Vue指令在项目中的应用：

1. v-for 用于以列表形式展示数据；例如：

<div v-for="(value, key, index) in object">

{{ index }}. {{ key }}: {{ value }}

</div>

1. 当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级。
2. v-if 控制页面元素插入、移除，对应表达式为真为插入元素，否则移除元素
3. v-show 控制元素是否显示，表达式为假时为元素设置display:none;
4. V-bind 用于响应式的更新HTML元素的属性；例如：v-bind:class=‘’动态添加class、id等。缩写形式为:class
5. V-on 用于监听DOM事件；

---- 可以触发js代码 <button v-on:click="counter += 1">增加 1</button>

---- 事件名，例如 <!-- `greet` 是在下面定义的方法名 -->

<button v-on:click="greet">Greet</button>

缩写形式为@click

---- 内联传值 <button v-on:click="say('hi')">Say hi</button>

---- 有时也需要在内联语句处理器中访问原生 DOM 事件。可以用特殊变量 $event 把它传入方法：<button v-on:click="warn('Form cannot be submitted yet.', $event)"> Submit</button>

1. 事件修饰符

.stop

.prevent

.capture

.self

.once

<!-- 阻止单击事件冒泡 -->

<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->

<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->

<form v-on:submit.prevent></form>

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当事件在该元素本身（比如不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

<!-- 点击事件将只会触发一次 -->

<a v-on:click.once="doThis"></a>

1. 键值修饰符

.enter

.tab

.delete (捕获 “删除” 和 “退格” 键)

.esc

.space

.up

.down

.left

.right

<!-- 只有在 keyCode 是 13 时调用 vm.submit() -->

<input v-on:keyup.13="submit">

记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

<!-- 同上 -->

<input v-on:keyup.enter="submit">

<!-- 缩写语法 -->

<input @keyup.enter="submit">

1. 可以用如下修饰符开启鼠标或键盘事件监听，使在按键按下时发生响应。

.ctrl

.alt

.shift

.meta

<!-- Alt + C -->

<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

修饰键比正常的按键不同；修饰键和 keyup 事件一起用时，事件引发时必须按下正常的按键。换一种说法：如果要引发 keyup.ctrl，必须按下 ctrl 时释放其他的按键；单单释放 ctrl 不会引发事件。

1. 鼠标按钮修饰符

.left

.right

.middle

Vue页面插入值的方式

1. mustache语法，即 {{ }} 双大括号
2. v-once 一次性插入值，数据动态改变，页面不更新
3. v-html 可插入带有HTML标签的值，相当于js中的append()方法
4. v-model 表单输入绑定

------ 在文本区域插值 (<textarea></textarea>) 并不会生效，应用 v-model 来代替。

------ 单选输入

<input type="radio" id="one" value="One" v-model="picked">

<label for="one">One</label>

<br>

<input type="radio" id="two" value="Two" v-model="picked">

<label for="two">Two</label>

------ 多选输入

<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">

<label for="jack">Jack</label>

<input type="checkbox" id="john" value="John" v-model="checkedNames">

<label for="john">John</label>

------ 单选列表

------ 值绑定

<!-- 当选中时，`picked` 为字符串 "a" -->

<input type="radio" v-model="picked" value="a">

<!-- `toggle` 为 true 或 false -->

<input type="checkbox" v-model="toggle">

<!-- 当选中时，`selected` 为字符串 "abc" -->

<select v-model="selected">

<option value="abc">ABC</option>

</select>

----- 修饰符

1. .lazy 在默认情况下，v-model 在 input 事件中同步输入框的值与数据 (除了 上述 IME 部分)，但你可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步：

<!-- 在 "change" 而不是 "input" 事件中更新 -->

<input v-model.lazy="msg" >

1. .number 如果想自动将用户的输入值转为 Number 类型 (如果原值的转换结果为 NaN 则返回原值)，可以添加一个修饰符 number 给 v-model 来处理输入值：

<input v-model.number="age" type="number">

1. .trim 如果要自动过滤用户输入的首尾空格，可以添加 trim 修饰符到 v-model 上过滤输入：<input v-model.trim="msg">

