1. 在.html 文件通过如下方式引入 Vue：（引入 Vue、script src=）

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

2. Vue.js 的核心是一个允许采用简洁的模板（template）语法来声明式地将数据渲染进 DOM 的系统：（核心、模板语法）

HTML

```
<div id="app">
  {{ message }}
</div>
```

JavaScript

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

3. 指令带有前缀 v-，以表示它们是 Vue 提供的特殊特性。它们会在渲染的 DOM 上应用特殊的响应式行为：（指令、v-）

HTML

```
<div id="app-2">
  <span v-bind:title="message">
    鼠标悬停几秒钟查看此处动态绑定的提示信息！
  </span>
</div>
```

JavaScript

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: '页面加载于 ' + new Date().toLocaleString()
  }
})
```

在这里，该指令的意思是：“将这个元素节点的 title 特性和 Vue 实例的 message 属性保持一致”。

4. 每个 Vue 应用都是通过用 Vue 函数创建一个新的 Vue 实例开始的：（vue 函数、创建、实例）

```
var vm = new Vue({
```

```
  // 选项
})
```

例如，当一个 Vue 实例被创建时，它向 Vue 的响应式系统中加入了其 data 对象中能找到的所有的属性。当这些属性的值发生改变时，视图将会产生“响应”，即匹配更新为新的值。（响应式系统、值改变、视图响应）

```
// 我们的数据对象
var data = { a: 1 }

// 该对象被加入到一个 Vue 实例中
var vm = new Vue({
  data: data
})

// 获得这个实例上的属性
// 返回源数据中对应的字段
vm.a == data.a // => true

// 设置属性也会影响到原始数据
vm.a = 2
data.a // => 2

// ……反之亦然
data.a = 3
vm.a // => 3
```

5. 使用 Object.freeze()阻止修改现有的属性，也意味着响应系统无法再追踪变化。（freeze、阻止修改）

HTML

```
<div id="app">
  <p>{{ foo }}</p>
  <!-- 这里的 `foo` 不会更新！ -->
  <button @click="foo = 'baz'">Change it</button>
</div>
```

JavaScript

```
var obj = {
  foo: 'bar'
}

Object.freeze(obj)

new Vue({
  el: '#app',
  data: obj
```

```
})
```

6. 实例属性与方法都有前缀 $，以便与用户定义的属性区分开来。实例属性与方法是 vue 自带的属性。（实例属性、方法、前缀$）

```
var data = { a: 1 }
var vm = new Vue({
  el: '#example',
  data: data
})

vm.$data === data // => true
vm.$el === document.getElementById('example') // => true

// $watch 是一个实例方法
vm.$watch('a', function (newValue, oldValue) {
  // 这个回调将在 `vm.a` 改变后调用
})
```

7. 生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。（生命周期、不同阶段、添加代码）

```
new Vue({
  data: {
    a: 1
  },
  created: function () {
    // `this` 指向 vm 实例
    console.log('a is: ' + this.a)
  }
})
```

也有一些其它的钩子，在实例生命周期的不同阶段被调用，如 mounted、updated 和 destroyed。生命周期钩子的 this 上下文指向调用它的 Vue 实例。

8. 所有 Vue.js 的模板都是合法的 HTML，所以能被遵循规范的浏览器和 HTML 解析器解析。（模板、合法的 HTML）

9. Vue.js 模板语法：

(1) 数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值：（数据绑定、双大括号）

```
<div id="app">
  <p>{{ message }}</p>
</div>
```

通过使用 v-once 指令，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。但请留心这会影响到该节点上的其它数据绑定：（v_once 指令、一次性插值、不更新）

```
<span v-once>这个将不会改变: {{ msg }}</span>
```

(2) v-html 指令可以输出真正的 HTML：（v-html、输出 html）

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>
</head>
<body>
<div id="app">
    <div v-html="message"></div>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    message: '<h1>菜鸟教程</h1>'
  }
})
</script>
</body>
</html>
```

输出：

```
菜鸟教程
```

(3) 对于所有的数据绑定，Vue.js 都提供了完全的 JavaScript 表达式支持。也就是说，可以在数据绑定里编写任何 javascript 表达式。（数据绑定、表达式支持）

