## 介绍

1. Vue数据和DOM绑定在一起，并且所有元素都是响应式的（打开控制台，修改值，会看到相应的更新）
2. v-bind属性被称为指令，前缀v-表示是Vue.js提供的特殊属性，在渲染过的DOM上应用特殊的响应式行为。v-bind:title="message"，表示将这个元素节点的title属性和Vue示例的message属性绑定到一起
3. 不仅可以绑定DOM文本到数据，也可以绑定DOM结构到数据，并且Vue.js也提供了一个强大的过渡效果系统，可以在Vue插入/删除元素时自动应用过渡效果。（v-if 控制切换一个元素的显示 v-for绑定数据到数组来渲染一个列表）
4. 组件系统是Vue.js另一个重要概念，它提供了一种抽象，让我们可以用独立可复用的小组件来构建大型应用
5. Vue.js组件非常类似于自定义元素---它是web组件规范（https://www.w3.org/wiki/WebComponents/）的一部分。Web组件规范仍然远未完成，并且没有浏览器实现。相比之下，Vue.js组件不需要任何补丁，并且在所有支持的浏览器之下表现一致，必要时，Vue.js组件也可以放在原生自定义元素之内；Vue.js组件提供了原生自定义元素所不具备的一些重要功能，比如组件间的数据流，自定义事件系统，以及动态的、带特效的组件替换。

## 示例

1. 构造器：每个Vue.js应用都是通过构造函数Vue创建一个Vue的根实例启动的：var vm = new Vue({ }) 虽然没有完全遵循mvvm模式，Vue的设计无疑收到了它的启发。因此在文档中经常会使用vm这个变量名表示Vue示例
2. 在实例化Vue时，需要传入一个选项对象，它可以包含数据、模板、挂载元素、方法、生命周期钩子等选项。可以扩展Vue构造器，从而用预定义选项创建可复用的组件构造器 ：var MyComponent = Vue.extend({// 扩展选项 })

// 所有的 `MyComponent` 实例都将以预定义的扩展选项被创建 var myComponentInstance = new MyComponent() 尽管可以命令式地创建扩展实例，不过在多数情况下建议将组建构造器注册为一个自定义元素，然后声明式地用在模板中。（所有点Vue.js组建都是被扩展的Vue实例）

1. 属性和方法：每个Vue实例都会代理其data对象里所有的属性。只有被代理的属性是响应的。如果在实例创建之后添加新的属性到实例上，它不会触发视图更新。 除了data属性，Vue实例暴露了一些有用的实例属性与方法。这些属性和方法都有前缀$，以便于代理的data属性区分。
2. 实例生命周期：每个Vue实例在被创建之前都要经历一系列的初始化过程。例如，实例需要配置数据观测、编译模版、挂载实例到DOM，然后在数据变化时更新DOM。在这个过程中，实例也会调用一些生命周期钩子，这就给我们提供了执行自定义逻辑的机会。