Computed计算属性

1. 计算属性是基于依赖进行缓存的，只有依赖发生变化才会重新求值
2. 计算属性默认只有 getter ，不过在需要时你也可以提供一个 setter

Vue生命周期钩子

1. new Vue() 创建Vue对象
2. beforeCreate 创建vue实例前的钩子函数；Observe Data 开始监听data对象数据变化情况；Init Events 初始化Vue内部事件
3. Created 实例创建完成之后的钩子函数 **==>** 编译模板：把data对象里面的数据和vue语法写的模板编译成HTML
4. beforeMount 开始挂载编译生成的HTML到对应位置时触发的钩子函数，此时还没有将便移除的html渲染到页面上 **==>** 将编译好的html替换掉el属性所指向的dom对象或替换对应HTML标签里面的内容
5. mounted 将编译好的html挂载到页面完成后执行的钩子函数，此时可以进行发送ajax请求获取火炬的操作进行数据初始化；注意：mounted在整个实例生命内只执行一次
6. 实时监控data数据变化随时去更新dom beforeUpdate和updated
7. beforeDestroy vue实例销毁之前执行的钩子函数 **==>** 销毁、拆除所有观察者，子组件和事件监听
8. Destroyed 实例销毁执行的钩子函数 **==>** 销毁vue实例

绑定内联样式

1. v-bind:style 的对象语法十分直观——看着非常像 CSS，其实它是一个 JavaScript 对象。 CSS 属性名可以用驼峰式 (camelCase) 如下：

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

data: {

activeColor: 'red',

fontSize: 30

}

1. v-bind:style 的数组语法可以将多个样式对象应用到一个元素上：
2. 多重值

|  |
| --- |
| <div :style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div> |

这会渲染数组中最后一个被浏览器支持的值。在这个例子中，如果浏览器支持不带浏览器前缀的 flexbox，那么渲染结果会是 display: flex。

过渡组件

Vue 提供了 transition 的封装组件，在下列情形中，可以给任何元素和组件添加 entering/leaving 过渡

1、自定义过渡类名

enter-class

enter-active-class

enter-to-class (2.1.8+)

leave-class

leave-active-class

leave-to-class (2.1.8+)

他们的优先级高于普通的类名，这对于 Vue 的过渡系统和其他第三方 CSS 动画库，如 [Animate.css](https://daneden.github.io/animate.css/) 结合使用十分有用。

2、用 <transition> 组件上的 duration 属性定制一个显性的过渡持续时间 (以毫秒计)：

|  |
| --- |
| <transition :duration="{ enter: 500, leave: 800 }">...</transition> |

Vue-cli项目中assets和static的区别

## Webpacked Assets

为了回答这个问题，我们首先需要了解Webpack如何处理静态资产。在 \*.vue 组件中，所有模板和CSS都会被 vue-html-loader 及 css-loader 解析，并查找资源URL。例如，在 <img src="./logo.png">  
和 background: url(./logo.png) 中，"./logo.png" 是相对的资源路径，将由****Webpack解析为模块依赖****。

因为 logo.png 不是 JavaScript，当被视为模块依赖时，需要使用 url-loader 和 file-loader  
处理它。vue-cli 的 webpack 脚手架已经配置了这些 loader，因此可以使用相对/模块路径。

由于这些资源可能在构建过程中被内联/复制/重命名，所以它们基本上是源代码的一部分。这就是为什么建议将  
Webpack 处理的静态资源放在 /src 目录中和其它源文件放一起的原因。事实上，甚至不必把它们全部放在 /src/assets：可以用模块/组件的组织方式来使用它们。例如，可以在每个放置组件的目录中存放静态资源。

相比之下，static/ 目录下的文件并不会被 Webpack 处理：它们会直接被复制到最终目录（默认是dist/static）下。必须使用绝对路径引用这些文件，这是通过在 config.js 文件中的 build.assetsPublicPath 和 build.assetsSubDirectory 连接来确定的。

任何放在 static/ 中文件需要以绝对路径的形式引用：/static/[filename]。如果更改 assetSubDirectory 的值为 assets，那么路径需改为 /assets/[filename]。