**Prop**

该页面假设你已经阅读过了[组件基础](https://cn.vuejs.org/v2/guide/components.html)。如果你还对组件不太了解，推荐你先阅读它。

[**Prop 的大小写 (camelCase vs kebab-case)**](https://cn.vuejs.org/v2/guide/components-props.html#Prop-的大小写-camelCase-vs-kebab-case)

HTML 中的 attribute 名是大小写不敏感的，所以浏览器会把所有大写字符解释为小写字符。这意味着当你使用 DOM 中的模板时，camelCase (驼峰命名法) 的 prop 名需要使用其等价的 kebab-case (短横线分隔命名) 命名：

Vue.component('blog-post', {

// 在 JavaScript 中是 camelCase 的

props: ['postTitle'],

template: '<h3>{{ postTitle }}</h3>'

})

<!-- 在 HTML 中是 kebab-case 的 -->

<blog-post post-title="hello!"></blog-post>

重申一次，如果你使用字符串模板，那么这个限制就不存在了。

[**Prop 类型**](https://cn.vuejs.org/v2/guide/components-props.html#Prop-类型)

到这里，我们只看到了以字符串数组形式列出的 prop：

props: ['title', 'likes', 'isPublished', 'commentIds', 'author']

但是，通常你希望每个 prop 都有指定的值类型。这时，你可以以对象形式列出 prop，这些属性的名称和值分别是 prop 各自的名称和类型：

props: {

title: String,

likes: Number,

isPublished: Boolean,

commentIds: Array,

author: Object,

callback: Function,

contactsPromise: Promise // or any other constructor

}

这不仅为你的组件提供了文档，还会在它们遇到错误的类型时从浏览器的 JavaScript 控制台提示用户。你会在这个页面接下来的部分看到[类型检查和其它 prop 验证](https://cn.vuejs.org/v2/guide/components-props.html" \l "Prop-验证)。

[**传递静态或动态 Prop**](https://cn.vuejs.org/v2/guide/components-props.html#传递静态或动态-Prop)

像这样，你已经知道了可以像这样给 prop 传入一个静态的值：

<blog-post title="My journey with Vue"></blog-post>

你也知道 prop 可以通过 v-bind 动态赋值，例如：

<!-- 动态赋予一个变量的值 -->

<blog-post v-bind:title="post.title"></blog-post>

<!-- 动态赋予一个复杂表达式的值 -->

<blog-post

v-bind:title="post.title + ' by ' + post.author.name"

></blog-post>

在上述两个示例中，我们传入的值都是字符串类型的，但实际上*任何*类型的值都可以传给一个 prop。

[**传入一个数字**](https://cn.vuejs.org/v2/guide/components-props.html#传入一个数字)

<!-- 即便 `42` 是静态的，我们仍然需要 `v-bind` 来告诉 Vue -->

<!-- 这是一个 JavaScript 表达式而不是一个字符串。-->

<blog-post v-bind:likes="42"></blog-post>

<!-- 用一个变量进行动态赋值。-->

<blog-post v-bind:likes="post.likes"></blog-post>

[**传入一个布尔值**](https://cn.vuejs.org/v2/guide/components-props.html#传入一个布尔值)

<!-- 包含该 prop 没有值的情况在内，都意味着 `true`。-->

<blog-post is-published></blog-post>

<!-- 即便 `false` 是静态的，我们仍然需要 `v-bind` 来告诉 Vue -->

<!-- 这是一个 JavaScript 表达式而不是一个字符串。-->

<blog-post v-bind:is-published="false"></blog-post>

<!-- 用一个变量进行动态赋值。-->

<blog-post v-bind:is-published="post.isPublished"></blog-post>

[**传入一个数组**](https://cn.vuejs.org/v2/guide/components-props.html#传入一个数组)