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

唯一有限制的是，每个绑定都只能包含单个表达式，所以下面的例子都不会生效。（只能单个表达式）

```
<!-- 这是语句，不是表达式 -->
{{ var a = 1 }}
```

```
<!-- 流控制也不会生效，请使用三元表达式 -->
{{ if (ok) { return message } }}
```

(4) 指令是带有 v- 前缀的特殊属性。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。（指令、改变、连带影响）

```
<p v-if="seen">现在你看到我了</p>
```

这里，v-if 指令将根据表达式 seen 的值的真假来插入/移除 <p> 元素。

(5) v-bind 用于动态地绑定一个或多个特性，或一个组件 prop 到表达式。（v-bind、动态地绑定）

(6) 一些指令能够接收一个“参数”，在指令名称之后以冒号表示。例如，v-bind 指令可以用于响应式地更新 HTML 属性：（指令参数、冒号表示）

```
<a v-bind:href="url">...</a>
```

在这里 href 是参数，告知 v-bind 指令将该元素的 href 属性与表达式 url 的值绑定。

(7) v-on 指令用于监听 DOM 事件：（v-on、监听事件）

```
<a v-on:click="doSomething">...</a>
```

(8) 修饰符 (Modifiers) 是以半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。例如，.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()：（修改符、句号后缀、特殊绑定）

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

(9) Vue.js 为 v-bind 和 v-on 这两个最常用的指令，提供了特定简写：

- v-bind 缩写为冒号：（v-bind、缩写、冒号）

```
<!-- 完整语法 -->
<a v-bind:href="url">...</a>

<!-- 缩写 -->
<a :href="url">...</a>
```

- v-on 缩写为@：（v-on、缩写、@）

```
<!-- 完整语法 -->
<a v-on:click="doSomething">...</a>

<!-- 缩写 -->
```

```
<a @click="doSomething">...</a>
```

: 与 @ 对于特性名来说都是合法字符，在所有支持 Vue.js 的浏览器都能被正确地解析。而且，它们不会出现在最终渲染的标记中。

10. 单页面应用，简称 SPA，就是只有一张 Web 页面的应用，是加载单个 HTML 页面并在用户与应用程序交互时动态更新该页面的 Web 应用程序。单页面应用主要优势是减少资源重复请求，页面切换快，但不利于 SEO，不利于业务划分。（单页面应用、减少重复请求、不利于 SEO）

多页面应用跳转时刷新所有资源，每个公共资源(js、css 等)需选择性重新加载。（多页面、需要重新加载）

11. 计算属性可以用来处理任何复杂逻辑，使用 computed 关键字定义计算属性：（computed、定义计算属性）

HTML

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

JavaScript

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // 计算属性的 getter
    reversedMessage: function () {
      // `this` 指向 vm 实例
      return this.message.split('').reverse().join('')
    }
  }
})
```

这里实际就相当于一个函数。

12. 我们可以使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值，也就是相关变量不变，就不会重新取值。而使用 methods ，在页面重新渲染的时候，函数总会重新调用执行。（computed、依赖改变、重新取值、methods、重新渲染、重新执行）

13. 值得注意的是，由于 computed 是基于依赖缓存，这也意味着下面的计算属性将不再更新，因为 Date.now() 不是响应式依赖：（Date.now()、非响应）

```
computed: {
```

```
    now: function () {
        return Date.now()
    }
}
```

14. 计算属性的 getter 用于处理逻辑，获取属性，而 setter 用于设置依赖的变量值。默认只提供 getter，不过在需要时你也可以提供一个 setter ：（getter、获取、setter、设置）

```
var vm = new Vue({
    el: '#app',
    data: {
        name: 'Google',
        url: 'http://www.google.com'
    },
    computed: {
        site: {
            // getter
            get: function () {
                return this.name + ' ' + this.url
            },
            // setter
            set: function (newValue) {
                var names = newValue.split(' ')
                this.name = names[0]
                this.url = names[names.length - 1]
            }
        }
    }
})
// 调用 setter， vm.name 和 vm.url 也会被对应更新
vm.site = '菜鸟教程 http://www.runoob.com';
document.write('name: ' + vm.name);
document.write('<br>');
document.write('url: ' + vm.url);
```

15. 侦听属性用于观察和响应 Vue 实例上的数据变动，使用 watch 实现：（侦听属性、观察、数据变动、watch）

