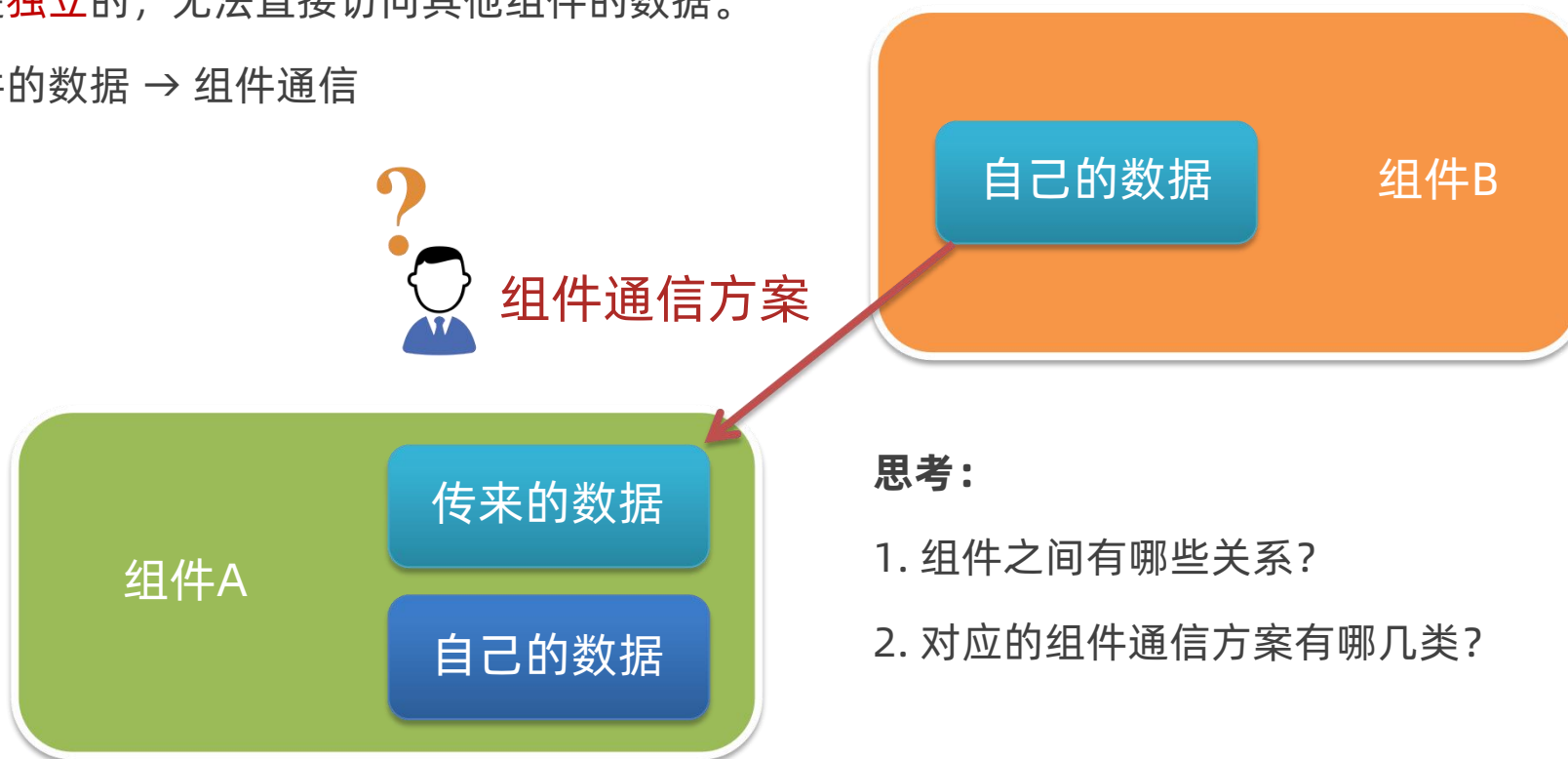
Contents

- ◆ 组件三大组成部分 (注意点)
- ◆ 组件通信
- ◆ 进阶用法
- ◆ 综合案例

什么是组件通信

组件通信, 就是指 **组件与组件** 之间的**数据传递**。

- 组件的数据是**独立**的，无法直接访问其他组件的数据。
- 想用其他组件的数据 → 组件通信



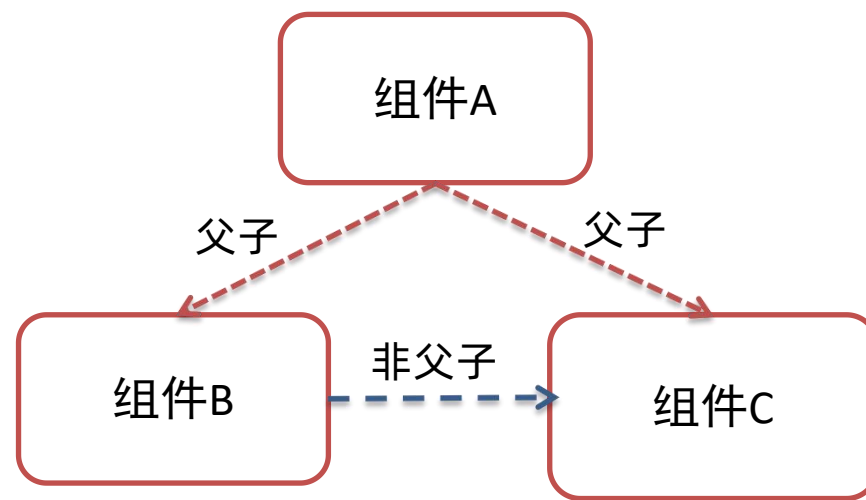
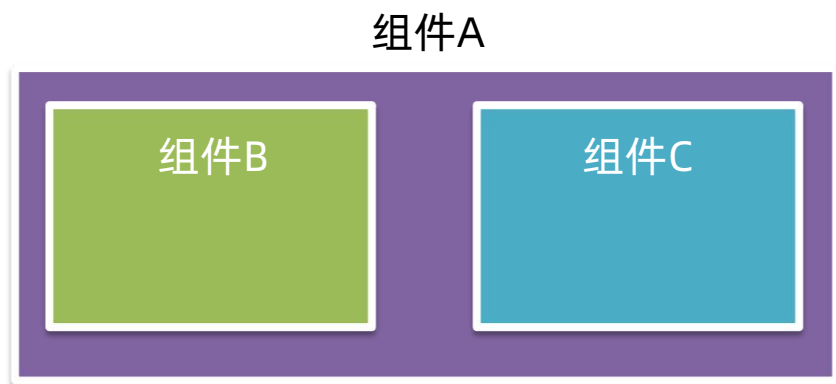
思考：

