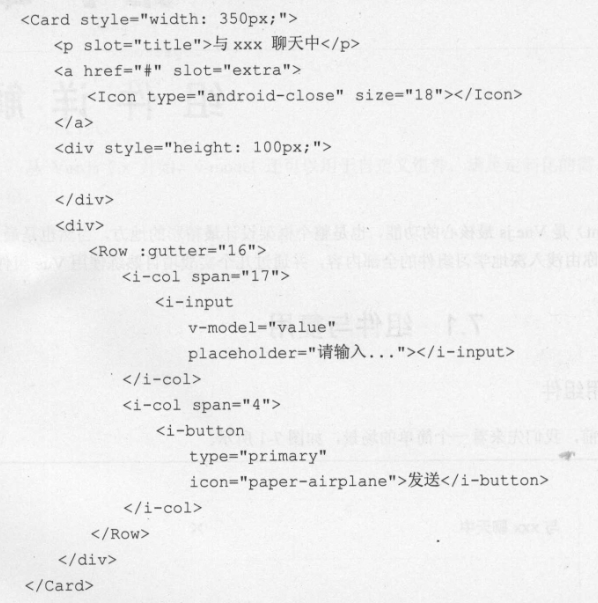
# 认识组件

## 为什么要使用组件

对于一些经常需要使用的控件（如轮播图），如果多个页面都要使用，通过复制粘贴代码的形式来实现复用，很明显效率是非常低的。此时，可以将这些控件定义为组件，提高复用性，在需要使用的地方引用。例如：



上述代码中，有很多我们从来没有见过的标签，比如<Card>、<Row>等，这些没见过的自定义标签就是组件，在任何使用Vue的地方都可以直接使用。

## 地位

组件（Component）是Vue.js最核心的功能。

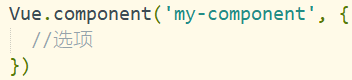
# 组件的注册与使用

## 注册组件

组件需要注册后才可以使用，注册组件分为全局注册和局部注册两种方式。

**全局注册**

全局注册的组件，在任何Vue实例中都可以使用。语法如下：



my-component就是注册组件的自定义标签名称，推荐使用小写加减号分隔的形式命名。示例代码如下：



<my-component>标签被渲染为template选项对应的内容，需注意，template的值必须被一个DOM元素包含，上述代码中如果写成“这里是组件的内容”，不使用<div></div>包含是无法被渲染的。同时，使用自定义组件的所在区域位置必须被Vue实例挂载了。

**局部注册**

在Vue实例中，使用components选项可以局部注册组件，注册后的组件只能在该实例作用域下有效（即该实例挂载元素内范围）。示例代码如下：



注册局部组件中，组件名称的值是一个至少包含template属性的对象。

## 使用组件

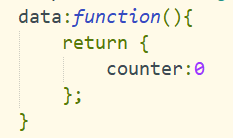
1. 自定义组件中除了使用template选项定义结构外，还可以像Vue实例那样使用其他的选项，比如data、computed（计算属性）、method等。但是在使用data时，和实例稍有区别，data必须是函数，然后将数据作为对象的一个属性将该对象return返回。示例如下：



同时需要注意，JavaScript对象是引用关系，所以如果return返回的是一个外部对象，那么这个外部对象就是共享的，任何一方修改都会同步。示例如下：



当点击任意一个按钮时，所有按钮上的值都会变化，这就是因为组件的data引用的外部的对象，实现了共享的效果，这肯定不是我们期望的结果，所以最好不要给组件返回一个外部对象，而是使用新data对象来独立。



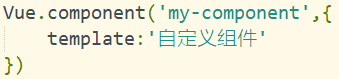
这样，点击3个按钮就互不影响了，达到了复用的目的。

1. Vue组件的模板在某些情况下会受到HTML语法的限制，比如<table>内规定只允许时<tr>、<td>、<th>等这些元素，所以在<table>内直接使用组件是无效的。这种情况下，可以使用特殊的is属性来挂载组件，而不是使用标签。示例如下：

