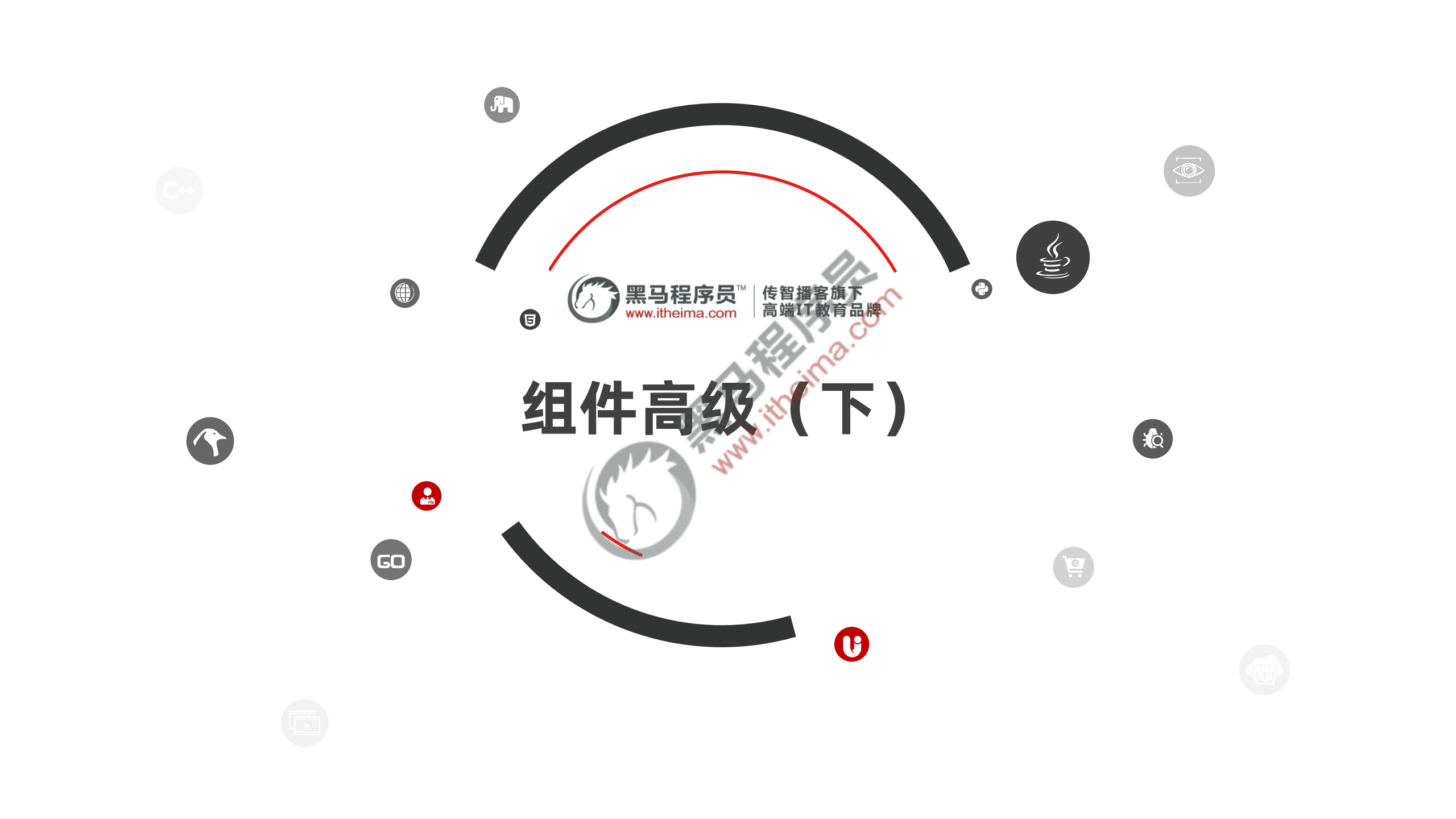


组件高级（下）

目录 Contents

ref 引用

1. 什么是 ref 引用

ref 用来辅助开发者在不依赖于 jQuery 的情况下，获取 DOM 元素或组件的引用。

每个 vue 的组件实例上，都包含一个 $refs 对象，里面存储着对应的 DOM 元素或组件的引用。默认情况下，组件的 $refs 指向一个空对象。

```
1 <template>
2   <h3>MyRef 组件</h3>
3   <button @click="getRef">获取 $refs 引用</button>
4 </template>
5
6 <script>
7 export default {
8   methods: {
9     getRef() { console.log(this) } // this 代表当前组件的实例对象， this.$refs 默认指向空对象
10  }
11 }
12 </script>
```

