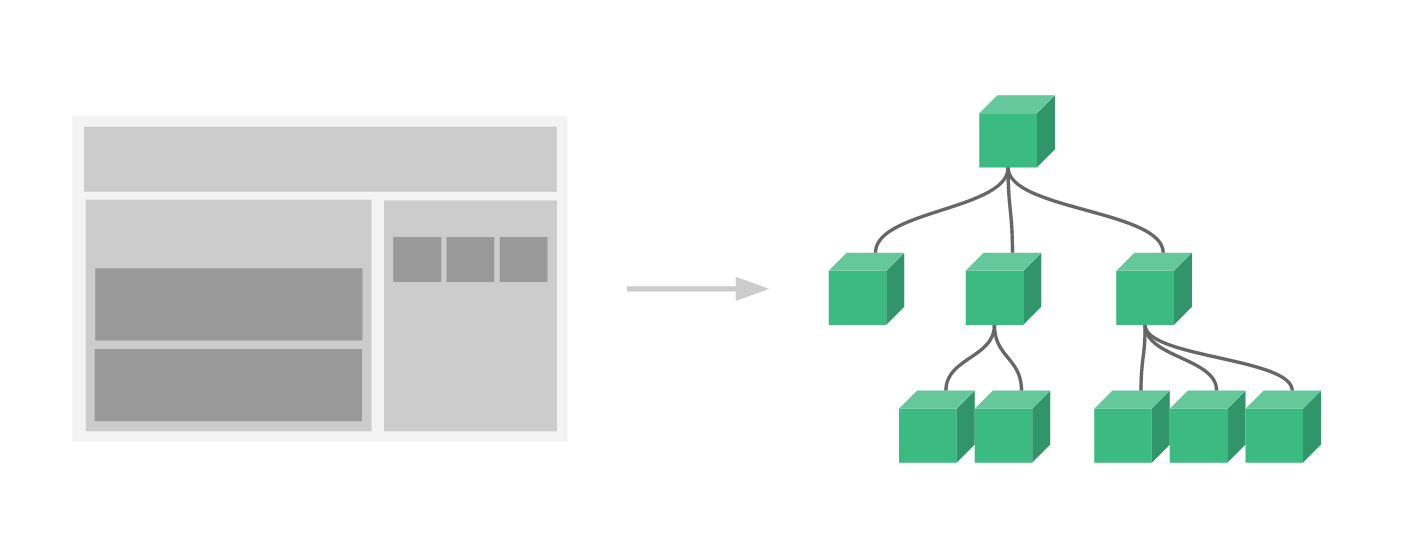
# Vue.js 组件

组件（Component）是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



注册一个全局组件语法格式如下：

Vue.component(tagName, options)

tagName 为组件名，options 为配置选项。注册后，我们可以使用以下方式来调用组件：

<tag-Name></tag-Name>

### **组件名**

### 在注册一个组件的时候，我们始终需要给它一个名字。比如在全局注册的时候我们已经看到了：

### Vue.component('my-component-name', { /\* ... \*/ })

### 该组件名就是 Vue.component 的第一个参数。

### 你给予组件的名字可能依赖于你打算拿它来做什么。当直接在 DOM 中使用一个组件 (而不是在字符串模板或单文件组件) 的时候，我们强烈推荐遵循 W3C 规范中的自定义组件名 (字母全小写且必须包含一个连字符)。这会帮助你避免和当前以及未来的 HTML 元素相冲突。

### **组件名大小写**

### 定义组件名的方式有两种：

### **使用 kebab-case**

### Vue.component('my-component-name', { /\* ... \*/ })

### 当使用 kebab-case (短横线分隔命名) 定义一个组件时，你也必须在引用这个自定义元素时使用 kebab-case，例如 <my-component-name>。

### **使用 PascalCase**

### Vue.component('MyComponentName', { /\* ... \*/ })

### 当使用 PascalCase (驼峰式命名) 定义一个组件时，你在引用这个自定义元素时两种命名法都可以使用。也就是说 <my-component-name> 和 <MyComponentName> 都是可接受的。注意，尽管如此，直接在 DOM (即非字符串的模板) 中使用时只有 kebab-case 是有效的。