## 模板语法

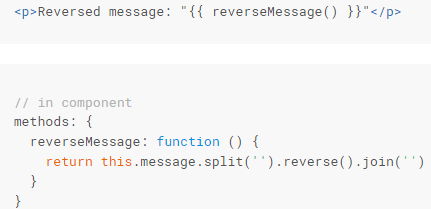
1. Vue.js是用来基于HTML的模版语法，允许开发者声明式地将DOM绑定至底层Vue实例的数据。所有的Vue.js的模版都是合法的HTML，所以能被遵循规范的浏览器和HTML解析器解析；在底层的实现上，Vue将模板编译成虚拟DOM渲染函数，结合响应系统，在应用状态改变时，Vue能够智能地计算出重新渲染组件的最小代价并应用到DOM操作上。
2. 插值
3. 文本：数据绑定最常见的形式就是使用‘Mustache’语法（双大括号）的文本插值 <span>Message: {{ msg }}</span> Mustache标签将会被替代为对应数据对象上msg属性的值，无论何时，绑定的数据对象上msg属性发生了改变，插值处的内容都会更新。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（这会影响到该节点上所有的数据绑定）
4. 纯HTML：双大括号会将数据解析为纯文本，而非HTML。为了输出真正的HTML，你需要使用v-html指令 <div v-html="rawHtml"></div> 被插入的内容都会被当做HTML----数据绑定会被忽略。不能使用v-html来复合局部模板，因为Vue不是基于字符串的模板引擎，组件更适合担任UI重用和复合的基本单元。（站点上动态渲染的任意HTML可能会非常危险，因为它很容易导致xss攻击，请只对可信内容使用HTML插值，绝对不要对用户提供的内容插值）
5. 属性：Mustache不能再HTML属性中使用，应使用v-bind指令 <div v-bind:id="dynamicId"></div> 这对布尔值的属性可有效——如果条件被求值为false的话该属性会被移除：<button v-bind:disabled="someDynamicCondition">Button</button>
6. 对于所有的数据绑定，Vue.js都提供了完全的JavaScript表达式支持。这些表达式会在所属Vue实例的数据作用域下作为JavaScript被解析。有个限制就是，每个绑定都只能包含单个表达式
7. 指令（directives）：是带有v-前缀的特殊属性。指令属性的值预期是单一JavaScript表达式（除了v-for）。指令得职责就是当其表达式的值改变时相应地将某些行为应用到DOM上。
8. 参数：一些指令能接受一个‘参数’，在指令后以冒号指明。例如，v-bind指令被用来相应地更新HTML属性：<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。 V-on，用于监听DOM事件
9. 修饰符（Modifiers）：以半角句号. 指明的特殊后缀，用于指出一个指定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（）：<form v-on:submit.prevent="onSubmit"></form>
10. 过滤器（filters）：Vue.js允许你自定义过滤器，被用作一些常见的文本格式化。过滤器应该被添加在**mustache** 插值的尾部，由‘管道符’指示（Vue2.x中，过滤器只能在**mustache** 绑定和v-bind表达式中使用，因为过滤器设计目的就是用于文本转换，为了在其他指令中实现更复杂的数据变换，应该使用计算属性） 过滤器函数总接受表达式的值作为第一个参数；过滤器可以串联；过滤器是JavaScript函数，可以接受参数
11. 缩写：v-前缀在模板中是作为一个标示Vue特殊属性的明显标识。当你使用Vue.js为现有的标记添加动态行为时，它会有用，但对于一些经常使用的指令来说有点繁琐。同时，当搭建Vue.js管理所有模块的SPA时，v-前缀也变得没那么重要了。因此，Vue.js为两个最为常用的指令提供了特别的缩写：<a v-bind:href="url"></a> <a :href="url"></a> <a v-on:click="doSomething"></a> <a @click="doSomething"></a> 它们看起来可能与普通的HTML略有不同，但：与@对于属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解析。而且不会出现在最终渲染的标记。

## 计算属性

1. 计算属性：在模板中绑定表达式是非常便利的，但是它们实际上只用于简单的操作。在模板中放入太多的逻辑会让模板过重且难以维护，例如：{{ message.split('').reverse().join('') }} 在这种情况下，模板不再简单清晰。在实现反向显示message之前，你应该确认它
2. 基础例子：当vm.message发生改变时，依赖于它的vm.reversedMessage的绑定也会更新



1. 计算缓存 vs Methods：可以通过表达式中的method来达到相同的效果，不经过计算属性，可以在method中定义一个相同的函数来替代它。结果相同，不同的是**计算属性是基于它的依赖缓存**。计算属性只有在它的相关依赖发生改变时才会重新取值。每当重新渲染的时候，method调用总会执行函数。



假设有一个重要的计算属性A，这个计算属性需要一个巨大的数组遍历和做大量的计算，然后可能有其它的计算属性依赖于A，如果没有缓存，将不可避免的多次执行A的getter（不希望有缓存，用method替代）

1. 计算属性 vs Watched Property：Vue.js提供了一个方法$watch，它用于观察Vue实例上的数据变动。（计算属性相比$watch更好）
2. 计算 setter：计算属性默认只有getter，不过在需要时你也可以提供一个setter，计算属性改变时，会调用setter
3. 观察watchers：虽然计算属性在大多数情况下更适合，但有时也需要一个自定义的watcher，这也是为什么Vue提供一个更通用的方法通过watch选项来响应数据的变化，当你想要在数据变化响应时，执行异步操作或开销较大的操作，这是很有用的