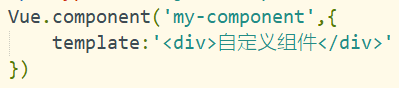


上述代码中，<tbody>在渲染时，会被替换为组件的内容。除了<table>外，常见的限制元素还有<ul>、<ol>、<select>。

1. 在注册组件时，无论是全局注册还是局部注册，组件的结构内容即template选项的值的最外层应该被一个DOM结构包含。如下是错误的：



因为template选项的值最外层必须被DOM元素包含，如下是正确的：



1. 在注册组件时，也可以使用components选型来注册组件，是组件嵌套，但嵌套的组件只能在最近的上层组件中使用。示例如下：



在嵌套的组件中，仍然还可以使用components实现组件的再嵌套，同理，嵌套的组件只能在最近的上层组件中使用。

1. 在使用自定义的组件（不论全局组件还是局部组件）时，使用的位置必须被Vue实例挂载。例如：



由于id=app的<div>未被Vue实例挂载，所以自定义的<my-component>不能正常显示，要想正常显示，必须将id=app的<div>元素挂载到Vue实例上。如下：



同时，在注册全局组件时，其定义必须创建Vue实例的前面，如下先定义组件后创建Vue实例：



以下先创建Vue实例后定义组件是错误的：



1. 在注册组件时，组件的名称（即自定义标签的名称）推荐使用小写加减号分隔的形式命名，如my-component。

# 使用props传递数据（组件的开放能力）

## 为什么要传递数据给组件

组件不仅仅是要把模板的内容进行复用，更重要的是组件间要进行通信。通常父组件的模板包含子组件，父组件要正向的向子组件传递数据或参数，子组件接收到后根据参数的不同来渲染不同的内容或执行操作，这个正向传递数据的过程就是通过props来实现的。

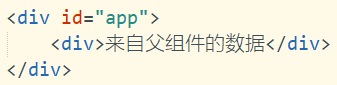
## 基本用法

在组件中，使用props选项接收父级中传过来的数据，props选项的值可以是两种：一种是字符串数组，另一种是对象。本节只介绍字符串数组形式的用法，关于对象形式的用法会在后面讲数据验证时使用。

1）使用字符串数组形式接受来自父级的数据，示例如下：



渲染后的结果为：



上例中，父级通过自定义属性并给属性赋值的方式向组件中传入数据，组件中通过props选项，使用字符串数组的形式接受参数，并在template中使用。注意，props中声明的数据与组件data函数return返回的数据主要区别就是props来自父级，而data中的是组件自己的数据，作用域是组件本身。这两种数据都可以在模板template、计算属性computed、方法methods中使用。

2）由于HTML语法不区分字母大小写，当使用DOM模板时，驼峰命名（camelCase）

的props名称要转换为分隔命名，在父组件上定义的属性要么是小写，要么是a-b的形式，然后子组件在props中使用驼峰命名aB接收，不允许父组件通过aB的形式定义属性向组件传递数据（总之记住一点，不允许父组件通过驼峰命名的形式属性向子组件传递数据）。例如：



父级使用waring-text，在props将其转成驼峰命名warningText（如果使用的是字符串模板，可以忽略这些限制，关于字符串模板后面的章节会讲）。

3）有时传递的数据并不是直接写死的，而是来自父级的动态数据，这是可以在父级使

用指令v-bind来动态绑定一个props的值，当父组件的数据变化时，也会触发传递给子组件。示例如下：



上例中，使用v-model绑定了数据parentMessage，当在文本框任意输入内容时将实时同步到Vue示例中的parentMessage，而父级通过v-bind将message和parentMessage进行绑定，一旦parentMessage发生改变，子组件接收到message也会实时响应，并更新组件模板。