```
<div id = "computed_props">
    千米 : <input type = "text" v-model = "kilometers">
    米 : <input type = "text" v-model = "meters">
</div>
<p id="info"></p>
<script type = "text/javascript">
```

```
    var vm = new Vue({
    el: '#computed_props',
    data: {
        kilometers : 0,
        meters:0
    },
    methods: {
    },
    computed :{
    },
    watch : {
        kilometers:function(val) {
            this.kilometers = val;
            this.meters = val * 1000;
        },
        meters : function (val) {
            this.kilometers = val/ 1000;
            this.meters = val;
        }
    }
    });
    // $watch 是一个实例方法
    vm.$watch('kilometers', function (newValue, oldValue) {
    // 这个回调将在  vm.kilometers 改变后调用
    document.getElementById ("info").innerHTML = "修改前值为: " + oldValue + "，修改后值为: " + newValue;
})
</script>
```

输出结果如下：

千米 : 0 米 : 0

修改前值为: 00，修改后值为: 0

在输入框输入内容后，另一个输入框会自动更新。

16. class 与 style 是 HTML 元素的属性，用于设置元素的样式，我们可以用 v-bind 来设置样式属性。
17. 我们可以为 v-bind:class 设置一个对象，从而动态的切换 class：（v-bind:class、设置对象、切换）

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

上面的语法表示 active 这个 class 存在与否将取决于数据属性 isActive 。（active 、取决于、isActive ）

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>
<style>
.active {
        width: 100px;
        height: 100px;
        background: green;
}
</style>
</head>
<body>
<div id="app">
  <div v-bind:class="{ active: isActive }"></div>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    isActive: true
  }
})
</script>
</body>
</html>
```

实例中将 isActive 设置为 true 显示了一个绿色的 div 块，如果设置为 false 则不显示：

18. 可以在对象中传入更多属性来动态切换多个 class。此外，v-bind:class 指令也可以与普通的 class 属性共存。当有如下模板：（更多属性、普通 class、共存）

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

和如下 data：

```
data: {
  isActive: true,
  hasError: false
}
```

结果渲染为：

```
<div class="static active"></div>
```

当 isActive 或者 hasError 变化时，class 列表将相应地更新。例如，如果 hasError 的值为 true，class 列表将变为 "static active text-danger"。

19. 可以把一个数组传给 v-bind:class，以应用一个 class 列表：（数组、传给、v-bind）

HTML

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

JavaScript

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

数组中的值会相应地替换为数组对应的值。这里渲染为：

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

我们还可以使用三元表达式来切换列表中的 class：（三元表达式、切换）

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

20. v-bind:class 也可以直接绑定数据里的一个对象：（v-bind:class、绑定对象）

```
<html>
<head>
<meta charset="utf-8">
<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>
<style>
.active {
        width: 100px;
        height: 100px;
        background: green;
}
.text-danger {
        background: red;
}
</style>
</head>
<body>
<div id="app">
   <div v-bind:class="classObject"></div>
</div>

<script>
new Vue({
```

```
   el: '#app',
   data: {
      classObject: {
         active: true,
         'text-danger': true
      }
   }
})
</script>
</body>
</html>
```

21. v-bind:class 也可以在这里绑定返回对象的计算属性。这是一个常用且强大的模式：（v-bind:class、返回计算属性）

```
<html>
<head>
<meta charset="utf-8">
<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>
<style>
.active {
        width: 100px;
        height: 100px;
        background: green;
}
.text-danger {
        background: red;
}
</style>
</head>
<body>
<div id="app">
   <div v-bind:class="classObject"></div>
</div>

<script>
new Vue({
   el: '#app',
   data: {
   isActive: true,
   error: null
   },
   computed: {
      classObject: function () {
         return {
```

```
            active: this.isActive && !this.error,
            'text-danger': this.error && this.error.type === 'fatal',
        }
      }
   }
})
</script>
</body>
</html>
```

22. 我们可以在 v-bind:style 直接设置样式：（v-bind:style、设置样式）

```
<div id="app">
      <div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">菜鸟教程</div>
</div>
```

注意，第二个 fontSize 值为 30。以上实例 div style 为：

```
<div style="color: green; font-size: 30px;">菜鸟教程</div>
```