ref 引用

2. 使用 ref 引用 DOM 元素

如果想要使用 ref 引用页面上的 DOM 元素，则可以按照如下的方式进行操作：

```
<!-- 使用 ref 属性，为对应的 DOM 添加引用名称 -->
<h3 ref="myh3">MyRef 组件</h3>
<button @click="getRef">获取 $refs 引用</button>

methods: {
  getRef() {
    // 通过 this.$refs.引用的名称 可以获取到 DOM 元素的引用
    console.log(this.$refs.myh3)
    // 操作 DOM 元素，把文本颜色改为红色
    this.$refs.myh3.style.color = 'red'
  },
}
```

ref 引用

3. 使用 ref 引用组件实例

如果想要使用 ref 引用页面上的组件实例，则可以按照如下的方式进行操作：

```
<!-- 使用 ref 属性，为对应的“组件”添加引用名称 -->
<my-counter ref="counterRef"></my-counter>
<button @click="getRef">获取 $refs 引用</button>

methods: {
  getRef() {
    // 通过 this.$refs.引用的名称 可以引用组件的实例
    console.log(this.$refs.counterRef)
    // 引用到组件的实例之后，就可以调用组件上的 methods 方法
    this.$refs.counterRef.add()
  },
}
```

4. 控制文本框和按钮的按需切换

通过布尔值 inputVisible 来控制组件中的文本框与按钮的按需切换。示例代码如下：

```
<template>
  <input type="text" v-if="inputVisible">
  <button v-else @click="showInput">展示input输入框</button>
</template>
```

```
<script>
export default {
  data() {
    return {
      // 控制文本框和按钮的按需切换
      inputVisible: false,
    }
  },
  methods: {
    showInput() { // 切换布尔值，显示文本框
      this.inputVisible = true
    },
  },
}
</script>
```

5. 让文本框自动获得焦点

当文本框展示出来之后，如果希望它立即获得焦点，则可以为其添加 ref 引用，并调用原生 DOM 对象的 .focus() 方法即可。示例代码如下：

```
<input type="text" v-if="inputVisible" ref="ipt">
<button v-else @click="showInput">展示input输入框</button>

methods: {
  showInput() {
    this.inputVisible = true
    // 获取文本框的 DOM 引用，并调用 .focus() 使其自动获得焦点
    this.$refs.ipt.focus()
  },
}
```

6. this.$nextTick(cb) 方法

组件的 $nextTick(cb) 方法，会把 cb 回调推迟到下一个 DOM 更新周期之后执行。通俗的理解是：等组件的 DOM 异步地重新渲染完成后，再执行 cb 回调函数。从而能保证 cb 回调函数可以操作到最新的 DOM 元素。

```
<input type="text" v-if="inputVisible" ref="ipt">
<button v-else @click="showInput">展示input输入框</button>

methods: {
  showInput() {
    this.inputVisible = true
    // 把对 input 文本框的操作，推迟到下次 DOM 更新之后。否则页面上根本不存在文本框元素
    this.$nextTick(() => {
      this.$refs.ipt.focus()
    })
  },
}
```

目录 Contents

动态组件

1. 什么是动态组件

动态组件指的是动态切换组件的显示与隐藏。vue 提供了一个内置的 <component> 组件，专门用来实现组件的动态渲染。

① <component> 是组件的占位符

② 通过 is 属性动态指定要渲染的组件名称

③ <component is="要渲染的组件的名称"></component>

动态组件

2. 如何实现动态组件渲染

示例代码如下：

```
data() {
  return {
    comName: 'my-dynamic-1' // 1. 当前要渲染的组件的名称
  }
}

<template>
  <!-- 3. 点击按钮，动态切换组件的名称 -->
  <button @click="comName='my-dynamic-1'">组件1</button>
  <button @click="comName='my-dynamic-2'">组件2</button>
  <!-- 2. 通过 is 属性，动态指定要渲染的组件的名称 -->
  <component :is="comName"></component>
</template>
```