## Class 与 Style 绑定

1. 数据绑定一个常见需求是操作元素的class列表和它的内联样式。因为它们都是属性，可以用v-bind处理：只需要计算出表达式最终的字符串。字符串拼接麻烦又易错。故在v-bind用于class和style时，Vue专门增强了它，表达式的结果类型除了字符串之外，还可以是对象或数组。
2. 绑定HTML Class
3. 对象语法：传给v-bind:class一个对象，以动态地切换class。<div v-bind:class="{ active: isActive }"></div> class active的更新取决于数据属性isActive是否为真值。也可以在对象中传入更多属性用来动态切换多个class。V-bind:class指令可以与普通的class属性共存。也可以直接绑定数据里的一个对象。这里也可以返回对象的计算属性。
4. 数组语法：可以把一个数组传给v-bind:class，以应用一个class列表<div v-bind:class="[activeClass, errorClass]"> 如果想根据条件切换列表中的class，可以用三元表达式：<div v-bind:class="[isActive ? activeClass : '', errorClass]"> 不过当有多个条件class时这样写有些繁琐，可以在数组语法中使用对象语法<div v-bind:class="[{ active: isActive }, errorClass]">
5. 用在组件上：当在一个定制的组件上用到class属性的时候，这些类将被添加到根元素上面，这个元素上已经存在的类不会被覆盖。
6. 绑定内联样式
7. 对象语法：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（cameCase）或者短横分隔命名（kebab-case）<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div> 直接绑定到一个样式对象上更好，让模板更清晰<div v-bind:style="styleObject"></div> 同样的，对象语法常常结合返回对象的计算属性使用
8. 数组语法：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
9. 自动添加前缀：当v-bind:style使用需要特定前缀的CSS属性时，如transform，Vue会自动侦测并添加相应的前缀

## 条件渲染

1. v-if：v-if是一个指令，需要将它添加到一个元素上。可以添加else块，v-else元素必须紧跟在v-if元素或者v-else-if的后面，否则不能被识别。Vue尝试尽可能高效的渲染元素，通常会复用已有元素而不是从头开始渲染
2. v-show：根据条件展示元素的选项。V-show的元素会始终渲染并保持在DOM中，它是简单的切换元素的CSS属性display。（v-show不支持<template>语法）
3. v-if VS v-show ：v-if是真实的条件渲染，因为它会确保条件块在切换当中适当地销毁与重建条件块内的事件监听器和子组件，它也是惰性的，如果在初始渲染时条件为假，则什么都不做——在条件第一次变为真时才开始局部编译（编译会被缓存起来）。相比之下，v-show简单得多——元素始终被编译保存，只是简单地基于CSS切换。一般来说，v-if有更高的切换消耗而v-show有更高的切换消耗而v-show有更高的初始渲染消耗，因此，如果需要频繁切换使用v-show较好，如果在运行条件不太可能改变则使用v-if较好。

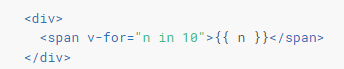
## 列表渲染

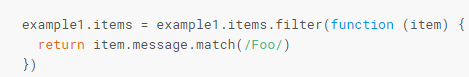
1. v-for：v-for指令根据一组数组的选项列表进行渲染，需要以item in items形式的特殊语法，items是源数据数组并且item是数组元素迭代的别名。
2. 基本用法：在v-for块中，拥有对父级作用域属性的完全访问权限，还支持一个可选的第二个参数为当前项的索引。也可以用of替代in作为分隔符，因为它最接近JavaScript迭代器的语法



1. Template v-for：如同v-if模板，也可以用带有v-for的<template>标签来渲染多个元素块
2. 对象迭代v-for：可以用v-for通过一个对象的属性来迭代，提供第二个参数为键名，第三个参数为索引（在遍历对象时，是按Object.keys()的结果遍历，但是不能保证它的结果在不同的JavaScript引擎下是一致的）



1. 整数迭代v-for：v-for也可以取整数，这种情况下，它将重复多次模板
2. 组件和v-for：在自定义组件里，可以像任何普通元素一样用v-for，但是它不能自动传递数据到组件里面，因为组件有自己独立的作用域，为了传递迭代数据到组件里，要用props。不自动注入item到组件里的原因是这使得组件会紧密耦合到v-for如何运行。在一些情况下，明确数据的来源可以使组件可重用
3. Key：当Vue.js用v-for正在更新已渲染过的元素列表时，它默认用‘就地复用’策略。如果数据项的顺序被改变，Vue将不是移动DOM元素来匹配数据项的顺序，而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素
4. 数组更新检测
5. 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：push()添加到末尾、pop()删除最后一个数据、shift()删除第一个、unshift()添加到首位、splice()删除/添加某一个、sort()排序、reverse()倒置
6. 重塑数组：变异方法（mutation，method），会改变被这些方法调用的原始数组。相比之下也有非变异方法：filter()、concat()添加新的元素、slice()查找指定元素，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。