4）如果父级要传递数字、布尔值、数组、对象给组件，必须使用v-bind，否则传递的

仅仅是字符串。示例如下：



上例中，同一个组件使用了两次，区别仅仅是第二个使用的是v-bind，渲染后的结果，如下：

从结果可以看出，第一个没有使用v-bind的父级传递的数组被当成了字符串处理，而第二个使用了v-bind传递的数组依旧是数组。

## 单向数据流

Vue2.x与Vue1.x比较大的改变就是，Vue2.x通过props传递数据时单向的了，也就是父组件数据变化时会传递给子组件，但是反过来不行；在Vue1.x提供了.sync修饰符来支持双向绑定，之所以这样设计，是尽可能将父子组件解耦，避免子组件无意中修改了父组件的状态。

## 使用数据

当子组件使用props接收父级传的数据后，针对接收到的数据，大致有三种用途。但无论是哪一种用途，数据最终都会被用到模板template或业务代码中（不然没有意义）。

**在模板中通过通过插值表达式或指令直接使用**

将父组件传递过来的数据直接在模板中通过插值表达式或指令进行使用。示例：



该示例中直接将父组件传递的数据通过插值表达式（或通过指令引用）用在了模板上，当然，也可以在插值表达式中进行一些简单的运算或调用方法进行处理。

**为组件的data赋值**

将父组件中传递过来的数据为组件中声明的data数据赋值，组件再通过其他方式使用data中数据。如下：



**对数据进行处理后再使用（使用计算属性）**

将父组件中传递过来的数据进行处理再使用。示例如下：



该示例中，演示了通过父组件的传递的参数来决定设置子组件的长度、字体颜色，通过计算属性进行处理，然后通过指令在模板中使用处理后的数据。需要注意，在computed、data、method中获取父组件传递的数据需要使用this关键字，表示当前组件。

## 数据验证

前面示例中，props选项的值都是一个数组，一开始节介绍过，除了数组外，还可以是props选项的值还可以是一个对象，当需要验证父组件传的prop时，props就需要使用对象写法。当你的组件需要提供给别人使用时，推荐都进行数据验证，比如某个数据必须是数字类型，如果传入字符串，就会在控制台弹出警告。





验证的type类型可以是：

·String

·Number

·Boolean

·Object

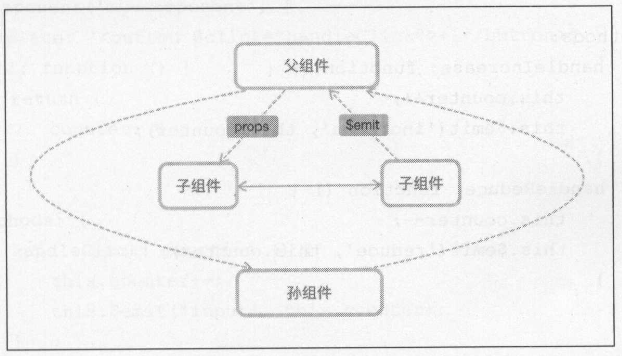
·Array

·Function

type也可以是一个自定义构造器，使用instanof检测。当prop验证失败时，在开发版本下会在控制台抛出一条警告。

# 组件通信

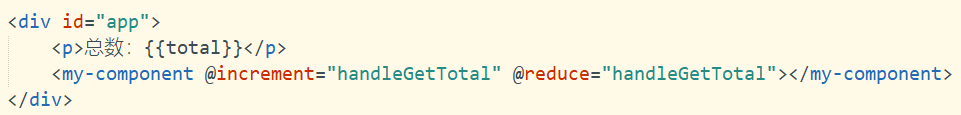
通过前面的学习我们已经知道，父组件向子组件通信，通过props传递数据就可以了。但Vue组件通信的场景不止有这一种，归纳起来，组件之间的通信可以用下图表示：



组件关系分为父子组件通信、兄弟组件通信、跨级组件通信。本节将介绍各种组件之间通信的方法。

## 自定义事件

当子组件需要向父组件传递数据时，可以使用自定义事件。前面在学习v-on指令时有提到，v-on除了可以监听DOM事件外，还可以用于组件之间的自定义事件。Vue组件中，子组件用$emit( )来触发事件，父组件使用$on( )来监听子组件的事件，父组件也可以直接在子组件的自定义标签上使用v-on来监听子组件触发的自定义事件。示例如下：