也可以直接绑定到一个样式对象，让模板更清晰：（绑定到对象）

```
<div id="app">
   <div v-bind:style="styleObject">菜鸟教程</div>
</div>

<script>
new Vue({
   el: '#app',
   data: {
      styleObject: {
         color: 'green',
         fontSize: '30px'
      }
   }
})
</script>
```

23. v-bind:style 可以使用数组将多个样式对象应用到一个元素上：（v-bind:style、使用数组）

```
<div id="app">
   <div v-bind:style="[baseStyles, overridingStyles]">菜鸟教程</div>
</div>

<script>
new Vue({
```

```
    el: '#app',
    data: {
        baseStyles: {
            color: 'green',
            fontSize: '30px'
        },
            overridingStyles: {
            'font-weight': 'bold'
        }
    }
})
</script>
```

24. 条件判断使用 v-if 指令：（条件判断、v-if）

```
<div id="app">
        <p v-if="seen">现在你看到我了</p>
        <template v-if="ok">
            <h1>菜鸟教程</h1>
            <p>学的不仅是技术，更是梦想！</p>
            <p>哈哈哈，打字辛苦啊！！！</p>
        </template>
</div>

<script>
new Vue({
    el: '#app',
    data: {
        seen: true,
        ok: true
    }
})
</script>
```

这里， v-if 指令将根据表达式 seen 的值(true 或 false)来决定是否插入 p 元素。

25. 可以用 v-else 指令给 v-if 添加一个 "else" 块：（v-else、else 块）

```
<div id="app">
        <div v-if="Math.random() > 0.5">
            Sorry
        </div>
        <div v-else>
            Not sorry
        </div>
</div>
```

```
<script>
new Vue({
   el: '#app'
})
</script>
```

26. v-else-if 在 2.1.0 新增，顾名思义，用作 v-if 的 else-if 块。（v-else-if）

27. 我们也可以使用 v-show 指令来根据条件展示元素：（v-show、展示元素）

```
<h1 v-show="ok">Hello!</h1>
```

带有 v-show 的元素始终会被渲染并保留在 DOM 中。（v-show、始终、保留在）

28. Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染：（高效地、复用已有）

```
<template v-if="loginType === 'username'">
   <label>Username</label>
   <input placeholder="Enter your username">
</template>
<template v-else>
   <label>Email</label>
   <input placeholder="Enter your email address">
</template>
```

由于两个模板使用了相同的元素，<input> 不会被替换掉，仅仅是替换了它的 placeholder，那么在上面的代码中切换 loginType 将不会清除用户已经输入的内容。

- 添加一个具有唯一值的 key 属性，来表达“这两个元素是完全独立的，不要复用它们”：（添加 key、不复用）

```
<template v-if="loginType === 'username'">
   <label>Username</label>
   <input placeholder="Enter your username" key="username-input">
</template>
<template v-else>
   <label>Email</label>
   <input placeholder="Enter your email address" key="email-input">
</template>
```

现在，每次切换时，输入框都将被重新渲染。

29. v-show 不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。（v-show、总会被渲染）

30. 循环使用 v-for 指令。v-for 指令需要以 site in sites 形式的特殊语法， sites 是源数据数组并且 site 是数组元素迭代的别名。（循环、v-for、site in sites）

```
<div id="app">
   <ol>
      <li v-for="site in sites">
         {{ site.name }}
      </li>
   </ol>
</div>

<script>
new Vue({
   el: '#app',
   data: {
      sites: [
         { name: 'Runoob' },
         { name: 'Google' },
         { name: 'Taobao' }
      ]
   }
})
</script>
```

31. 模板中使用 v-for：（模板中、v-for）

```
<ul>
   <template v-for="site in sites">
      <li>{{ site.name }}</li>
      <li>----------</li>
   </template>
</ul>
```

32. v-for 可以通过一个对象的属性来迭代数据：（v-for、对象、迭代）

```
<div id="app">
   <ul>
      <li v-for="value in object">
      {{ value }}
      </li>
   </ul>
</div>

<script>
new Vue({
   el: '#app',
   data: {
      object: {
```

```
        name: '菜鸟教程',
        url: 'http://www.runoob.com',
        slogan: '学的不仅是技术，更是梦想！'
      }
  }
})
</script>
```