3. 使用 keep-alive 保持状态

默认情况下，切换动态组件时无法保持组件的状态。此时可以使用 vue 内置的 <keep-alive> 组件保持动态组件的状态。示例代码如下：

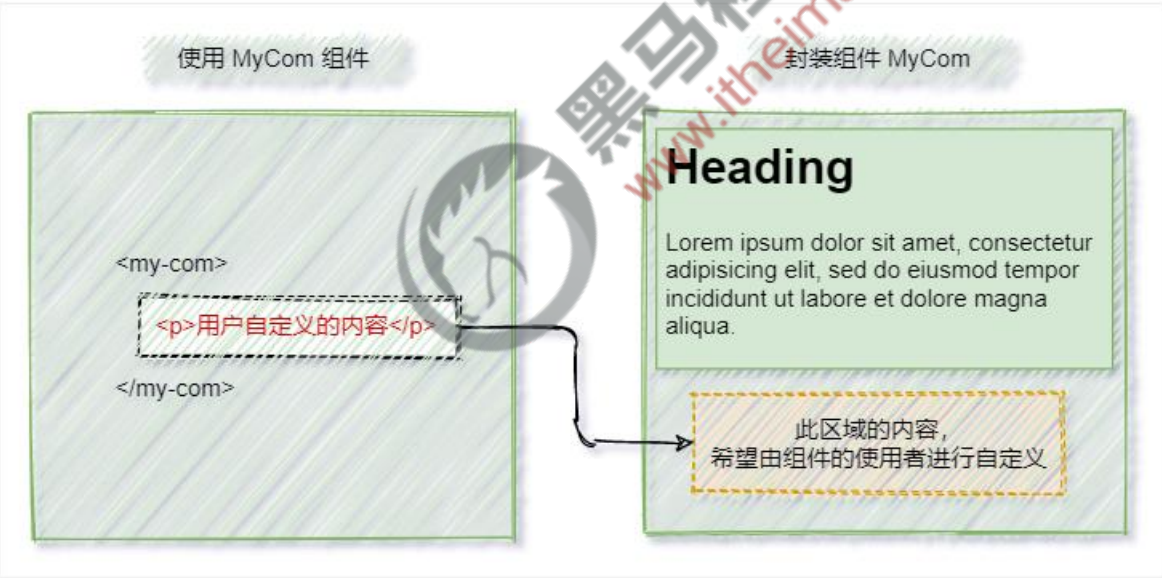
```
<keep-alive>
  <component :is="comName"></component>
</keep-alive>
```

目录 Contents

插槽

1. 什么是插槽

插槽（Slot）是 vue 为组件的封装者提供的能力。允许开发者在封装组件时，把不确定的、希望由用户指定的部分定义为插槽。



可以把插槽认为是组件封装期间，为用户预留的内容的占位符。

插槽

2. 体验插槽的基础用法

在封装组件时，可以通过 <slot> 元素定义插槽，从而为用户预留内容占位符。示例代码如下：

```
<template>
  <p>这是 MyCom1 组件的第 1 个 p 标签</p>
  <!-- 通过 slot 标签，为用户预留内容占位符（插槽） -->
  <slot></slot>
  <p>这是 MyCom1 组件最后一个 p 标签</p>
</template>
```

```
<my-com-1>
  <!-- 在使用 MyCom1 组件时，为插槽指定具体的内容 -->
  <p>~~~用户自定义的内容~~~</p>
</my-com-1>
```

2.1 没有预留插槽的内容会被丢弃

如果在封装组件时没有预留任何 <slot> 插槽，则用户提供的任何自定义内容都会被丢弃。示例代码如下：

```
<template>
  <p>这是 MyCom1 组件的第 1 个 p 标签</p>
  <!-- 封装组件时吗，没有预留任何插槽 -->
  <p>这是 MyCom1 组件最后一个 p 标签</p>
</template>
```

```
<my-com-1>
  <!-- 自定义的内容会被丢弃 -->
  <p>~~~用户自定义的内容~~~</p>
</my-com-1>
```

2.2 后备内容

封装组件时，可以为预留的 <slot> 插槽提供后备内容（默认内容）。如果组件的使用者没有为插槽提供任何内容，则后备内容会生效。示例代码如下：

```
<template>
  <p>这是 MyCom1 组件的第 1 个 p 标签</p>
  <slot>这是后备内容</slot>
  <p>这是 MyCom1 组件最后一个 p 标签</p>
</template>
```

3. 具名插槽

如果在封装组件时需要预留多个插槽节点，则需要为每个 <slot> 插槽指定具体的 name 名称。这种带有具体名称的插槽叫做“具名插槽”。示例代码如下：

```
<div class="container">
  <header>
    <!-- 我们希望把页头放这里 -->
    <slot name="header"></slot>
  </header>
  <main>
    <!-- 我们希望把主要内容放这里 -->
    <slot></slot>
  </main>
  <footer>
    <!-- 我们希望把页脚放这里 -->
    <slot name="footer"></slot>
  </footer>
</div>
```

