目录

[第一节 单页面应用程序 1](#_Toc22510)

[1.什么是单页面应用程序 1](#_Toc25570)

[2.单页面应用程序的特点 1](#_Toc29248)

[3.单页面应用程序的优点 2](#_Toc4906)

[4.单页面应用程序的缺点 2](#_Toc25676)

[5.如何快速创建Vue的单页面应用程序（SPA）项目 3](#_Toc19829)

[第二节 vite的基本使用 4](#_Toc21438)

[1.创建vite的项目 4](#_Toc13641)

[2.梳理项目的结构 4](#_Toc18016)

[3.vite项目的运行流程 5](#_Toc31012)

[3.1 在App.vue中编写模板结构 5](#_Toc22833)

[3.2 在index.html中预留el区域 6](#_Toc4271)

[3.3 在main.js中进行渲染 6](#_Toc24883)

[第三节 组件化开发思想 7](#_Toc20706)

[1.什么是组件化开发 7](#_Toc19343)

[2.组件化开发的好处 7](#_Toc30217)

[3.Vue中的组件化开发 7](#_Toc8)

[第四节 Vue组件的构成 7](#_Toc31480)

[1.Vue组件组成构成 7](#_Toc24529)

[2.组件的template节点 8](#_Toc3837)

[2.1 在template中使用指令 8](#_Toc3135)

[2.2 在template中定义根节点 9](#_Toc18516)

[3.组件的script节点 9](#_Toc15350)

[3.1 script中的name节点 10](#_Toc24348)

[3.2 script中的data节点 10](#_Toc23161)

[3.3 组件中的data必须是函数 11](#_Toc23844)

[3.4 script中的methods节点 11](#_Toc19786)

[4.组件的style节点 12](#_Toc20754)

[4.1 让style中支持less语法 12](#_Toc21245)

[第五节 组件的基本使用 13](#_Toc18340)

[1.组件的注册 13](#_Toc21480)

[1.1 注册组件的两种方式 13](#_Toc9248)

[1.2 全局注册组件 14](#_Toc31067)

[1.3 使用全局注册组件 14](#_Toc31028)

[1.4 局部注册组件 15](#_Toc18302)

[1.5 全局注册和局部注册的区别 15](#_Toc19917)

[1.6 组件注册时名称的大小写 16](#_Toc5967)

[1.7 通过name属性注册组件 16](#_Toc21934)

[2.组件之间的样式冲突问题 17](#_Toc31056)

[2.1 思考：如何解决组件样式冲突的问题 17](#_Toc9176)

[2.2 style节点的scoped属性 18](#_Toc3815)

[2.3 /deep/样式穿透 18](#_Toc19702)

[3.组件的props 19](#_Toc6016)

[3.1 什么是组件的props 19](#_Toc25200)

[3.2 在组件中声明props 19](#_Toc6120)

[3.3 无法使用未声明的props 20](#_Toc13011)

[3.4 动态绑定props的值 20](#_Toc30092)

[3.5 props的大小写命名 20](#_Toc25255)

[4.Class与Style绑定 21](#_Toc27759)

[4.1 动态绑定HTML的class 21](#_Toc18685)

[4.2 以数组语法绑定HTML的class 21](#_Toc15896)

[4.3 以对象语法绑定HTML的class 22](#_Toc5413)

[4.4 以对象语法绑定内联的style 22](#_Toc10452)

[第六节 props验证 23](#_Toc12151)

[1.什么是props验证 23](#_Toc29166)

[2.对象类型的props节点 23](#_Toc5401)

[3.props验证 23](#_Toc25299)

[3.1 基础的类型检查 24](#_Toc12062)

[3.2 多个可能的类型 24](#_Toc22595)

[3.3 必填项校验 25](#_Toc26486)

[3.4 属性默认值 25](#_Toc15342)

[3.5 自定义验证函数 26](#_Toc4739)

[第七节 计算属性 27](#_Toc17468)

[1.什么是计算属性 27](#_Toc14939)

[2.如何声明计算属性 27](#_Toc29453)

[3.计算属性的使用注意点 27](#_Toc246)

[4.计算属性vs方法 28](#_Toc21424)

[第八节 自定义事件 29](#_Toc14546)

[1.什么是自定义事件 29](#_Toc14696)

[2.自定义事件的3个使用步骤 29](#_Toc4067)

[2.1 声明自定义事件 30](#_Toc2864)

[2.2 触发自定义事件 30](#_Toc18110)

[2.3 监听自定义事件 31](#_Toc20134)

[3.自定义事件传参 31](#_Toc32354)

[第九节 组件上的v-model 32](#_Toc29508)

[1.为什么需要在组件上使用v-model 32](#_Toc14607)