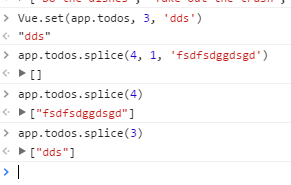
1. 由于JavaScript的限制，Vue不能检测到以下变动的数组：
2. 当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue
3. 当你修改数组的长度时，例如vm.items.length = newLength等
4. 为了避免第一种种情况，用以下两种方式达到同样效果，同时也会将触发状态更新。



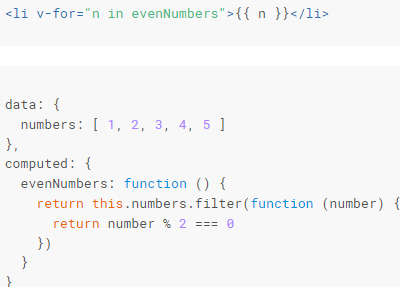
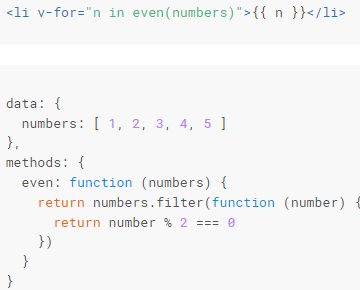


为了避免第二种情况，使用splice





1. 显示过滤/排序结果：显示一个数组的过滤或排序副本，而不实际改变或重置原始数据，这种情况下，可以创建返回过滤或排序数组的计算属性。或者，在计算属性不合适的情况下（在嵌套v-for循环中）使用method方法

## 事件处理器

1. 监听事件：可以用v-on指令监听DOM事件来触发一些JavaScript代码
2. 方法事件处理器：许多事件处理的逻辑都很复杂，所以直接把JavaScript代码写在v-on指令中是不可行的，因此v-on可以接受一个定义的方法来调用
3. 内联处理器：除了直接绑定到一个方法，也可以用内联JavaScript语句，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
4. 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。尽管可以再methods中轻松实现这点，但更好的方式是：methods只是纯粹的数据逻辑，而不是去处理DOM事件细节。为了解决这个问题，Vue为v-on提供了事件修饰符。通过有点（.）表示的指令后缀来调用修饰符 eg：.stop（阻止事件冒泡） .prevent（阻止默认事件） .capture（修饰符加在父容器事件上面，事件捕获模式 先执行父容器事件 在子元素事件） .self（当事件在该元素本身（而不是子元素）触发时触发回调） .once（绑定事件只执行一次）
5. 按键修饰符：在监听键盘事件时，常需要监测常见的键值，Vue允许为v-on在监听键盘事件时添加按键修饰符



Vue为最常用的按键提供了别名：.enter .tab .delete .esc .space .up .down .left .right，可以通过全局config.keyCodes对象自定义按键修饰符别名。可以用以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

1. 为什么在HTML中监听事件？
2. 扫一眼HTML模板便能轻松定位在JavaScript代码里对应的方法
3. 无须在JavaScript里手动绑定事件，ViewModel代码可以是非常纯粹的逻辑，和DOM完全解耦，更易于测试
4. 当一个ViewModel被销毁时，所有的事件处理器都会自动被删除

## 表单控件绑定

1. 基础用法：可以用v-model指令在表单控件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，它本质上市语法糖，负责监听用户的输入事件以更新数据，并特别处理一些极端的例子
2. 文本
3. 多行文本
4. 复选框
5. 单选按钮
6. 选择列表
7. 绑定value：对于单选按钮、复选框以及选择列表，v-model绑定的value通常是静态字符串（对于复选框是逻辑值）
8. 复选框
9. 单选按钮
10. 选择列表设置
11. 修饰符
12. .lazy：在默认情况下，v-model在input事件中同步输入框的值与数据，但可以添加一个修饰符lazy，从而转变为在change事件中同步



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，在type=“number”时HTML中输入的值也总是会返回字符串类型



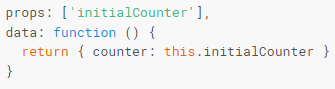
1. .trim：如果要自动过滤用户输入的首尾空格，可以添加trim修饰符到v-model上过滤输入
2. V-model与组件：HTML内建的input类型有时不能满足你的需求。Vue的组件系统允许你创建一个具有自定义行为可复用的input类型，这些input类型甚至可以和v-model一起使用。

