1. Vue组件都是一个Vue实例，并且接受相同的选项对象 (一些根实例特有的选项除外)

2. **生命周期的钩子的this上下文指向调用他的Vue实例**，所以不要在选项属性或回调函数上使用箭头函数，比如 `created:()=>{}` ,这样会导致报错

3. 第三方类库会提供很多常用的方法，比如Loadash库，可以提供debounce Ajax请求防抖的功能

4. 当在需要在数据变化时执行异步或者开销比较大的操作的时候，可以选择watch,比如实时请求的时候就可以用watch来侦听绑定的数据

5. 当在一个自定义组件上使用 class 属性时，这些类将被添加到该组件的根元素上面。这个元素上已经存在的类不会被覆盖。

6. 由于Vue会高效的复用元素，但是有时候必须要区分成两个的时候，可以用key来标识，表示这是一个唯一的东西

7.

8. 一些变异的数组操作将会触发Vue的视图更新。

变异方法

Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。这些方法如下：

- `push()`
- `pop()`
- `shift()`
- `unshift()`
- `splice()`
- `sort()`
- `reverse()`

你打开控制台，然后用前面例子的 `items` 数组调用变异方法： `example1.items.push({ message: 'Baz' })` 。

变异方法 (mutation method)，顾名思义，会改变被这些方法调用的原始数组。相比之下，也有非变异 (non-mutating method) 方法，例如： `filter()` ， `concat()` 和 `slice()` 。这些不会改变原始数组，但**总是返回一个新数组**。当使用非变异方法时，可以用新数组替换旧数组：

有时会用这种方法来特意更新视图，也可以使用Vue.set() ,也可以使用vm.$set实例方法，该方法是全局方法Vue.set的一个别名

还是由于 JavaScript 的限制，**Vue 不能检测对象属性的添加或删除**：

```JS
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
```

对于已经创建的实例，Vue 不能动态添加根级别的响应式属性。但是，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，对于：

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

9.

`v-for` with `v-if`

当它们处于同一节点，`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。当你想为仅有的一些项渲染节点时，这种优先级的机制会十分有用，如下：

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

上面的代码只传递了未完成的 todos。

10.

然而，任何数据都不会被自动传递到组件里，因为组件有自己独立的作用域。为了把迭代数据传递到组件里，我们要用 `props` ：

```HTML
<my-component
  v-for="(item, index) in items"
  v-bind:item="item"
  v-bind:index="index"
  v-bind:key="item.id"
></my-component>
```

不自动将 `item` 注入到组件里的原因是，这会使得组件与 `v-for` 的运作紧密耦合。明确组件数据的来源能够使组件在其他场合重复使用。

11.

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：

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

```JS
// ...
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)
  }
}
```

12.

事件修饰符

在事件处理程序中调用 `event.preventDefault()` 或 `event.stopPropagation()` 是非常常见的需求。尽管我们可以在方法中轻松实现这点，但更好的方式是：方法只有纯粹的数据逻辑，而不是去处理 DOM 事件细节。

为了解决这个问题，Vue.js 为 `v-on` 提供了**事件修饰符**。之前提过，修饰符是由点开头的指令后缀来表示的。

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`
- `.passive`

13.

修饰符

.lazy

在默认情况下，`v-model` 在每次 `input` 事件触发后将输入框的值与数据进行同步(除了上述输入法组合文字时)。你可以添加 `lazy` 修饰符，从而转变为使用 `change` 事件进行同步：

```HTML
<!-- 在"change"时而非"input"时更新 -->
<input v-model.lazy="msg" >
```

.number

如果想自动将用户的输入值转为数值类型，可以给 `v-model` 添加 `number` 修饰符：

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

这通常很有用，因为即使在 `type="number"` 时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 `parseFloat()` 解析，则会返回原始的值。

.trim

如果要自动过滤用户输入的首尾空白字符，可以给 `v-model` 添加 `trim` 修饰符：

```HTML
<input v-model.trim="msg">
```

14.

15.