1. 组件之间有哪些关系？
2. 对应的组件通信方案有哪几类？

不同的组件关系 和 组件通信方案分类

组件关系分类：

1. 父子关系
2. 非父子关系

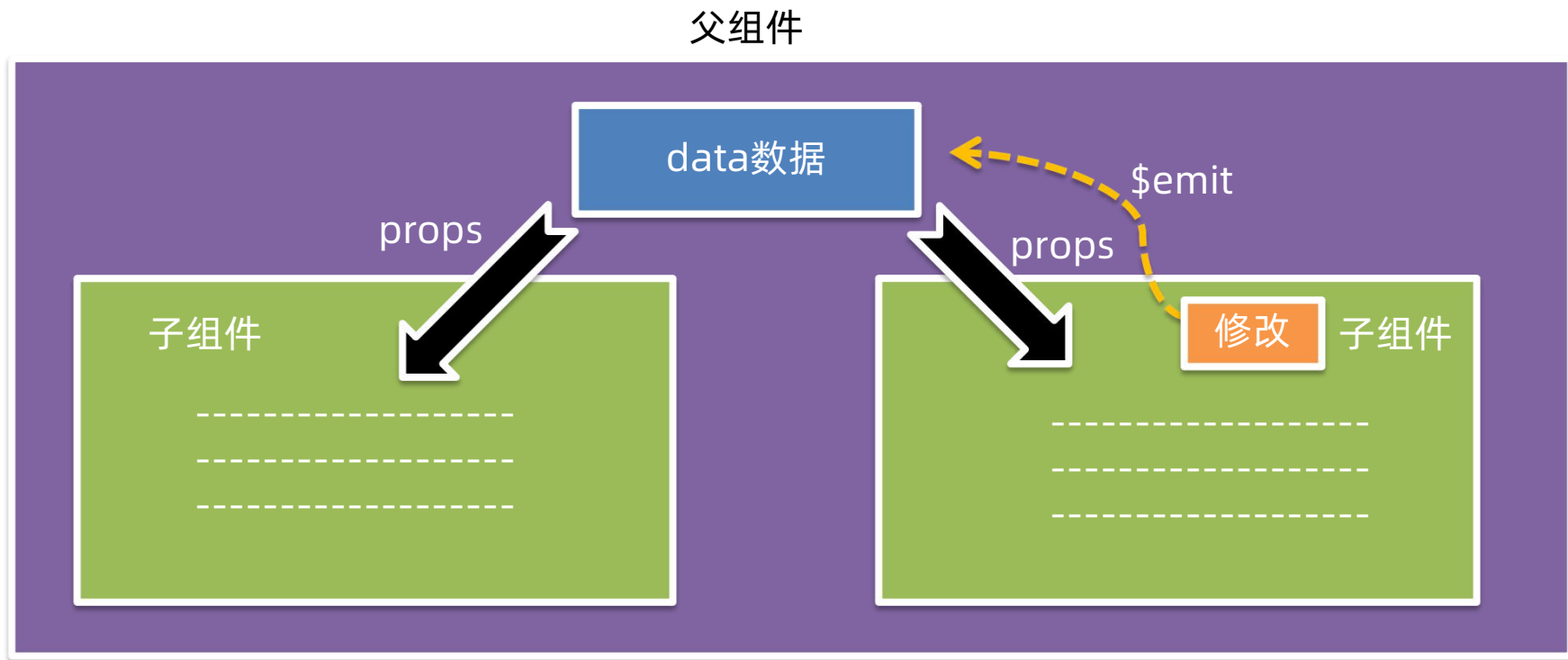


组件通信解决方案：



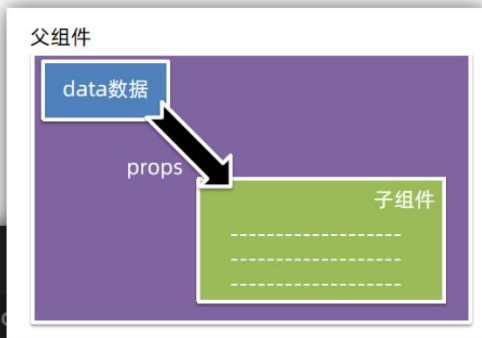
父子通信流程图：

1. 父组件通过 `props` 将数据传递给子组件
2. 子组件利用 `$emit` 通知父组件修改更新



父 → 子

父组件通过 **props** 将数据传递给子组件



```
src > App.vue M × 父组件
src > App.vue > {} "App.vue" > template > div#app
1 <template>
2   <div id="app" style="border: 3px solid #000; margin:
3     我是父 App组件
4     <!-- 1. 给子组件以添加属性的方式传值 -->
5     <Son :title="msg"></Son>
6   </div>
7 </template>
8
9 <script>
10 import Son from './components/Son.vue'
11 export default {
12   name: 'App',
13   data () {
14     return {
15       msg: '学前端来黑马'
16     }
17   },
18
src > components > Son.vue U × 子组件
src > components > Son.vue > {} "Son
1 <template>
2   <div style="border: 3px solid #000; margin: 10px"
3     <!-- 3. 模板中直接使用 -->
4     我是子 Son组件 - {{ title }}
5   </div>
6 </template>
7
8 <script>
9 export default {
10   name: 'SonIndex',
11   // 2. 子组件内部通过 props 接收
12   props: ['title']
13 }
14 </script>
15
16 <style scoped>
17
```

子 → 父

子组件利用 `$emit` 通知父组件，进行修改更新

▼ App.vue U × 父组件

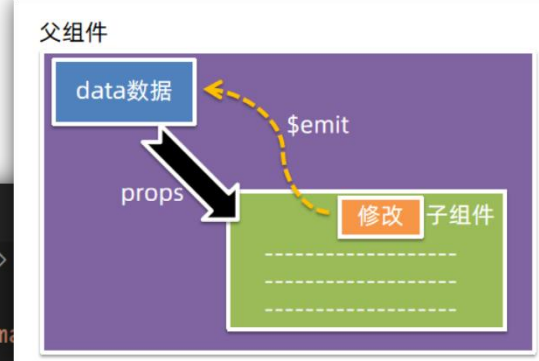