上例中，子组件有两个按钮，并绑定了事件，实现+1和-1的效果，当改变data中counter后，通过$emit( )将counter传给父组件，$emit( )函数的第一个参数是自定义事件的名称。

## 中央事件总线

Vue.js2.x中，推荐使用一个空的Vue实例作为中央事件总线（bus），它相当于一个中介的作用，通过该中介实现组件间的通信。示例代码如下：



上例中，首先创建了一个名为bus的空Vue实例，充当总线；然后全局定义了组件component-a；最后创建Vue实例app，在app实例化时，也就是在生命周期mounted钩子函数里监听bus的事件on-message，而在组件component-a中，点击按钮会通过bus把事件on-message发出去，此时app实例就会接收到来自bus的事件，进行在回调里完成自己的业务逻辑。这种方法巧妙而轻量地实现了任何组件间的通信，包括父子、兄弟、跨级，而且Vue1.X和Vue2.X都适用。如果深入适用，可以扩展bus实例，给它添加data、methods、computed等选项，这些都是可以公用的。在业务中，尤其是协同开发时非常有用，因为经常需要共享一些通用的信息。

## 父链和子链

在子组件中可以使用this.$parent访问该组件的父实例或组件，在父组件中可以通过this.$children访问它所有的子组件，而且可以递归向上或向下无限访问，直到根实例或最内层的组件。示例代码如下：



尽管Vue允许在子组件中使用父链访问父组件或在父组件中使用子链访问子组件，但在业务中，子组件应该尽可能的避免依赖父组件的数据，更不应该去修改父组件中的数据，因为这样使得父子组件紧耦合，只看父组件，很难理解父组件的状态，因为它可能被任意组件修改了。在理想情况下，只有组件自己能修改它的状态。父子组件最好还是通过props和$emit来通信。

## 子组件索引

当子组件较多时，通过this.$children（子链）来一一遍历出我们需要的一个组件实例时比较麻烦的，尤其是组件动态渲染时，它们的序列是不固定的。Vue子组件索引的方法，用特殊的属性来为子组件指定一个索引名称。示例如下：



上例中，子组件标签上使用ref属性指定一个名称，在父组件中通过this.$refs来获取指定名称的子组件。（提示：$refs只在组件渲染完成后才填充，并且它是非响应式的，应当避免在模板或计算属性中使用$refs，它仅仅作为一个直接访问子组件的应急方案。）

# slot分发内容

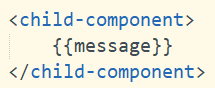
## 认识slot

当需要混合父组件的内容与子组件的模板时，就会用到slot，slot是自定义组件标签中的内容；在组件的template中定义<slot>元素，对该组件标签中的内容进行分发，这个过程叫作内容分发（transaclusion）。

虽然slot在业务中几乎用不到，但在用render函数（进阶篇中将介绍）创建组件时会比较有用，但主要还是用于独立组件的开发中。

## 作用域

在介绍slot前，需要先知道一个概念：编译的作用域。比如父组件中有如下模板：



这个自定义组件标签包含的内容message就是一个slot，但是它绑定的是父组件的数据，而不是<child-component>的数据。父组件模板的内容是在父组件作用域内编译，子组件模板的内容是在子组件作用域内编译。例如：



上例的showChild绑定的是父组件的数据，如果想在子组件上绑定，那应该是：



slot分发的内容，作用域是在父组件上的。

## slot用法

**单个slot**

在子组件内使用<slot>元素就可以为这个子组件开启一个slot（插槽），在父组件模板里，插入在子组件标签内的所有内容将替代子组件的<slot>标签及它的内容。示例如下：



子组件child-component的模板template中定义了一个<slot>元素，并且用一个<p>作为默认的内容，在父组件没使用slot时，会渲染这段默认的文本；如果父组件写入了slot，就会替换整个<slot>