`data` 必须是一个函数

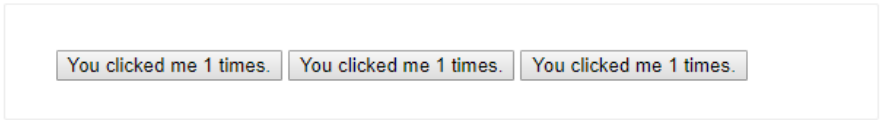
当我们定义这个 `<button-counter>` 组件时，你可能会发现它的 `data` 并不是像这样直接提供一个对象：

```
JS
data: {
  count: 0
}
```

取而代之的是，**一个组件的 `data` 选项必须是一个函数**，因此每个实例可以维护一份被返回对象的独立的拷贝：

```
JS
data: function () {
  return {
    count: 0
  }
}
```

如果 Vue 没有这条规则，点击一个按钮就可能会像如下代码一样影响到*其它所有实例*：



在组件上使用 `v-model`

自定义事件也可以用于创建支持 `v-model` 的自定义输入组件。记住：

```HTML
<input v-model="searchText">
```

等价于：

```HTML
<input
  v-bind:value="searchText"
  v-on:input="searchText = $event.target.value"
>
```

当用在组件上时，`v-model` 则会这样：

```HTML
<custom-input
  v-bind:value="searchText"
  v-on:input="searchText = $event"
></custom-input>
```

为了让它正常工作，这个组件内的 `<input>` 必须：

- 将其 `value` 特性绑定到一个名叫 `value` 的 prop 上
- 在其 `input` 事件被触发时，将新的值通过自定义的 `input` 事件抛出

写成代码之后是这样的：

```JS
Vue.component('custom-input', {
  props: ['value'],
  template: `
    <input
      v-bind:value="value"
      v-on:input="$emit('input', $event.target.value)"
    >
  `
})
```

现在 `v-model` 就应该可以在这个组件上完美地工作起来了：

16.

动态组件

有的时候，在不同组件之间进行动态切换是非常有用的，比如在一个多标签的界面里：

上述内容可以通过 Vue 的 `<component>` 元素加一个特殊的 `is` 特性来实现：

```HTML
<!-- 组件会在 `currentTabComponent` 改变时改变 -->
<component v-bind:is="currentTabComponent"></component>
```

在上述示例中，`currentTabComponent` 可以包括

- 已注册组件的名字，或
- 一个组件的选项对象

你可以在这里查阅并体验完整的代码，或在这个版本了解绑定组件选项对象，而不是已注册组件名的示例。

到目前为止，关于动态组件你需要了解的大概就这些了，如果你阅读完本页内容并掌握了它的内容，我们会推荐你再回来把动态和异步组件读完。

17.

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：

```JS
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。

你可以在风格指南中查阅到关于组件名的其它建议。

18.

组件名大小写

定义组件名的方式有两种：

使用 kebab-case

```JS
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

```JS
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。

19.

20. 路由像是一张网，把整个项目的组件搭建了起来。我们现在用vue-cli构建的项目，组件其实是在模块系统中局部注册的。

传入一个数字

```HTML
<!-- 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>
```

21.

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` ：

```JS
post: {
  id: 1,
  title: 'My Journey with Vue'
}
```

下面的模板：

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

等价于：

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

22.

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：

```
JS
props: ['initialCounter'],
data: function () {
  return {
    counter: this.initialCounter
  }
}
```

2. **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：

```
JS
props: ['size'],
computed: {
  normalizedSize: function () {
    return this.size.trim().toLowerCase()
  }
}
```

23.

> 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。

24. 直接改变数组下标的方式来改变数组中的某个值的时候，Vue是监听不到的。这个是js的限制，这个时候可以采用Vue.set()，或者一些数组的变异方法来实现数组的刷新
25. created 的时候做loading的动画开始，mounted的时候做loading的结束动画。这是一个比较常见的利用生命周期做的效果
26. Vue混入的先执行，原生的后执行。如果混入和原生构造器里面的方法重名了，混入的方法是不会执行的。而是执行原生构造器里面的方法。

27.