注意：没有指定 name 名称的插槽，会有隐含的名称叫做 “default”。

3.1 为具名插槽提供内容

在向具名插槽提供内容的时候，我们可以在一个 <template> 元素上使用 v-slot 指令，并以 v-slot 的参数的形式提供其名称。示例代码如下：

```
<my-com-2>
  <template v-slot:header>
    <h1>滕王阁序</h1>
  </template>

  <template v-slot:default>
    <p>豫章故郡，洪都新府。</p>
    <p>星分翼轸，地接衡庐。</p>
    <p>襟三江而带五湖，控蛮荆而引瓯越。</p>
  </template>

  <template v-slot:footer>
    <p>落款：王勃</p>
  </template>
</my-com-2>
```

3.2 具名插槽的简写形式

跟 v-on 和 v-bind 一样，v-slot 也有缩写，即把参数之前的所有内容 (v-slot:) 替换为字符 #。例如 v-slot:header 可以被重写为 #header

```
<my-com-2>
  <template #header>
    <h1>滕王阁序</h1>
  </template>

  <template #default>
    <p>豫章故郡，洪都新府。</p>
    <p>星分翼轸，地接衡庐。</p>
    <p>襟三江而带五湖，控蛮荆而引瓯越。</p>
  </template>

  <template #footer>
    <p>落款：王勃</p>
  </template>
</my-com-2>
```

4. 作用域插槽

在封装组件的过程中，可以为预留的 <slot> 插槽绑定 props 数据，这种带有 props 数据的 <slot> 叫做“作用域插槽”。示例代码如下：

```
<div>
  <h3>这是 TEST 组件</h3>
  <slot :info="infomation"></slot>
</div>

<!-- 使用自定义组件 -->
<my-test>
  <template v-slot:default="scope">
    {{ scope }}
  </template>
</my-test>
```

4.1 解构作用域插槽的 Prop

作用域插槽对外提供的数据对象，可以使用解构赋值简化数据的接收过程。示例代码如下：

```
<my-table>
  <!-- v-slot: 可以简写成 # -->
  <!-- 作用域插槽对外提供的数据对象，可以通过“解构赋值”简化接收的过程 -->
  <template #default="{ user }">
    <!-- 【使用】作用域插槽的数据 -->
    <td>{{ user.id }}</td>
    <td>{{ user.name }}</td>
    <td>{{ user.state }}</td>
  </template>
</my-table>
```

4.2 声明作用域插槽

在封装 MyTable 组件的过程中，可以通过作用域插槽把表格每一行的数据传递给组件的使用者。示例代码如下：

```
<!-- 表格主体区域 -->
<tbody>
  <!-- 循环渲染表格数据 -->
  <tr v-for="item in list" :key="item.id">
    <!-- 下面的 slot 是一个【作用域插槽】 -->
    <slot :user="item"></slot>
  </tr>
</tbody>
```

4.3 使用作用域插槽

在使用 MyTable 组件时，自定义单元格的渲染方式，并接收作用域插槽对外提供的数据。示例代码如下：

```
<!-- 使用自定义组件 -->
<my-table>
  <!-- 【接收】作用域插槽对外提供的数据 -->
  <template v-slot:default="scope">
    <!-- 【使用】作用域插槽的数据 -->
    <td>{{ scope.user.id }}</td>
    <td>{{ scope.user.name }}</td>
    <td>{{ scope.user.state }}</td>
  </template>
</my-table>
```

目录 Contents

自定义指令

1. 什么是自定义指令

vue 官方提供了 v-for、v-model、v-if 等常用的内置指令。除此之外 vue 还允许开发者自定义指令。

vue 中的自定义指令分为两类，分别是：

- 私有自定义指令
- 全局自定义指令

自定义指令

2. 声明私有自定义指令的语法

在每个 vue 组件中，可以在 directives 节点下声明私有自定义指令。示例代码如下：

```
directives: {
  // 自定义一个私有指令
  focus: {
    // 当被绑定的元素插入到 DOM 中时，自动触发 mounted 函数
    mounted(el) {
      el.focus() // 让被绑定的元素自动获得焦点
    }
  }
}
```

自定义指令

3. 使用自定义指令

在使用自定义指令时，需要加上 v- 前缀。示例代码如下：

```
<!-- 声明自定义指令时，指令的名字是 focus -->
<!-- 使用自定义指令是，需要加上 v- 指令前缀 -->
<input v-focus />
```