**具名slot**

给自定义组件模板中包含的<slot>元素指定一个name后可以分发多个内容，具名slot可以与单个slot共存。示例如下：



子组件内声明了3个<slot>元素，其中在<div class=”main”>内的<slot>没有使用name特性，它将作为默认slot出现，父组件没有使用slot特性的元素与内容都将出现在这里，如果此处没有指定默认的匿名slot，父组件内多余的内容片段都将被抛弃。上例中最终的渲染结果为：



## 作用域插槽

作用域插槽是一种特殊的slot，使用一个可以复用的模板替换已渲染元素。

1）示例如下：



在组件的模板中，在<slot>元素上有一个类似props传递数据给组件的写法，msg=”xxx”，将数据传到了插槽。父组件中使用了<template>元素，而且拥有一个scope=”props”的特性，这里的props只是一个临时变量，就像v-for=”item in teims”里面的item一样，该变量是一个对象，通过属性的方式保存了子组件模板中传递给插槽的数据。template内可以访问该对象的属性来获取子组件插槽的数据。上例渲染结果为：



2）作用域插槽更具代表性的用例是列表组件，允许组件自定义应该如何渲染列表每一

项。示例如下：



子组件my-list接收一个来自父级的prop数组books，并且将它在name为book的slot上使用v-for指令循环，同时暴露一个变量bookName，将其传给template。（上例只是为了介绍作用域插槽的用法，实际业务中并不会这么写，上例中可以直接在父组件用v-for就可以了）

## 访问slot

Vue 2.X提供了$slots来访问被slot分发的内容。示例如下：



上例中，在组件中通过$slots来访问该组件中被slot分发的内容，其中，default包括了所有没有被包含在具名slot中的节点。

# 组件高级用法

本章介绍的组件的一些高级用法，虽然在实际业务中不是很常用，但是在开发独立组件时可能会用到。

## 递归组件

给组件设置name选项后，可以允许组件在它的模板内可以递归的调用自己。示例如下：



为组件设置name选项后，在组件模板内就可以递归使用了，但需要注意，必须在递归调用时给定一个条件来限制递归数量，否则会抛出错误：Maximum call stack size exceeded。组件递归使用可以用来开发一些具有未知层级关系的独立组件，比如级联选择器和树形控件等。

## 内联模板

前面在定义组件时，组件的模板都是在template选项中定义的。Vue提供了内联模板的功能，在使用组件时，给组件标签加上inline-template属性，组件就会把标签内的内容当做模板，而不是把它当做内容分发。示例如下：



（运行结果和书上的运行结果不符）。如果不是非常特殊的场景，建议不要轻易使用内联模板。

## 动态组件

Vue.js提供了一个特殊的元素<component>用来挂载不同的组件，使用is属性来选择要挂载的组件。示例如下：



上例中，动态改变currentView的值就可以动态挂载组件了。当然，也可以将is属性直接绑定在组件对象上：



## 异步组件

当你的工程足够大，使用的组件足够多时，是时候需要考虑下性能问题了，因为一开始把所有的组件都加载是没必要的一笔开销。好在Vue.js允许将组件定义为一个工厂函数，动态地解析组件。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。示例如下：



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。也可以调用reject(reason)指示加载失败。上例中的setTimeout只是为了演示异步，具体的下载逻辑可以由自己决定，比如把组件选项写成一个组件配置对象，通过Ajax来请求，然后调用resolve传入组件配置对象。

# 其他

## $nextTick

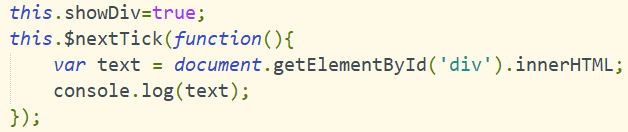
存在场景：有一个<div>，默认用v-if将它隐藏，点击一个按钮后，改变v-if的值，让该div显示出来，同时拿到该div中的文本内容。如果v-if的值是false，直接去获取div中的内容是获取不到的，因为此时div还没有被创建出来，那么应该是点击按钮后，改变v-if的值为true，div才会被创建，此时再去获取div中的内容，代码如下：