<!-- 即便数组是静态的，我们仍然需要 `v-bind` 来告诉 Vue -->

<!-- 这是一个 JavaScript 表达式而不是一个字符串。-->

<blog-post v-bind:comment-ids="[234, 266, 273]"></blog-post>

<!-- 用一个变量进行动态赋值。-->

<blog-post v-bind:comment-ids="post.commentIds"></blog-post>

[**传入一个对象**](https://cn.vuejs.org/v2/guide/components-props.html#传入一个对象)

<!-- 即便对象是静态的，我们仍然需要 `v-bind` 来告诉 Vue -->

<!-- 这是一个 JavaScript 表达式而不是一个字符串。-->

<blog-post

v-bind:author="{

name: 'Veronica',

company: 'Veridian Dynamics'

}"

></blog-post>

<!-- 用一个变量进行动态赋值。-->

<blog-post v-bind:author="post.author"></blog-post>

[**传入一个对象的所有属性**](https://cn.vuejs.org/v2/guide/components-props.html#传入一个对象的所有属性)

如果你想要将一个对象的所有属性都作为 prop 传入，你可以使用不带参数的 v-bind (取代 v-bind:prop-name)。例如，对于一个给定的对象 post：

post: {

id: 1,

title: 'My Journey with Vue'

}

下面的模板：

<blog-post v-bind="post"></blog-post>

等价于：

<blog-post

v-bind:id="post.id"

v-bind:title="post.title"

></blog-post>

[**单向数据流**](https://cn.vuejs.org/v2/guide/components-props.html#单向数据流)

所有的 prop 都使得其父子 prop 之间形成了一个**单向下行绑定**：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。

额外的，每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。这意味着你**不**应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器的控制台中发出警告。

这里有两种常见的试图改变一个 prop 的情形：

1. **这个 prop 用来传递一个初始值；这个子组件接下来希望将其作为一个本地的 prop 数据来使用。**在这种情况下，最好定义一个本地的 data 属性并将这个 prop 用作其初始值：
2. props: ['initialCounter'],
3. data: function () {
4. return {
5. counter: this.initialCounter
6. }

}

1. **这个 prop 以一种原始的值传入且需要进行转换。**在这种情况下，最好使用这个 prop 的值来定义一个计算属性：
2. props: ['size'],
3. computed: {
4. normalizedSize: function () {
5. return this.size.trim().toLowerCase()
6. }

}

注意在 JavaScript 中对象和数组是通过引用传入的，所以对于一个数组或对象类型的 prop 来说，在子组件中改变这个对象或数组本身**将会**影响到父组件的状态。

[**Prop 验证**](https://cn.vuejs.org/v2/guide/components-props.html#Prop-验证)

我们可以为组件的 prop 指定验证要求，例如你知道的这些类型。如果有一个需求没有被满足，则 Vue 会在浏览器控制台中警告你。这在开发一个会被别人用到的组件时尤其有帮助。

为了定制 prop 的验证方式，你可以为 props 中的值提供一个带有验证需求的对象，而不是一个字符串数组。例如：

Vue.component('my-component', {

props: {

// 基础的类型检查 (`null` 和 `undefined` 会通过任何类型验证)

propA: Number,

// 多个可能的类型

propB: [String, Number],

// 必填的字符串

propC: {

type: String,

required: true

},

// 带有默认值的数字

propD: {

type: Number,

default: 100

},

// 带有默认值的对象

propE: {

type: Object,

// 对象或数组默认值必须从一个工厂函数获取

default: function () {

return { message: 'hello' }

}

},

// 自定义验证函数

propF: {

validator: function (value) {

// 这个值必须匹配下列字符串中的一个

return ['success', 'warning', 'danger'].indexOf(value) !== -1

}

}

}

})

当 prop 验证失败的时候，(开发环境构建版本的) Vue 将会产生一个控制台的警告。

注意那些 prop 会在一个组件实例创建**之前**进行验证，所以实例的属性 (如 data、computed 等) 在 default 或 validator 函数中是不可用的。