Vue 1.0中，在 DOM中直接使用MyComponentName会报错；vue 2.0取消了严格的限制；但是， 为了格式统一，推荐使用kebab-case 模式。

### **全局组件**

所有实例都能用全局组件。

注册一个简单的全局组件 run，并使用它：

<div id="app">

<run></run>

</div>

<script>

// 注册

Vue.component('run', {

template: '<h1>自定义组件!</h1>'

})

// 创建根实例

new Vue({ el: '#app' })

</script>

全局注册往往是不够理想的。比如，如果你使用一个像 webpack 这样的构建系统，全局注册所有的组件意味着即便你已经不再使用一个组件了，它仍然会被包含在你最终的构建结果中。这造成了用户下载的 JavaScript 的无谓的增加。

**局部组件**

在这些情况下，你可以通过一个普通的 JavaScript 对象来定义组件：

var ComponentA = { /\* ... \*/ }

var ComponentB = { /\* ... \*/ }

var ComponentC = { /\* ... \*/ }

然后在 components 选项中定义你想要使用的组件：

new Vue({

el: '#app'

components: {

'component-a': ComponentA,

'component-b': ComponentB

}

})

对于 components 对象中的每个属性来说，其属性名就是自定义元素的名字，其属性值就是这个组件的选项对象。

注意局部注册的组件在其子组件中不可用。例如，如果你希望 ComponentA 在 ComponentB 中可用，则你需要这样写：

var ComponentA = { /\* ... \*/ }

var ComponentB = {

components: {

'component-a': ComponentA

},

// ...

}

## Prop

prop 是父组件用来传递数据的一个自定义属性。

父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"：

## Prop 实例

<div id="app">

<child message="hello!"></child>

</div>

<script>

// 注册

Vue.component('child', {

// 声明 props

props: ['message'],

// 同样也可以在 vm 实例中像 "this.message" 这样使用

template: '<span>{{ message }}</span>'

})

// 创建根实例

new Vue({ el: '#app' })

</script>

### **动态 Prop**

类似于用 v-bind 绑定 HTML 特性到一个表达式，也可以用 v-bind 动态绑定 props 的值到父组件的数据中。每当父组件的数据变化时，该变化也会传导给子组件：

## Prop 实例

<div id="app">

<div>

<input v-model="parentMsg">

<br>

<child v-bind:message="parentMsg"></child>

</div>

</div>

<script>

// 注册

Vue.component('child', {

// 声明 props

props: ['message'],

// 同样也可以在 vm 实例中像 "this.message" 这样使用

template: '<span>{{ message }}</span>'

})

// 创建根实例

new Vue({

el: '#app',

data: { parentMsg: '父组件内容' }

})

</script>

注意: prop 是单向绑定的：当父组件的属性变化时，将传导给子组件，但是不会反过来。

### **Prop 验证**

组件可以为 props 指定验证要求。

prop 是一个对象而不是字符串数组时，它包含验证要求：

Vue.component('example', {

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

//当一个函数返回一个对象时，称之为 工厂函数

propE: {

type: Object,

default: function () {

return { message: 'hello' }

}

},

// 自定义验证函数

propF: {

validator: function (value) {

return value > 10

}

}

}})

# **自定义事件**

父组件是使用 props 传递数据给子组件，但如果子组件要把数据传递回去，就需要使用自定义事件！

$emit(事件名,所传参数)

父组件：

<h3 @click="$emit('toChild',count)"></h3>

<input type="text" @input="$emit('input',$event.target.value)"/>

子组件：

<counter @to-child='count=$event' @input="price=$event"></counter>

跟组件和 prop 不同，事件名不会被用作一个 JavaScript 变量名或属性名，所以就没有理由使用 camelCase 或 PascalCase 了。并且 v-on 事件监听器在 DOM 模板中会被自动转换为全小写 (因为 HTML 是大小写不敏感的)，所以 v-on:myEvent 将会变成 v-on:myevent——导致 myEvent 不可能被监听到。