03-组件通信-父子通信 > ▼ App.vue > {} "App.vue" > script > default > methods

```
1 <template>
2   <div id="app" style="border: 3px solid #000; margin:
3     我是App组件
4     <!-- 2. 父组件监听事件 -->
5     <Son :title="msg" @changeTitle="handleChange"></Son>
6   </div>
7 </template>
8
9 <script>
10 import Son from './components/Son'
11 export default {
12   data () {
13     return {
14       msg: '黑马程序员'
15     }
16   },
17   methods: {
18     // 3. 提供处理函数，形参中获取参数
19     handleChange (title) {
20       this.msg = title
21     }
22   },
```

▼ Son.vue U × 子组件

03-组件通信-父子通信 > components > ▼ Son.vue > {} "Son.vue" >

```
1 <template>
2   <div style="border: 3px solid #000; ma
3     我是Son组件 {{ title }}
4     <button @click="changeTitle">修改标题</button>
5   </div>
6 </template>
7
8 <script>
9 export default {
10   props: ['title'],
11   methods: {
12     changeTitle () {
13       // 1. $emit触发事件，给父组件发送消息通知
14       this.$emit('changeTitle', '传智教育')
15     }
16   }
17 }
18 </script>
19
20 <style scoped>
21
22 </style>
```





总结

1. 两种组件关系分类 和 对应的组件通信方案

父子关系 → props & \$emit

非父子关系 → provide & inject 或 eventbus

通用方案 → vuex

2. 父子通信方案的核心流程

2.1 父传子props:

① 父中给子添加属性传值 ② 子props 接收 ③ 子组件使用

2.2 子传父\$emit:

① 子\$emit 发送消息 ②父中给子添加消息监听 ③ 父中实现处理函数

什么是 prop

Prop 定义：组件上注册的一些 自定义属性

Prop 作用：向子组件传递数据

特点：

- 可以传递 任意数量 的prop
- 可以传递 任意类型 的prop

我是个人信息组件

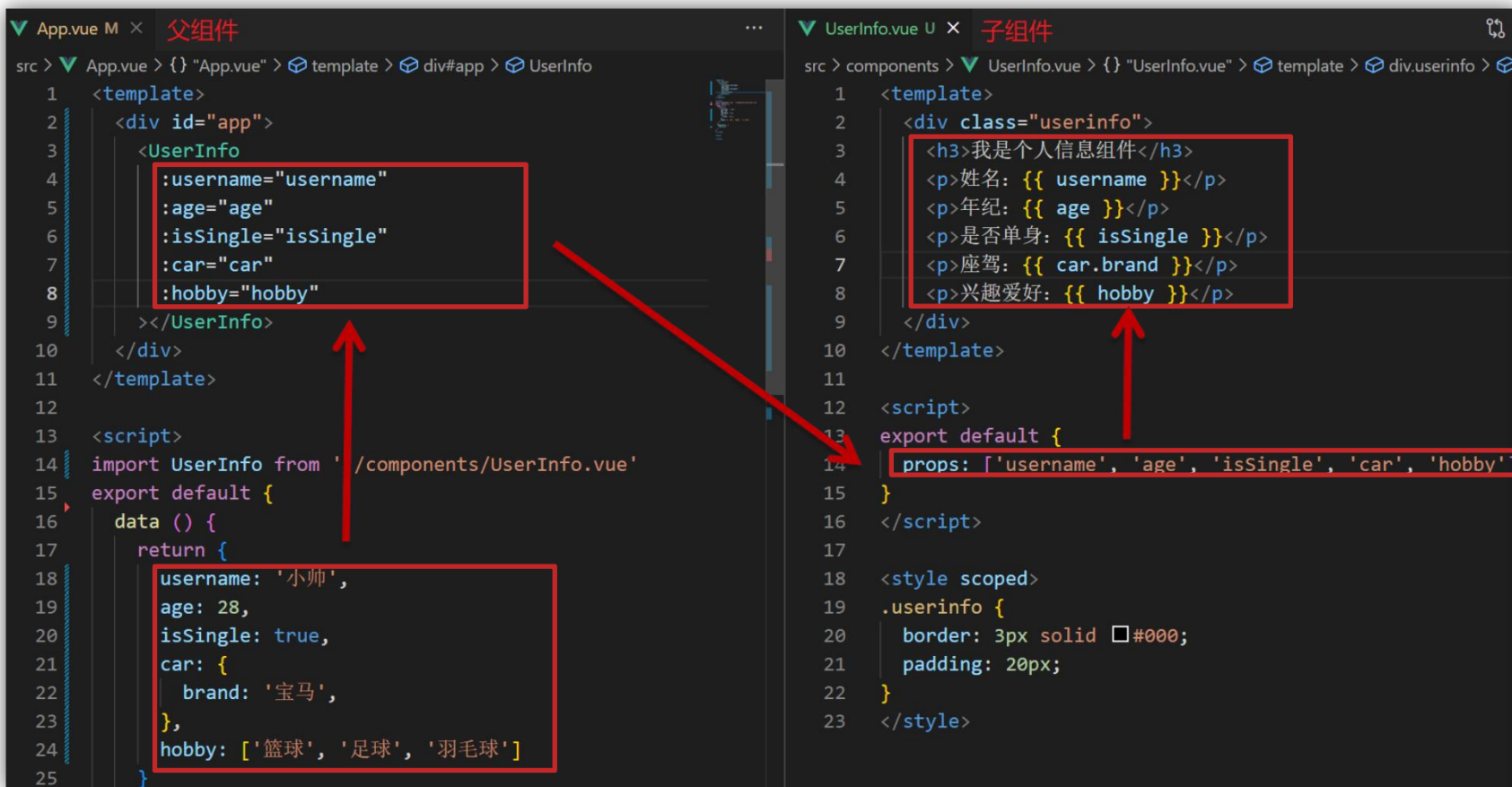
姓名：小帅

年纪：28

是否单身：是

座驾：宝马

兴趣爱好：篮球、足球、羽毛球



```
App.vue M x 父组件
src > App.vue > {} "App.vue" > template > div#app > UserInfo
1 <template>
2   <div id="app">
3     <UserInfo
4       :username="username"
5       :age="age"
6       :isSingle="isSingle"
7       :car="car"
8       :hobby="hobby"
9     ></UserInfo>
10  </div>
11 </template>
12
13 <script>
14 import UserInfo from './components/UserInfo.vue'
15 export default {
16   data () {
17     return {
18       username: '小帅',
19       age: 28,
20       isSingle: true,
21       car: {
22         brand: '宝马',
23       },
24       hobby: ['篮球', '足球', '羽毛球']
25     }
  }
}

UserInfo.vue U x 子组件
src > components > UserInfo.vue > {} "UserInfo.vue" > template > div.userinfo >
1 <template>
2   <div class="userinfo">
3     <h3>我是个人信息组件</h3>
4     <p>姓名: {{ username }}</p>
5     <p>年纪: {{ age }}</p>
6     <p>是否单身: {{ isSingle }}</p>
7     <p>座驾: {{ car.brand }}</p>
8     <p>兴趣爱好: {{ hobby }}</p>
9   </div>
10 </template>
11
12 <script>
13 export default {
14   props: ['username', 'age', 'isSingle', 'car', 'hobby']
15 }
16 </script>
17
18 <style scoped>
19 .userinfo {
20   border: 3px solid #000;
21   padding: 20px;
22 }
23 </style>
```

props 校验

思考：组件的 prop 可以乱传么？

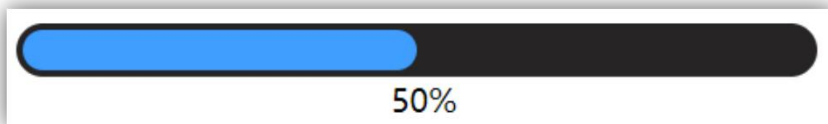
作用：为组件的 prop 指定**验证要求**，不符合要求，控制台就会有**错误提示** → 帮助开发者，快速发现错误

语法：

- ① 类型校验
- ② 非空校验
- ③ 默认值
- ④ 自定义校验

```
props: {  
  校验的属性名: 类型 // Number String Boolean ...  
},
```

```
props: {  
  校验的属性名: {  
    type: 类型, // Number String Boolean ...  
    required: true, // 是否必填  
    default: 默认值, // 默认值  
    validator (value) {  
      // 自定义校验逻辑  
      return 是否通过校验  
    }  
  }  
},
```