输出结果：

- 菜鸟教程
- http://www.runoob.com
- 学的不仅是技术，更是梦想！

33. 也可以为 v-for 提供第二个的参数为键名：（v-for 、第二个参数、键名）

```
<div id="app">
  <ul>
    <li v-for="(value, key) in object">
    {{ key }} : {{ value }}
    </li>
  </ul>
</div>
```

34. v-for 也可以循环整数：（v-for 、循环整数）

```
<div id="app">
  <ul>
    <li v-for="n in 10">
     {{ n }}
    </li>
  </ul>
</div>
```

35. 由于 JavaScript 的限制，Vue 不能检测以下变动的数组：（不能检测、变动的数组）

- 当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue
- 当你修改数组的长度时，例如：vm.items.length = newLength

```
var vm = new Vue({
  data: {
    items: ['a', 'b', 'c']
  }
})
vm.items[1] = 'x' // 不是响应性的
```

```
vm.items.length = 2 // 不是响应性的
```

36. vm.$set 是全局方法 Vue.set 的一个别名。该方法用于设置对象的属性。如果对象是响应式的，确保属性被创建后也是响应式的，同时触发视图更新。这个方法主要用于避开 Vue 不能检测属性被添加的限制。（set、设置对象、创建属性）

```
vm.$set(vm.items, indexOfItem, newValue)
```

37. 使用 splice 可以修改数组长度：（splice、修改数组长度）

```
vm.items.splice(newLength)
```

38. 由于 JavaScript 的限制，Vue 不能检测对象属性的添加或删除：（不能检测、属性、添加或删除）

```
var vm = new Vue({
  data: {
    a: 1
  }
})
// `vm.a` 现在是响应式的

vm.b = 2
// `vm.b` 不是响应式的
```

39. 可以使用 Vue.set(object, key, value) 方法向嵌套对象添加响应式属性，例如：（set、添加、响应式属性）

```
var vm = new Vue({
  data: {
    userProfile: {
      name: 'Anika'
    }
  }
})
```

你可以添加一个新的 age 属性到嵌套的 userProfile 对象：

```
Vue.set(vm.userProfile, 'age', 27)
```

40. v-for 和 v-if 处于同一节点时，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。（v-for、优先级高于 v-if）

```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```

41. 用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。（v-on、监听事件）

HTML

```
<div id="example-1">
  <button v-on:click="counter += 1">Add 1</button>
  <p>The button above has been clicked {{ counter }} times.</p>
</div>
```

JavaScript

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

42. v-on 还可以接收一个需要调用的方法名称。（v-on、接收、要调用的方法）

HTML

```
<div id="example-2">
  <!-- `greet` 是在下面定义的方法名 -->
  <button v-on:click="greet">Greet</button>
</div>
```

JavaScript

```
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // 在 `methods` 对象中定义方法
  methods: {
    greet: function (event) {
      // `this` 在方法里指向当前 Vue 实例
      alert('Hello ' + this.name + '!')
      // `event` 是原生 DOM 事件
      if (event) {
        alert(event.target.tagName)
      }
    }
```

```
  }
})

// 也可以用 JavaScript 直接调用方法
example2.greet() // => 'Hello Vue.js!'
```

43. 可以在内联 JavaScript 语句中调用 v-on：（内联的 JavaScript、调用 v-on）

HTML

```
<div id="example-3">
  <button v-on:click="say('hi')">Say hi</button>
  <button v-on:click="say('what')">Say what</button>
</div>
```

JavaScript

```
new Vue({
  el: '#example-3',
  methods: {
    say: function (message) {
      alert(message)
    }
  }
})
```

44. 有时也需要在内联语句处理器中访问原始的 DOM 事件。可以用特殊变量 $event 把它传入方法：（原始事件、$event）

HTML

```
<button v-on:click="warn('Form cannot be submitted yet.', $event)">
  Submit
</button>
```

JavaScript

```
// ...
methods: {
  warn: function (message, event) {
    // 现在我们可以访问原生事件对象
    if (event) event.preventDefault()
    alert(message)
  }
}
```

45. 在事件处理程序中调用 event.preventDefault() 或 event.stopPropagation() 是非常常见的需求。注意，这两个是 JavaScript 里面的函数，不是 vue 的。

46. Vue.js 为 v-on 提供了事件修饰符：（事件修饰符、stop、prevent、capture、self、once）

- .stop
- .prevent
- .capture
- .self
- .once