[**类型检查**](https://cn.vuejs.org/v2/guide/components-props.html#类型检查)

type 可以是下列原生构造函数中的一个：

* String
* Number
* Boolean
* Array
* Object
* Date
* Function
* Symbol

额外的，type 还可以是一个自定义的构造函数，并且通过 instanceof 来进行检查确认。例如，给定下列现成的构造函数：

function Person (firstName, lastName) {

this.firstName = firstName

this.lastName = lastName

}

你可以使用：

Vue.component('blog-post', {

props: {

author: Person

}

})

来验证 author prop 的值是否是通过 new Person 创建的。

[**非 Prop 的 Attribute**](https://cn.vuejs.org/v2/guide/components-props.html#非-Prop-的-Attribute)

一个非 prop 的 attribute 是指传向一个组件，但是该组件并没有相应 prop 定义的 attribute。

因为显式定义的 prop 适用于向一个子组件传入信息，然而组件库的作者并不总能预见组件会被用于怎样的场景。这也是为什么组件可以接受任意的 attribute，而这些 attribute 会被添加到这个组件的根元素上。

例如，想象一下你通过一个 Bootstrap 插件使用了一个第三方的 <bootstrap-date-input> 组件，这个插件需要在其 <input> 上用到一个 data-date-picker attribute。我们可以将这个 attribute 添加到你的组件实例上：

<bootstrap-date-input data-date-picker="activated"></bootstrap-date-input>

然后这个 data-date-picker="activated" attribute 就会自动添加到 <bootstrap-date-input> 的根元素上。

[**替换/合并已有的 Attribute**](https://cn.vuejs.org/v2/guide/components-props.html#替换-合并已有的-Attribute)

想象一下 <bootstrap-date-input> 的模板是这样的：

<input type="date" class="form-control">

为了给我们的日期选择器插件定制一个主题，我们可能需要像这样添加一个特别的类名：

<bootstrap-date-input

data-date-picker="activated"

class="date-picker-theme-dark"

></bootstrap-date-input>

在这种情况下，我们定义了两个不同的 class 的值：

* form-control，这是在组件的模板内设置好的
* date-picker-theme-dark，这是从组件的父级传入的

对于绝大多数 attribute 来说，从外部提供给组件的值会替换掉组件内部设置好的值。所以如果传入 type="text" 就会替换掉 type="date" 并把它破坏！庆幸的是，class 和 style attribute 会稍微智能一些，即两边的值会被合并起来，从而得到最终的值：form-control date-picker-theme-dark。

[**禁用 Attribute 继承**](https://cn.vuejs.org/v2/guide/components-props.html#禁用-Attribute-继承)

如果你**不**希望组件的根元素继承 attribute，你可以在组件的选项中设置 inheritAttrs: false。例如：

Vue.component('my-component', {

inheritAttrs: false,

// ...

})

这尤其适合配合实例的 $attrs 属性使用，该属性包含了传递给一个组件的 attribute 名和 attribute 值，例如：

{

required: true,

placeholder: 'Enter your username'

}

有了 inheritAttrs: false 和 $attrs，你就可以手动决定这些 attribute 会被赋予哪个元素。在撰写[基础组件](https://cn.vuejs.org/v2/style-guide/" \l "基础组件名-强烈推荐)的时候是常会用到的：

Vue.component('base-input', {

inheritAttrs: false,

props: ['label', 'value'],

template: `

<label>

{{ label }}

<input

v-bind="$attrs"

v-bind:value="value"

v-on:input="$emit('input', $event.target.value)"

>

</label>

`

})

注意 inheritAttrs: false 选项**不会**影响 style 和 class 的绑定。

这个模式允许你在使用基础组件的时候更像是使用原始的 HTML 元素，而不会担心哪个元素是真正的根元素：

<base-input

v-model="username"

required

placeholder="Enter your username"

></base-input>