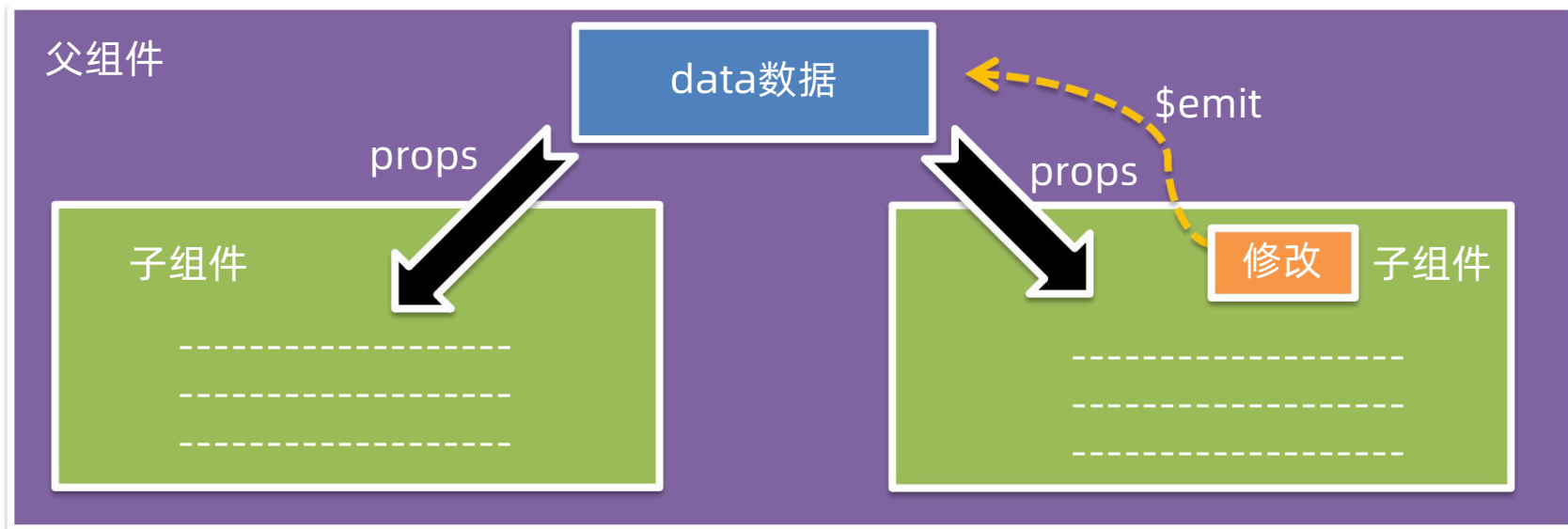
prop & data、单向数据流

共同点：都可以给组件提供数据。

区别：

- data 的数据是**自己的** → 随便改
- prop 的数据是**外部的** → 不能直接改，要遵循 **单向数据流**

单向数据流：父级 prop 的数据更新，会向下流动，影响子组件。这个数据流动是单向的。



口诀：谁的数据谁负责

组件通信案例：小黑记事本 - 组件版

需求说明：

- ① 拆分基础组件
- ② 渲染待办任务
- ③ 添加任务
- ④ 删除任务
- ⑤ 底部合计 和 清空功能
- ⑥ 持久化存储





小结

核心步骤：

① 拆分基础组件

新建组件 → 拆分存放结构 → 导入注册使用

② 渲染待办任务

提供数据(公共父组件) → 父传子传递 list → v-for 渲染

③ 添加任务

收集数据 v-model → 监听事件 → 子传父传递任务 → 父组件 unshift

④ 删除任务

监听删除携带 id → 子传父传递 id → 父组件 filter 删除

⑤ 底部合计 和 清空功能

底部合计：父传子传递 list → 合计展示

清空功能：监听点击 → 子传父通知父组件 → 父组件清空

⑥ 持久化存储：watch监视数据变化，持久化到本地

非父子通信 (拓展) - event bus 事件总线

作用：非父子组件之间，进行简易消息传递。(复杂场景 → Vuex)