```
<!-- 阻止单击事件继续传播 -->
<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>

<!-- 添加事件监听器时使用事件捕获模式 -->
<!-- 即元素自身触发的事件先在此处处理，然后才交由内部元素进行处理 -->
<div v-on:click.capture="doThis">...</div>

<!-- 只当在 event.target 是当前元素自身时触发处理函数 -->
<!-- 即事件不是从内部元素触发的 -->
<div v-on:click.self="doThat">...</div>
```

47. Vue 允许为 v-on 在监听键盘事件时添加按键修饰符：（v-on、按键修饰符）

```
<!-- 只有在 `keyCode` 是 13 时调用 `vm.submit()` -->
<input v-on:keyup.13="submit">
```

48. 可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器：（修饰符、按下相应键、触发）

- .ctrl
- .alt
- .shift
- .meta

```
<!-- Alt + C -->
```

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

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

49. 你可以用 v-model 指令在表单 <input> 及 <textarea> 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。（v-model、双向数据绑定）

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

- v-model 会忽略所有表单元素的 value、checked、selected 特性的初始值而总是将 Vue 实例的数据作为数据来源。（忽略、表单元素）

50. 组件可以扩展 HTML 元素，封装可重用的代码。使用 Vue.component(tagName, options) 注册一个全局组件：（组件、component、注册全局组件）

```
Vue.component('my-component', {
  // 选项
})
```

组件在注册之后，便可以作为自定义元素 <my-component></my-component> 在一个实例的模板中使用：（自定义元素、实例中使用）

```
<div id="example">
  <my-component></my-component>
</div>
```

51. 通过某个 Vue 实例/组件的实例选项 components 注册可以注册局部组件：（实例选项 components、注册局部组件）

```
var Child = {
  template: '<div>A custom component!</div>'
}

new Vue({
  // ...
  components: {
    // <my-component> 将只在父组件模板中可用
    'my-component': Child
  }
})
```

52. 、<ol>、<table>、<select> 这样的元素里允许包含的元素有限制，而另一些像 <option> 这样的元素只能出现在某些特定元素的内部。在自定义组件中使用这些受限制的元素时会导致一些问题：（自定义组件、使用受限制元素、出问题）

```
<table>
  <my-row>...</my-row>
</table>
```

自定义组件 `<my-row>` 会被当作无效的内容，因此会导致错误的渲染结果。变通的方案是使用特殊的 `is` 特性：（变通、使用 is 特性）

```
<table>
  <tr is="my-row"></tr>
</table>
```

53. 构造 Vue 实例时传入的各种选项大多数都可以在组件里使用。只有一个例外：data 必须是函数。（data、必须是函数）

Html

```
<div id="example-2">
  <simple-counter></simple-counter>
  <simple-counter></simple-counter>
  <simple-counter></simple-counter>
</div>
```

JavaScript

```
var data = { counter: 0 }

Vue.component('simple-counter', {
  template: '<button v-on:click="counter += 1">{{ counter }}</button>',
  // 技术上 data 的确是一个函数了，因此 Vue 不会警告，
  // 但是我们却给每个组件实例返回了同一个对象的引用
  data: function () {
    return data
  }
})

new Vue({
  el: '#example-2'
})
```

由于这三个组件实例共享了同一个 data 对象，因此递增一个 counter 会影响所有组件！这就错了。我们可以通过为每个组件返回全新的数据对象来修复这个问题：（同一个 data 对象、修复、每个组件返回）

```
data: function () {
  return {
```

```
    counter: 0
  }
}
```

54. 父组件的数据可以通过 prop 才能下发到子组件中。子组件要显式地用 props 选项声明它想要的数据：（父组件、prop、下发子组件）

```
Vue.component('child', {
  // 声明 props
  props: ['message'],
  // 就像 data 一样，prop 也可以在模板中使用
  // 同样也可以在 vm 实例中通过 this.message 来使用
  template: '<span>{{ message }}</span>'
})
```

55. 与绑定到任何普通的 HTML 特性相类似，我们可以用 v-bind 来动态地将 prop 绑定到父组件的数据。每当父组件的数据变化时，该变化也会传导给子组件：（v-bind、prop、绑定到父组件）

HTML

```
<div id="prop-example-2">
  <input v-model="parentMsg">
  <br>
  <child v-bind:my-message="parentMsg"></child>
</div>
```

JavaScript