## 组件

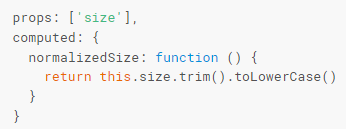
1. 什么是组件？

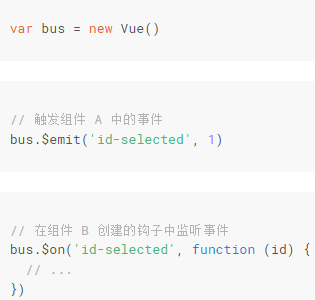
组件是Vue最强大的功能之一。组件可以扩展HTML元素，封装可重用的代码。在较高层面上，组件是自定义元素，Vue的编译器为它添加特殊功能。在有些情况下，组件也可以是原生HTML元素的形式，以is特性扩展

1. 使用组件
2. 注册：可以使用Vue.component('my-component', options)注册一个全局组件，组件在注册之后，便可以在父实例的模块中以自定义元素<my-component></my-component>的形式使用，要确保在初始化根实例之前注册了组件。
3. 局部注册：不必在全局注册每个组件，通过使用组件实例选项注册，可以使组件仅在另一个实例/组件的作用域中可用。这种封装也适用于其它可注册的Vue功能，如指令。
4. DOM模板解析说明：当使用DOM作为模板时（例如，将el选项挂载到一个已存在的元素上），会受到HTML的一些限制，因为Vue只有在浏览器解析和标准化HTML后才能获取模板内容。尤其像这些元素<ul>、<ol>、<table>、<select>限制了能被它包裹的元素，<option>只能出现其它元素内部，在自定义组件中使用这些受限制的元素时导致一些问题
5. data必须是函数：使用组件时，大多数可以传入到Vue构造器中的选项可以在注册组件时使用，有一个例外：data必须是函数
6. 构成组件：组件意味着协同工作，通常父子组件会是这样的关系：组件A在它的模板中使用了组件B。他们之间必然需要相互通信：父组件要给子组件传递数据，子组件需要将它内部发生的事情告知给父组件。然而在一个良好定义的接口中尽可能将父子组件解耦是很重要的，这保证了每个组件 可以在相对隔离的环境中书写和理解，也大幅提高了组件的可维护性和可重用性。在Vue中，父子组件的关系可以总结为props down，events up。父组件通过props向下传递数据给子组件，子组件通过events给父组件发送消息。
7. Prop
8. 使用Prop传递数据：组件实例的作用域是孤立的。这意味着不能并且不应该在子组件的模板内直接引用父组件的数据，可以使用props把数据传给子组件。prop是父组件用来传递数据的一个自定义属性。子组件需要显式地用props 选项声明prop
9. camelCase vs kebab-case：HTML特性不区分大小写，当使用非字符串模板时，prop的名字形式会从camelCase（驼峰）转为kebab-case（短横线隔开）
10. 动态Prop：类似于用v-bind绑定HTML特性到一个表达式，也可以用v-bind动态绑定props的值到父组件的数据中，每当父容器的数据变化时，改变化也会传导给子组件
11. 字面量语法 vs 动态语法：一个常见的错误是使用字面量语法传递数值<comp some-prop="1"></comp> 它是一个字面prop，它的值以字符串“1”而不是以实际数字传下去。如果想传递一个实际的JavaScript数字，需要使用v-bind，从而让它的值被当作JavaScript表达式计算<comp v-bind:some-prop="1"></comp>
12. 单向数据流：prop是单向绑定的：当父组件的属性变化时，，将传导给子组件，但是不会反过来，这是为了防止子组件无意修改了父组件的状态——这会让应用的数据流难以理解。每次父组件更新时，子组件的所有prop都会更新为最新值，这意味着不应该在子组件内部改变prop。通常有两种改变prop的情况：
13. prop作为初始值传入，子组件之后只是将它的初始值作为本地数据的初始值使用（定义一个局部data属性，并将prop的初始值作为局部数据的初始值）