1. 创建一个都能访问到的事件总线 (空 Vue 实例) → utils/EventBus.js

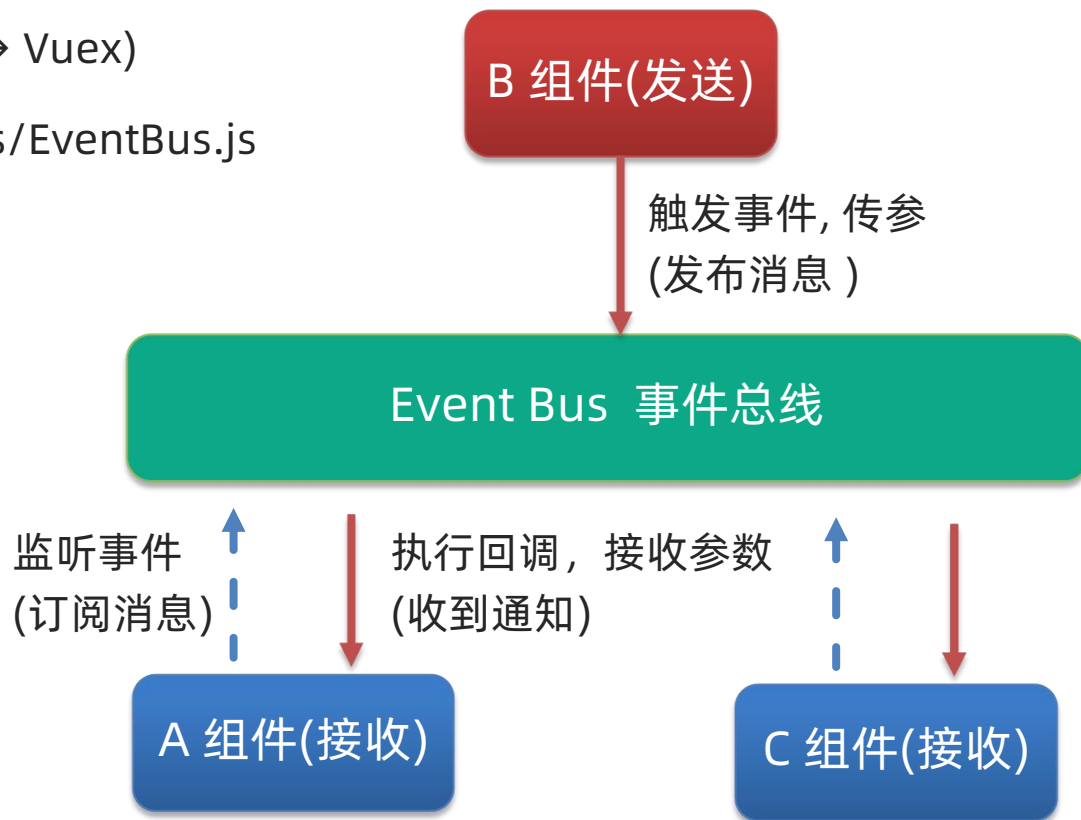
```
import Vue from 'vue'
const Bus = new Vue()
export default Bus
```

2. A 组件(接收方), 监听 Bus 实例的事件

```
created () {
  Bus.$on('sendMsg', (msg) => {
    this.msg = msg
  })
}
```

3. B 组件(发送方), 触发 Bus 实例的事件

```
Bus.$emit('sendMsg', '这是一个消息')
```



非父子通信 (拓展) - provide & inject

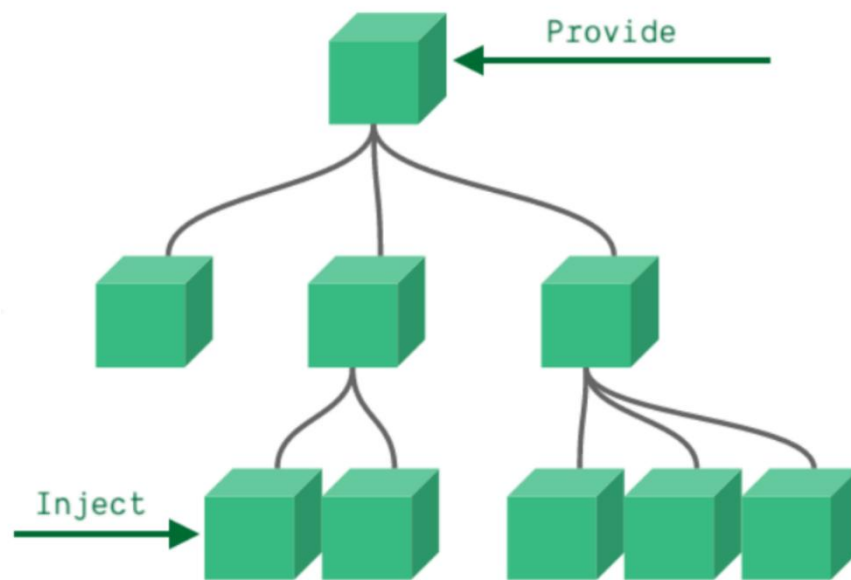
provide & inject 作用：跨层级共享数据。

1. 父组件 provide 提供数据

```
export default {  
  provide () {  
    return {  
      // 普通类型【非响应式】  
      color: this.color,  
      // 复杂类型【响应式】  
      userInfo: this.userInfo,  
    }  
  }  
}
```

2. 子/孙组件 inject 取值使用

```
export default {  
  inject: ['color', 'userInfo'],  
  created () {  
    console.log(this.color, this.userInfo)  
  }  
}
```





目录

Contents

- ◆ 组件三大组成部分 (注意点)
- ◆ 组件通信
- ◆ 进阶用法
- ◆ 综合案例

v-model 原理

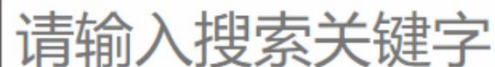
原理：v-model本质上是一个**语法糖**。例如应用在输入框上，就是 **value属性** 和 **input事件** 的合写。

作用：提供数据的双向绑定

① 数据变，视图跟着变 **:value**

② 视图变，数据跟着变 **@input**

注意：**\$event** 用于在模板中，获取事件的形参



```
<template>
  <div id="app" >
    <input v-model="msg" type="text">

    <input :value="msg" @input="msg = $event.target.value" type="text">
  </div>
</template>
```

表单类组件封装 & v-model 简化代码

1. 表单类组件 **封装** → 实现 子组件 和 父组件数据 的**双向绑定**

① **父传子**：数据 应该是父组件 **props** 传递 过来的，**拆解 v-model** 绑定数据

② **子传父**：监听输入，子传父传值给父组件修改

```
<BaseSelect :cityId="selectId" @事件名="selecteId = $event" />
```