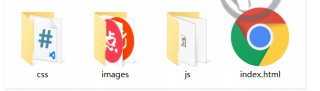
[2.在组件上使用v-model的步骤 32](#_Toc10310)

**第一节 单页面应用程序**

**1.什么是单页面应用程序**

单页面应用程序（英文名: Single Page Application)简称SPA，顾名思义，指的是一个web网站中只有唯一的一个HTML页面，所有的功能与交互都在这唯一的一个页面内完成。

例如资料中的这个Demo项目:





**2.单页面应用程序的特点**

单页面应用程序将所有的功能局限于一个web页面中，仅在该web质面初始化时加载相应的资源（HTML、JavaScript和CSS)。

一旦页面加载完成了，SPA不会因为用户的操作而进行页面的重新加载或跳转。而是利用JavaScript动态地变换HTML的内容，从而实现页面与用户的交互。

**3.单页面应用程序的优点**

SPA单页面应用程序最显著的3个优点如下:

①良好的交互体验

·单页应用的内容的改变不需要重新加载整个页面

·获取数据也是通过Ajax异步获取

·没有页面之间的跳转，不会出现“白屏现象”

②良好的前后端工作分离模式

·后端专注于提供API接口，更易实现API接口的复用

·前端专注于页面的渲染，更利于前端工程化的发展

③减轻服务器的压力

·服务器只提供数据，不负责页面的合成与逻辑的处理，吞吐能力会提高几倍

**4.单页面应用程序的缺点**

任何一种技术都有自己的局限性，对于SPA单页面应用程序来说，主要的缺点有如下两个:

①首屏加载慢

·路由懒加载

·代码压缩

·CDN加速

·网络传输压缩

②不利于SEO

·SSR服务器端渲染

**5.如何快速创建Vue的单页面应用程序（SPA）项目**

vue官方提供了两种快速创建工程化的SPA项目的方式:

①基于vite创建SPA项目

②基于vue-cli创建SPA项目



**第二节 vite的基本使用**

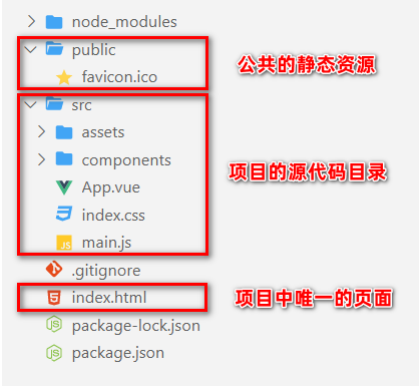
**1.创建vite的项目**

按照顺序执行如下的命令，即可基于vite创建vue 3.x的工程化项目



**2.梳理项目的结构**

使用vite创建的项目结构如下:

其中:

·node\_modules目录用来存放第三方依赖包

·public是公共的静态资源目录

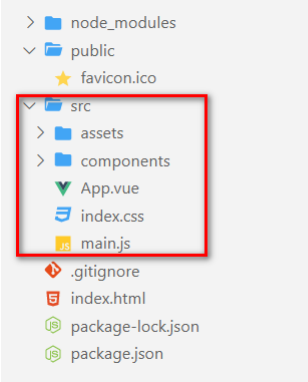
·src是项目的源代码目录（程序员写的所有代码都要放在此目录下)

·gitignore是Git的忽略文件

·index.html是SPA单页面应用程序中唯一的HTML页面

·package.json是项目的包管理配置文件

在src这个项目源代码目录之下，包含了如下的文件和文件夹:

其中:

·assets目录用夹存放项目中所有的静态资源文件(css、fonts等)

·components目录用来存放项目中所有的自定义组件

·App.vue是项目的根组件

·index.css是项目的全局样式表文件

·main.js是整个项目的打包入口文件

**3.vite项目的运行流程**

在工程化的项目中，vue要做的事情很单纯:**通过main.js 把 App.vue渲染到index.html的指定区域中。**

其中:

①App.vue用来编写待渲染的模板结构

②index.html中需要预留一个el区域

③main.js 把 App.vue渲染到了index.html所预留的区域中

**3.1 在App.vue中编写模板结构**

清空App.vue的默认内容，并书写如下的模板结构:



**3.2 在index.html中预留el区域**

打开index.html页面，确认预留了el区域:



**3.3 在main.js中进行渲染**

按照vue 3.x的标准用法，把 App.vue中的模板内容渲染到 index.html页面的el区域中:



**第三节 组件化开发思想**

**1.什么是组件化开发**

组件化开发指的是∶根据封装的思想，把页面上可重用的部分封装为组件，从而方便项目的开发和维护。例如: http://www.ibootstrap.cn/所展示的效果，就契合了组件化开发的思想。用户可以通过拖拽组件的方式，快速生成一个页面的布局结构。

