**Class and Style Bindings**

A common need for data binding is manipulating an element’s class list and its inline styles. Since they are both attributes, we can use v-bind to handle them: we only need to calculate a final string with our expressions. However, meddling with string concatenation is annoying and error-prone. For this reason, Vue provides special enhancements when v-bind is used with class and style. In addition to strings, the expressions can also evaluate to objects or arrays.

**Binding HTML Classes**

**Object Syntax**

We can pass an object to v-bind:class to dynamically toggle classes:

|  |
| --- |
| <div v-bind:class="{ active: isActive }"></div> |

The above syntax means the presence of the active class will be determined by the **truthiness** of the data property isActive.

You can have multiple classes toggled by having more fields in the object. In addition, the v-bind:class directive can also co-exist with the plain class attribute. So given the following template:

|  |
| --- |
| <div class="static"  v-bind:class="{ active: isActive, 'text-danger': hasError }"> </div> |

And the following data:

|  |
| --- |
| data: {  isActive: true,  hasError: false } |

It will render:

|  |
| --- |
| <div class="static active"></div> |

When isActive or hasError changes, the class list will be updated accordingly. For example, if hasError becomes true, the class list will become "static active text-danger".

The bound object doesn’t have to be inline:

|  |
| --- |
| <div v-bind:class="classObject"></div> |

|  |
| --- |
| data: {  classObject: {  active: true,  'text-danger': false  } } |

This will render the same result. We can also bind to a **computed property** that returns an object. This is a common and powerful pattern:

|  |
| --- |
| <div v-bind:class="classObject"></div> |

|  |
| --- |
| data: {  isActive: true,  error: null }, computed: {  classObject: function () {  return {  active: this.isActive && !this.error,  'text-danger': this.error && this.error.type === 'fatal'  }  } } |

**Array Syntax**

We can pass an array to v-bind:class to apply a list of classes:

|  |
| --- |
| <div v-bind:class="[activeClass, errorClass]"></div> |

|  |
| --- |
| data: {  activeClass: 'active',  errorClass: 'text-danger' } |

Which will render:

|  |
| --- |
| <div class="active text-danger"></div> |

If you would like to also toggle a class in the list conditionally, you can do it with a ternary expression:

|  |
| --- |
| <div v-bind:class="[isActive ? activeClass : '', errorClass]"></div> |

This will always apply errorClass, but will only apply activeClass when isActive is truthy.

However, this can be a bit verbose if you have multiple conditional classes. That’s why it’s also possible to use the object syntax inside array syntax:

|  |
| --- |
| <div v-bind:class="[{ active: isActive }, errorClass]"></div> |

**With Components**

**This section assumes knowledge of Vue Components. Feel free to skip it and come back later.**

When you use the class attribute on a custom component, those classes will be added to the component’s root element. Existing classes on this element will not be overwritten.

For example, if you declare this component:

|  |
| --- |
| Vue.component('my-component', {  template: '<p class="foo bar">Hi</p>' }) |

Then add some classes when using it:

|  |
| --- |
| <my-component class="baz boo"></my-component> |

The rendered HTML will be:

|  |
| --- |
| <p class="foo bar baz boo">Hi</p> |

The same is true for class bindings:

|  |
| --- |
| <my-component v-bind:class="{ active: isActive }"></my-component> |

When isActive is truthy, the rendered HTML will be:

|  |
| --- |
| <p class="foo bar active">Hi</p> |

**Binding Inline Styles**

**Object Syntax**

The object syntax for v-bind:style is pretty straightforward - it looks almost like CSS, except it’s a JavaScript object. You can use either camelCase or kebab-case (use quotes with kebab-case) for the CSS property names:

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

|  |
| --- |
| data: {  activeColor: 'red',  fontSize: 30 } |

It is often a good idea to bind to a style object directly so that the template is cleaner:

|  |
| --- |
| <div v-bind:style="styleObject"></div> |

|  |
| --- |
| data: {  styleObject: {  color: 'red',  fontSize: '13px'  } } |

Again, the object syntax is often used in conjunction with computed properties that return objects.

**Array Syntax**

The array syntax for v-bind:style allows you to apply multiple style objects to the same element:

|  |
| --- |
| <div v-bind:style="[baseStyles, overridingStyles]"></div> |

**Auto-prefixing**

When you use a CSS property that requires **vendor prefixes** in v-bind:style, for example transform, Vue will automatically detect and add appropriate prefixes to the applied styles.

**Multiple Values**

**2.3.0+**

Starting in 2.3.0+ you can provide an array of multiple (prefixed) values to a style property, for example:

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

This will only render the last value in the array which the browser supports. In this example, it will render display: flex for browsers that support the unprefixed version of flexbox.