自定义指令

4. 声明全局自定义指令的语法

全局共享的自定义指令需要通过“单页面应用程序的实例对象”进行声明，示例代码如下：

```
const app = Vue.createApp({})

// 注册一个全局自定义指令 `v-focus`
app.directive('focus', {
  // 当被绑定的元素插入到 DOM 中时，自动触发 mounted 函数
  mounted(el) {
    // Focus the element
    el.focus()
  }
})
```

自定义指令

5. updated 函数

mounted 函数只在元素第一次插入 DOM 时被调用，当 DOM 更新时 mounted 函数不会被触发。 updated 函数会在每次 DOM 更新完成后被调用。示例代码如下：

```
app.directive('focus', {
  mounted(el) { // 第一次插入 DOM 时触发这个函数
    el.focus()
  },
  updated(el) { // 每次 DOM 更新时都会触发 updated 函数
    el.focus()
  }
})
```

注意：在 vue2 的项目中使用自定义指令时，【 mounted -> bind 】【 updated -> update 】

6. 函数简写

如果 mounted 和updated 函数中的逻辑完全相同，则可以简写成如下格式：

```
app.directive('focus', (el) => {
  // 在 mounted 和 updated 时都会触发相同的业务处理
  el.focus()
})
```

自定义指令

7. 指令的参数值

在绑定指令时，可以通过“等号”的形式为指令绑定具体的参数值，示例代码如下：

```
<!-- 在使用 v-color 指令时，可以通过“等号”绑定指令的值 -->
<input type="text" v-model.number="count" v-focus v-color="'red'">
<p v-color="'cyan'">{{count}}</p>

<button @click="count++">+1</button>

// 自定义 v-color 指令
app.directive('color', (el, binding) => {
  // binding.value 就是通过“等号”为指令绑定的值
  el.style.color = binding.value
})
```

目录 Contents

Table 案例

1. 案例效果

#	商品名称	价格	标签	操作
1	Teenmix/天美意夏季专柜同款金色布女鞋6YF18BT6	¥298	+Tag 舒适 透气	删除
2	奥休斯(all shoes) 冬季保暖女士休闲雪地靴 舒适加绒防水短靴 防滑棉鞋子	¥89	+Tag 保暖 防滑	删除
3	初语秋冬新款毛衣女 套头宽松针织衫简约插肩袖上衣	¥199	+Tag 秋冬 毛衣	删除
4	佐露絲蕾丝衫女2020春秋装新款大码女装衬衫上衣雪纺衫韩版打底衫长袖	¥19	+Tag 雪纺衫 打底	删除
5	熙世界中长款长袖圆领毛衣女2022秋装新款假两件连衣裙女107SL170	¥178	+Tag 圆领 连衣裙	删除
6	烟花烫2021秋季装新品女装简约修身显瘦七分袖欧根纱连衣裙 花央	¥282	+Tag 秋季新品 显瘦	删除
7	韩都衣舍2021韩版女装秋装新宽松显瘦纯色系带长袖衬衫NG8201	¥128	+Tag 韩都衣舍 长袖衬衫	删除
8	预售纤莉秀大码女装胖妹妹秋装2020新款圆领百搭绣花胖mm休闲套头卫衣	¥128	+Tag 预售 卫衣	删除
9	莎密2022夏改良旗袍裙连衣裙修身复古时尚日常短款礼服旗袍	¥128	+Tag 莎密 礼服	删除
10	南极人秋冬韩版七彩棉加绒加厚一体保暖打底裤p7011	¥128	+Tag 南极人 打底裤	删除

Table 案例

2. 用到的知识点

- 组件封装
- 具名插槽
- 作用域插槽
- 自定义指令

Table 案例

3. 实现步骤

① 搭建项目的基本结构

② 请求商品列表的数据

③ 封装 MyTable 组件

④ 实现删除功能

⑤ 实现添加标签的功能

① 能够知道如何使用 ref 引用 DOM 和组件实例

- 通过 ref 属性指定引用的名称、使用 this.$refs 访问引用实例

② 能够知道 $nextTick 的调用时机

- 组件的 DOM 更新之后，才执行 $nextTick 中的回调

③ 能够说出 keep-alive 元素的作用

- 保持动态组件的状态

④ 能够掌握插槽的基本用法

- <slot> 标签、具名插槽、作用域插槽、v-slot: 简写为 #

⑤ 能够知道如何自定义指令

- 私有自定义指令、全局自定义指令

黑马程序员

www.itheima.com

传智播客旗下高端IT教育品牌