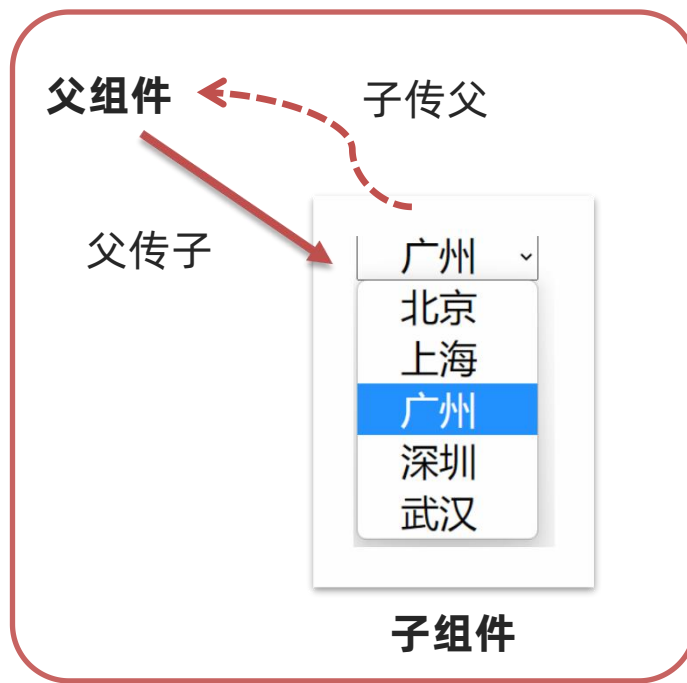
父组件
(使用)

```
<select :value="cityId" @change="handleChange">...</select>
```

```
props: {  
  cityId: String  
},
```

```
methods: {  
  handleChange (e) {  
    this.$emit('事件名', e.target.value)  
  }  
}
```

子组件
(封装)



表单类组件封装 & v-model 简化代码

2. 父组件 v-model 简化代码，实现 子组件 和 父组件数据 双向绑定

① 子组件中：props 通过 value 接收，事件触发 input

② 父组件中：v-model 给组件直接绑数据 (:value + @input)

```
<BaseSelect v-model="selectId"></BaseSelect>
```

父组件
(使用)

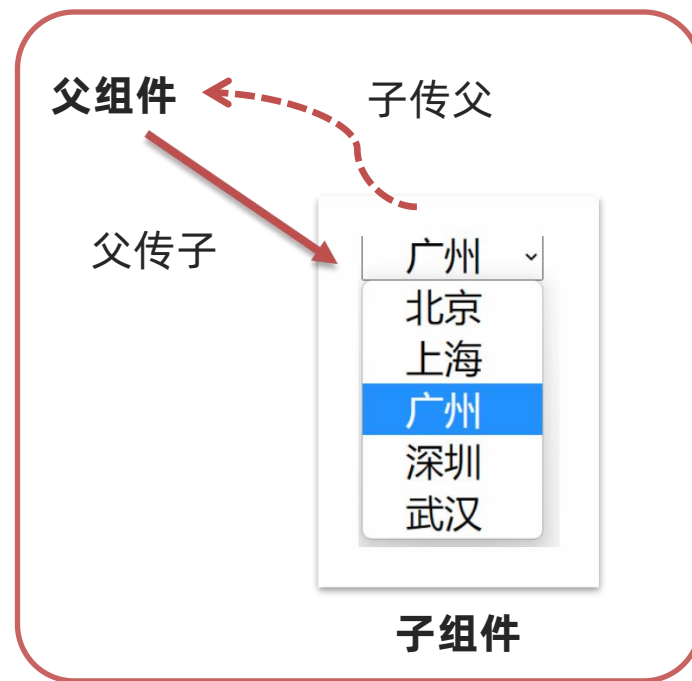
```
<BaseSelect :value="selectId" @input="selectId = $event" />
```

```
<select :value="value" @change="handleChange">...</select>
```

```
props: {  
  value: String  
},
```

```
methods: {  
  handleChange (e) {  
    this.$emit('input', e.target.value)  
  }  
}
```

子组件
(封装)





总结

1. 表单类基础组件封装思路

① 父传子：父组件动态传递 `prop` 数据，拆解 `v-model`，绑定数据

② 子传父：监听输入，子传父传值给父组件修改

本质：实现了实现 子组件 和 父组件数据 的双向绑定

2. v-model 简化代码的核心步骤

① 子组件中：props 通过 `value` 接收，事件触发 `input`

② 父组件中：`v-model` 给组件直接绑数据

3. 小作业：封装输入框组件，利用v-model简化代码

.sync 修饰符

作用：可以实现 子组件 与 父组件数据 的 双向绑定，简化代码

特点：prop属性名，可以自定义，非固定为 value

场景：封装弹框类的基础组件， visible属性 true显示 false隐藏

本质：就是 :属性名 和 @update:属性名 合写

```
<BaseDialog :visible.sync="isShow" />
-----
<BaseDialog
  :visible="isShow"
  @update:visible="isShow = $event"
/>
```

父组件
(使用)

```
props: {
  visible: Boolean
},
this.$emit('update:visible', false)
```

子组件
(封装)



ref 和 \$refs

作用：利用 ref 和 \$refs 可以用于 获取 dom 元素, 或 组件实例

特点：查找范围 → 当前组件内 (更精确稳定)