1. prop作为需要被转变的原始值传入（定义一个computed属性，此属性从prop的值计算得出）



1. 在JavaScript1中对象和数组是引用类型，指向同一个内存空间，如果prop是一个对象或数组，在子组件内部改变它会影响父组件的状态
2. Prop验证：组件可以为props指定验证要求。如果未指定验证要求，Vue会发出警告。当组件给其他人使用时这很有用。Prop是一个对象而不是字符串数组时，它包含验证要求，type可以是原生构造器：String、Number、Boolean、Function、Object、Array。Type也可以是一个自定义构造器，使用instanceof检测
3. 自定义事件：父组件是使用props传递数据给子组件，如果子组件要把数据传递回去，应该是自定义事件
4. 使用v-on绑定自定义事件：每个Vue实例都实现了事件接口（Events interface），即使用$on（eventName）监听事件，使用$emit（eventName）触发事件，另外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
5. 使用自定义事件的表单输入组件：自定义事件也可以用来创建自定义的表单输入组件，使用v-model来进行数据双向绑定。想要组件的v-model生效，它必须接受一个value属性，在有新的value时触发input事件
6. 非父子组件通信：有时候非父子关系的组件也需要通信
7. 使用Slot分发内容：在使用组件时，注意两点：<app>组件不知道它的挂载点会有什么内容，挂载点的内容是由<app>的父组件决定的；<app>组件很可能有它自己的模板。为了让组件可以组合，需要一种方式来混合父组件与子组件自己的模板，这个过程称为内容分发。Vue实现了一个内容分发API，使用特殊的<solt>元素作为原始内容的插槽。
8. 编译作用域：组件作用域简单来说就是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。分发内容是在父组件作用域内编译
9. 单个Solt：除非子组件模板包含至少一个<solt>插口，否则父组件的内容将会被丢弃。当子组件模板只有一个没有属性的solt时，父组件整个内容片段将插入到solt所在的DOM位置，并替换掉solt标签本身。最初在<solt>标签中的任何内容都被视为备用内容，备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容
10. 具名Solt：<solt>元素可以用一个特殊的属性name来配置如何分发内容。多个solt可以有不同的名字，具名solt将匹配内容片段中有对应solt特性的元素。仍然可以有一个匿名solt，它是默认solt，作为找不到匹配的内容片段的备用插槽，如果没有默认的solt，这些找不到匹配的内容片段将被抛弃
11. 作用域插槽：作用域插槽是一种特殊类型的插槽，用作使用一个（能够传递数据到）可重用模板替换已渲染元素。在子组件中，只需要将数据传递到插槽中，就像将prop传递给组件一样
12. 动态组件：多个组件可以使用同一个挂载点，然后动态地在它们之间切换。使用保留的<component>元素，动态地绑定到它的is特性，也可以直接绑定到组件对象上。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，可以添加一个keep-alive指令参数
13. 杂项：
14. 编写可服用组件：在编写组件时，记住是否复用组件有好处。一次性组件和其它组件紧密耦合没关系，但是可复用组件应当定义一个清晰的公开接口。Vue组件的API来自三个部分—props、events、solts。Props允许外部环境传递数据给组件；Events允许组件触发外部环境的副作用；Solts允许外部环境将额外的内容组合在组件中
15. 子组件索引：尽管有props和events，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为组件指定一个索引ID。当ref和v-for一起使用时，ref是一个数组或对象，包含相应的子组件。$refs只有组件渲染完成后才填充，并且它是非响应式的，它仅仅作为一个直接访问子组件的应急方案——应当避免在模板或计算属性中使用$refs
16. 异步组件：在大型应用中，我们可能需要将应用拆分为多个小模块，按需从服务器下载。Vue允许将组件定义为一个工厂函数，动态地解析组件的定义。Vue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，工厂函数接受一个resolve回调，在收到服务器下载的组件定义时调用，也可以调用reject（reason）指示加载失败
17. 组件命名约定：在注册组件（或者props）时，可以使用kebab-case，camelCase或TitleCase；在HTML模板中，请使用kebab-case形式；当使用字符串模式时，可以不受HTML的case-insensitive限制，可以使用kebab-case，camelCase或TitleCase来引用；如果组件未经solt元素传递内容，可以在组件名后使用/使其自闭合，只在字符串模板中有效。自闭的自定义元素是无效的HTML，浏览器原生的解析器也无法识别它
18. 递归组件：组件在它的模板内可以递归地调用自己，只有当它有name选项时才可以（确保递归调用有终止条件）
19. Circular References Between
20. Components
21. 内联模板：如果子组件有inline-template特性，组件将把它的内容当做它的模板，而不是把它当作分发内容，让模板更灵活，但是inline-template让模板的作用域难以理解，最佳实践是使用template选项在组件内定义模板或者在.vue文件中使用template。
22. X-Templates：另一种定义模板的方法是在JavaScript标签里使用text/template类型，并且指定一个id；这在有很多模板或者小的应用中有用，否则应该避免使用，因为它将模板和组件的其它定义隔离了
23. 使用v-on的低级静态组件：尽管在Vue中渲染HTML很快，不过当组件中包含大量静态内容时，可以考虑使用v-on将渲染结果缓存起来