正常情况下，当点击按钮会在后台打印div中的内容，但是它没有打印div中的内容，而是抛出一个错误：Cannot read property 'innerHTML' of null"，意思是获取不到div元素。这里就涉及到一个重要的概念：异步更新队列。

异步更新队列指定的是：Vue在观察到数据变化时并不是直接更新DOM，而是开启一个队列，并缓冲在同一事件循环中发生的所有数据变化。在缓冲时会去除重复数据，从而避免不必要的计算和DOM操作。然后，在下一个事件tick中，Vue刷新队列并执行（已去重的）工作。所以如果你用一个for循环来动态改变数据100次，其实它只会应用最后一次改变，如果没有这种机制，DOM就要重绘100次，这会是一个很大的开销。Vue会根据当前浏览器环境优先使用原生的Promise.then和MutationObserver，如果都不支持，就会采用setTimeout代替。

知道了Vue异步更新DOM的原理，上例中的报错也能理解了。在执行this.showDiv=true时，div仍然还没有被创建出来，直到下一个Vue事件循环时，才开始创建。$nextTick就是用来知道什么时候DOM更新完成的。所以需要将上例代码修改一下：



这时再点击按钮，就会在控制台打印div的内容了。在理论上，我们应该不用去主动操作DOM，因为Vue的核心思想是数据驱动DOM，但在很多业务里，避免不了会使用一些第三方库，这些基于原生JavaScript的库都有成交和更新及销毁的完整生命周期，与Vue配合使用时，就要利用好$nextTick。

## X-Templates

如果你没有使用webpack、gulp等工具，试想一下你的组件模板template的内容很冗长、负载，如果都在JavaScript里拼接字符串，效率是非常低的，因为不能像写HTML那样舒服。为此，Vue提供了另一种定义组件模板的方式，在<script>标签里使用text/x-template类型，并且指定一个id，将这个id赋给template。示例如下：

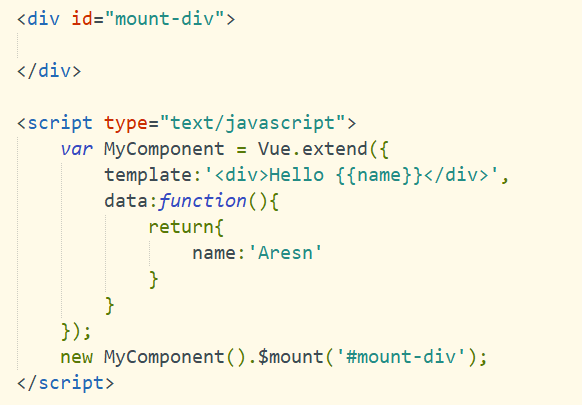


在<script>标签里，你可以愉快地写HTML代码，不用考虑换行的问题。很多刚接触的Vue的新手都会非常喜欢这个功能。不过，Vue的初衷并不是滥用它，因为它将模板和组件的其他定义隔离了。在进阶篇汇总，将会介绍如何使用webpack来编译.vue的单文件，从而优雅地解决HTML书写的问题。

## 手动挂载实例

前面的学习中，所创建的Vue实例都是通过new Vue( )的形式来创建的。在一些特殊的情况下，我们需要动态地去创建Vue实例。对此，Vue提供了Vue.extend和@mount来手动挂载一个Vue实例。

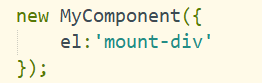
Vue.extend是基础Vue构造器，创建一个Vue的“子类”，参数是一个包含组件选项的对象。如果Vue实例在实例化时没有收到el选项，该实例就处于“未挂载”状态，没有关联DOM元素，此时可以使用$mount( )手动地挂载一个未挂载的实例。这个方法返回实例自身，因而可以链式调用其他实例方法。示例如下：



运行上例后，id为mount-div的div元素会被替换为组件MyComponent的template的内容：



除了上例中手动挂载的写法外，以下两种写法也是可以的：





手动挂载实例（组件）是一种比较极端的高级用法，在业务中几乎用不到，只在开发一些复杂的独立组件时可能会使用，所以只做了解即可。

# 组件实战

## 数字输入框组件

## 标签页组件