```
new Vue({
  el: '#prop-example-2',
  data: {
    parentMsg: 'Message from parent'
  }
})
```

56. 如果你想把一个对象的所有属性作为 prop 进行传递，可以使用不带任何参数的 v-bind，即用 v-bind 而不是 v-bind:prop-name。例如，已知一个 todo 对象：（对象、传递、不带参数的 v-bind）

```
todo: {
  text: 'Learn Vue',
  isComplete: false
}
```

然后：

```
<todo-item v-bind="todo"></todo-item>
```

57. 使用字面量语法传递数值，它的值是字符串 ″1″ 而不是一个数值。（字面量语法、字符串）

```
<!-- 传递了一个字符串 "1" -->
<comp some-prop="1"></comp>
```

如果想传递一个真正的 JavaScript 数值，则需要使用 v-bind，从而让它的值被当作 JavaScript 表达式计算：（数值、使用 v-bind）

```
<!-- 传递真正的数值 -->
<comp v-bind:some-prop="1"></comp>
```

58. 用对象的形式来定义 prop，为组件的 prop 指定验证规则。如果传入的数据不符合要求，Vue 会发出警告。（对象的形式、指定验证规则）

```
Vue.component('example', {
  props: {
    // 基础类型检测 (`null` 指允许任何类型)
    propA: Number,
    // 可能是多种类型
    propB: [String, Number],
    // 必传且是字符串
    propC: {
      type: String,
      required: true
    },
    // 数值且有默认值
    propD: {
      type: Number,
      default: 100
    },
    // 数组/对象的默认值应当由一个工厂函数返回
    propE: {
      type: Object,
      default: function () {
        return { message: 'hello' }
      }
    },
    // 自定义验证函数
    propF: {
      validator: function (value) {
        return value > 10
      }
    }
  }
```

```
})
```

59. 每个 Vue 实例都实现了事件接口，即：

- 使用 $on(eventName) 监听事件 （$on(eventName)、监听事件）
- 使用 $emit(eventName, optionalPayload) 触发事件 （$emit、触发事件）

60. 父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件。（v-on、监听子组件）

HTML

```
<div id="counter-event-example">
  <p>{{ total }}</p>
  <button-counter v-on:increment="incrementTotal"></button-counter>
  <button-counter v-on:increment="incrementTotal"></button-counter>
</div>
```

JavaScript

```
Vue.component('button-counter', {
  template: '<button v-on:click="incrementCounter">{{ counter }}</button>',
  data: function () {
    return {
      counter: 0
    }
  },
  methods: {
    incrementCounter: function () {
      this.counter += 1
      this.$emit('increment')
    }
  },
})

new Vue({
  el: '#counter-event-example',
  data: {
    total: 0
  },
  methods: {
    incrementTotal: function () {
      this.total += 1
    }
  }
})
```

61. 使用 v-on 的修饰符 .native 可以在某个组件的根元素上监听一个原生事件。（.native、监听原生事件）

```
<my-component v-on:click.native="doTheThing"></my-component>
```

62. 使用组件时常常会有组件组合使用的情况，如下：

```
<componentA>
<componentB></componentB>
<componentC></componentC>
</componentA>
```

直接套用组件的话，父级组件会将子级组件覆盖掉，不能实现需求的效果。使用 slots 来进行内容分发可以解决这个问题。（覆盖掉、slots、内容分发）

63. 父组件模板的内容在父组件作用域内编译；子组件模板的内容在子组件作用域内编译。

64. 当子组件模板只有一个没有属性的 slots 时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。假定 my-component 组件有如下模板：（没有属性、slots、整个内容、插入）

```
<div>
  <h2>我是子组件的标题</h2>
  <slot>
    只有在没有要分发的内容时才会显示。
  </slot>
</div>
```

父组件模板：

```
<div>
  <h1>我是父组件的标题</h1>
  <my-component>
    <p>这是一些初始内容</p>
    <p>这是更多的初始内容</p>
  </my-component>
</div>
```

渲染结果：

```
<div>
  <h1>我是父组件的标题</h1>
  <div>
    <h2>我是子组件的标题</h2>
    <p>这是一些初始内容</p>
    <p>这是更多的初始内容</p>
  </div>
</div>
```

65. 〈slot〉元素可以用一个特殊的特性 name 来进一步配置如何分发内容。多个 slot 可以有不同的名字。具名 slot 将匹配内容片段中有对应 slot 特性的元素。（name 属性、如何分发内容）

app-layout

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

父组件