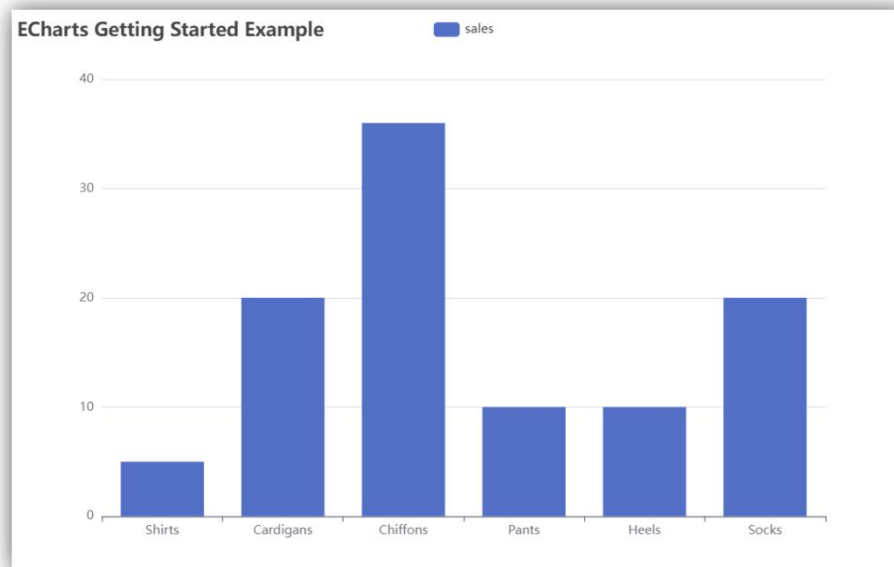
① 获取 dom:

1. 目标标签 - 添加 ref 属性

```
<div ref="chartRef">我是渲染图表的容器</div>
```

2. 恰当时机, 通过 this.\$refs.xxx, 获取目标标签

```
mounted () {  
  console.log(this.$refs.chartRef)  
},
```



```
// 基于准备好的dom, 初始化echarts实例  
const myChart = echarts.init(document.querySelector('.box'));
```

querySelector 查找范围 → 整个页面

ref 和 \$refs

作用：利用 ref 和 \$refs 可以用于 获取 dom 元素, 或 组件实例

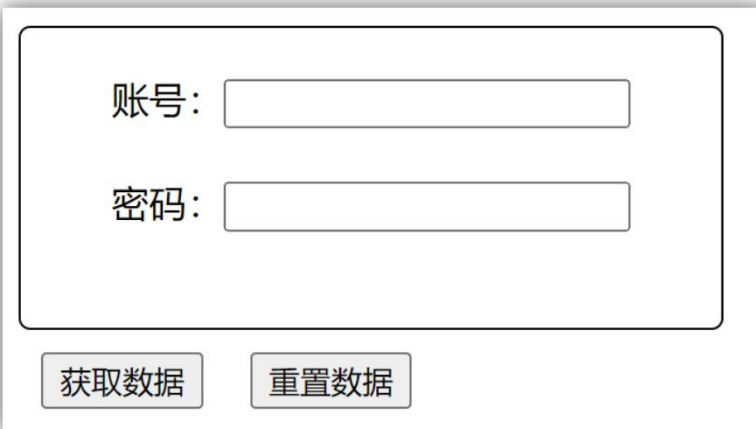
② 获取组件：

1. 目标组件 - 添加 ref 属性

```
<BaseForm ref="baseForm"></BaseForm>
```

2. 恰当时机, 通过 this.\$refs.xxx, 获取目标组件,
就可以调用组件对象里面的方法

```
this.$refs.baseForm.组件方法()
```



账号:

密码:

获取数据 重置数据

表单组件
内部已实现了获取和重置

Vue异步更新、\$nextTick

需求：编辑标题，编辑框自动聚焦

1. 点击编辑，显示编辑框
2. 让编辑框，立刻获取焦点

```
this.isShowEdit = true // 显示输入框  
this.$refs.inp.focus() // 获取焦点
```

问题："显示之后"，立刻获取焦点是不能成功的！

原因：Vue 是 异步更新 DOM (提升性能)



Vue异步更新、\$nextTick

\$nextTick: 等 DOM 更新后, 才会触发执行此方法里的函数体

语法: this.\$nextTick(函数体)

```
this.$nextTick(() => {  
  this.$refs.inp.focus()  
})
```

大标题





总结

1. Vue是异步更新 DOM 的
2. 想要在 DOM 更新完成之后做某件事，可以使用 `$nextTick`

```
this.$nextTick(() => {  
  // 业务逻辑  
})
```

综合案例 - 商品列表

需求说明：

① 表格动态渲染

② my-tag组件封装

(1) 双击显示输入框，回显标签信息

(2) 失去焦点，隐藏输入框

(3) 输入框内容修改，回车 → 修改标签信息

编号	图片	名称	标签
101		梨皮朱泥三绝清代小品壶经典款紫砂壶	茶具
102		全防水HABU旋钮牛皮户外徒步鞋山宁泰抗菌	男鞋
103		毛茸茸小熊出没，儿童羊羔绒背心73-90cm	儿童服饰
104		基础百搭，儿童套头针织毛衣1-9岁	儿童服饰



传智教育旗下高端IT教育品牌