因此，我们推荐你始终使用 kebab-case 的事件名。

将原生事件绑定到组件

我们引入一个原生标签(div、li、p......)的概念，那么原生事件（存在于标准当中的如'click', 'mouseover'）应该绑定到原生标签上。自定义组件上应该绑定自定义事件。  
如果我们想给自定义组件标签（注意是标签，不是内部元素）绑定原生事件（使原生事件生效于组件template的root element上）怎么办，加.native修饰符。

<base-input v-on:focus.native="onFocus"></base-input>

<p @focus="onFocus"></p>

# **插槽**

插槽内容

Vue 实现了一套内容分发的 API，这套 API 基于当前的 Web Components 规范草案，将 <slot> 元素作为承载分发内容的出口。

它允许你像这样合成组件：

<navigation-link url="/profile">

Your Profile

</navigation-link>

然后你在 <navigation-link> 的模板中可能会写为：

<a

v-bind:href="url"

class="nav-link"

>

<slot></slot>

</a>

当组件渲染的时候，这个 <slot> 元素将会被替换为“Your Profile”。插槽内可以包含任何模板代码，包括 HTML：

<navigation-link url="/profile">

<!-- 添加一个 Font Awesome 图标 -->

<span class="fa fa-user"></span>

Your Profile

</navigation-link>

甚至其它的组件：

<navigation-link url="/profile">

<!-- 添加一个图标的组件 -->

<font-awesome-icon name="user"></font-awesome-icon>

Your Profile

</navigation-link>

如果 <navigation-link> 没有包含一个 <slot> 元素，则任何传入它的内容都会被抛弃。

**具名插槽**

有些时候我们需要多个插槽。例如，一个假设的 <base-layout> 组件多模板如下：

<div class="container">

<header>

<!-- 我们希望把页头放这里 -->

</header>

<main>

<!-- 我们希望把主要内容放这里 -->

</main>

<footer>

<!-- 我们希望把页脚放这里 -->

</footer>

</div>

对于这样的情况，<slot> 元素有一个特殊的特性：name。这个特性可以用来定义额外的插槽：

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

<footer>

<slot name="footer"></slot>

</footer>

</div>

在向具名插槽提供内容的时候，我们可以在一个父组件的 <template> 元素上使用 slot 特性：

<base-layout>

<template slot="header">

<h1>Here might be a page title</h1>

</template>

<p>A paragraph for the main content.</p>

<p>And another one.</p>

<template slot="footer">

<p>Here's some contact info</p>

</template>

</base-layout>

另一种 slot 特性的用法是直接用在一个普通的元素上：

<base-layout>

<h1 slot="header">Here might be a page title</h1>

<p>A paragraph for the main content.</p>

<p>And another one.</p>

<p slot="footer">Here's some contact info</p>

</base-layout>

我们还是可以保留一个未命名插槽，这个插槽是默认插槽，也就是说它会作为所有未匹配到插槽的内容的统一出口。上述两个示例渲染出来的 HTML 都将会是：

<div class="container">

<header>

<h1>Here might be a page title</h1>

</header>

<main>

<p>A paragraph for the main content.</p>

<p>And another one.</p>

</main>

<footer>

<p>Here's some contact info</p>

</footer>

</div>

**插槽的默认内容**

有的时候为插槽提供默认的内容是很有用的。例如，一个 <submit-button> 组件可能希望这个按钮的默认内容是“Submit”，但是同时允许用户覆写为“Save”、“Upload”或别的内容。

你可以在 <slot> 标签内部指定默认的内容来做到这一点。

<button type="submit">

<slot>Submit</slot>

</button>

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。

总结：

使用组件树设计项目，配置文件链接各个组件--命名转换，动态组件

父组件向内传递属性--动态属性

子组件向外发布事件

Slot插槽传递模板--具名slot