```
<app-layout>
  <h1 slot="header">这里可能是一个页面标题</h1>

  <p>主要内容的一个段落。</p>
  <p>另一个主要段落。</p>

  <p slot="footer">这里有一些联系信息</p>
</app-layout>
```

渲染结果：

```
<div class="container">
  <header>
    <h1>这里可能是一个页面标题</h1>
  </header>
  <main>
    <p>主要内容的一个段落。</p>
    <p>另一个主要段落。</p>
  </main>
  <footer>
    <p>这里有一些联系信息</p>
  </footer>
</div>
```

66. 通过使用保留的 <component> 元素，并对其 is 特性进行动态绑定，你可以在同一个挂载点动态切换多个组件：（is 特性、动态切换）

HTML

```
<component v-bind:is="currentView">
  <!-- 组件在 vm.currentview 变化时改变！ -->
</component>
```

JavaScript

```
var vm = new Vue({
  el: '#example',
  data: {
    currentView: 'home'
  },
  components: {
    home: { /* ... */ },
    posts: { /* ... */ },
    archive: { /* ... */ }
  }
})
```

67. 如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个 keep-alive 指令参数：（keep-alive、保留在内存）

```
<keep-alive>
  <component :is="currentView">
    <!-- 非活动组件将被缓存！ -->
  </component>
</keep-alive>
```

68. 尽管有 prop 和事件，但是有时仍然需要在 JavaScript 中直接访问子组件。为此可以使用 ref 为子组件指定一个引用 ID。例如：（JavaScript 中访问、访问子组件、ref）

HTML

```
div id="parent">
  <user-profile ref="profile"></user-profile>
</div>
```

JavaScript

```
var parent = new Vue({ el: '#parent' })
// 访问子组件实例
var child = parent.$refs.profile
```

69. Vue.js 允许将组件定义为一个工厂函数，异步地解析组件的定义。Vue.js 只在组件需要渲染时触发工厂函数，并且把结果缓存起来，用于后面的再次渲染。例如：（工厂函数）

```
Vue.component('async-example', function (resolve, reject) {
  setTimeout(function () {
    // 将组件定义传入 resolve 回调函数
```

```
    resolve({
      template: '<div>I am async!</div>'
    })
  }, 1000)
})
```

工厂函数接收一个 resolve 回调，在收到从服务器下载的组件定义时调用。

70. 当组件中包含大量静态内容时，可以考虑使用 v-once 将渲染结果缓存起来，就像这样：（静态内容、v-once、缓存起来）

```
Vue.component('terms-of-service', {
  template: '\
    <div v-once>\
      <h1>Terms of Service</h1>\
      ...很多静态内容...\
    </div>\
  '
})
```

71. 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 动画过渡：（transition、动画过渡）

- 条件渲染 (使用 v-if)
- 条件展示 (使用 v-show)
- 动态组件
- 组件根节点

72. 混入 (mixins) 是一种分发 Vue 组件中可复用功能的非常灵活的方式。使用 mixins 来定义混入：（混入、分发内容、minxins、定义）

```
// 定义一个混入对象
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// 定义一个使用混入对象的组件
var Component = Vue.extend({
```

```
  mixins: [myMixin]
})

var component = new Component() // => "hello from mixin!"
```

73. 使用 directive 来自定义指令，例如， 注册一个全局自定义指令 `v-focus`：（directive、自定义指令）

```
Vue.directive('focus', {
  // 当被绑定的元素插入到 DOM 中时……
  inserted: function (el) {
    // 聚焦元素
    el.focus()
  }
})
```

74. 如果想注册局部指令，组件中也接受一个 directives 的选项：（directives、局部指令）

```
directives: {
  focus: {
    // 指令的定义
    inserted: function (el) {
      el.focus()
    }
  }
}
```

75. Vue.js 路由允许我们通过不同的 URL 访问不同的内容。通过 Vue.js 可以实现多视图的单页 Web 应用（single page web application，SPA）。Vue.js 路由需要载入 vue-router 库。（路由、URL、不同的内容）

76. Vue.js + vue-router 可以很简单的实现单页应用。（vue、vue-router、单页应用）