**2.组件化开发的好处**

前端组件化开发的好处主要体现在以下两方面:

·提高了前端代码的复用性和灵活性

·提升了开发效率和后期的可维护性

**3.Vue中的组件化开发**

vue是一个完全支持组件化开发的框架。vue中规定组件的后缀名是ue。之前接触到的App.vue文件本质上就是一个vue 的组件。

**第四节 Vue组件的构成**

**1.Vue组件组成构成**

每个.vue组件都由3部分构成，分别是:

·template ->组件的模板结构

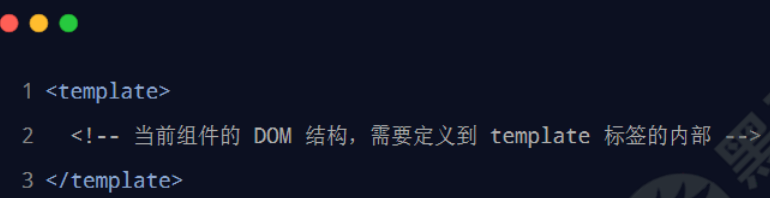
·script ->组件的JavaScript行为

·style ->组件的样式

其中，每个组件中必须包含template模板结构，而script行为和style样式是可选的组成部分。

**2.组件的template节点**

vue规定:每个组件对应的模板结构，需要定义到<template>节点中。



注意: <template>是vue提供的容器标签，只起到包裹性质的作用，它不会被渲染为真正的DOM元素。

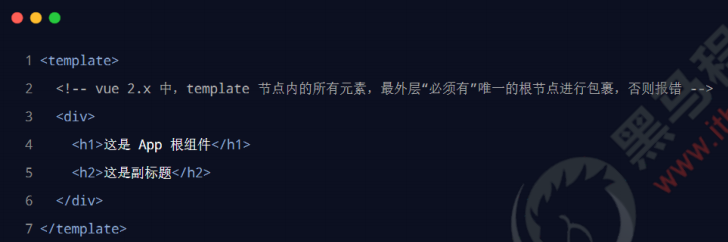
**2.1 在template中使用指令**

在组件的<template>节点中，支持使用前面所学的指令语法，来辅助开发者渲染当前组件的DOM结构。代码示例如下:

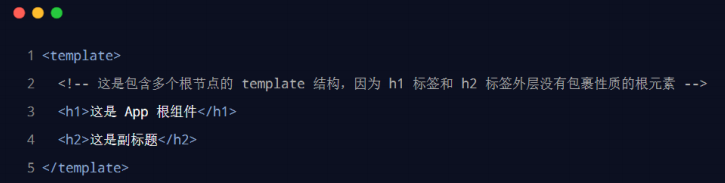


**2.2 在template中定义根节点**

在vue 2.x的版本中，<template>节点内的DOM结构仅支持单个根节点:



但是，在vue 3.x的版本中，<template>中支持定义多个根节点:



**3.组件的script节点**

vue规定:组件内的<script>节点是可选的，开发者可以在<script茄点中封装组件的JavaScript业务逻辑。<script >节点的基本结构如下:

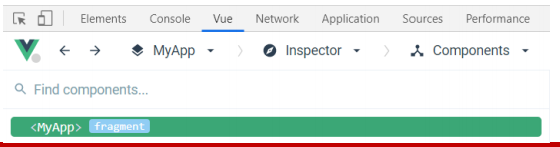


**3.1 script中的name节点**

可以通过name节点为当前组件定义一个名称:



在使用vue-devtools进行项目调试的时候，自定义的组件名称可以清晰的区分每个组件:



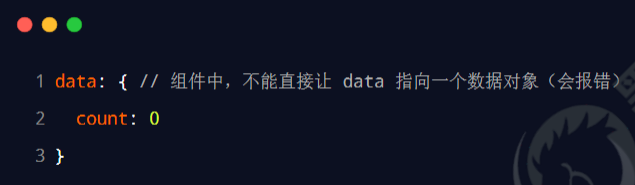
**3.2 script中的data节点**

vue组件渲染期间需要用到的数据，可以定义在data节点中:



**3.3 组件中的data必须是函数**

vue 规定:组件中的data必须是一个函数，不能直接指向一个数据对象。因此在组件中定义data数据节点时，下面的方式是错误的:



**3.4 script中的methods节点**

组件中的事件处理函数，必须定义到methods节点中，示例代码如下:



**4.组件的style节点**

vue规定:组件内的<style>节点是可选的，开发者可以在<style>节点中编写样式美化当前组件的UI结构。<script >节点的基本结构如下:

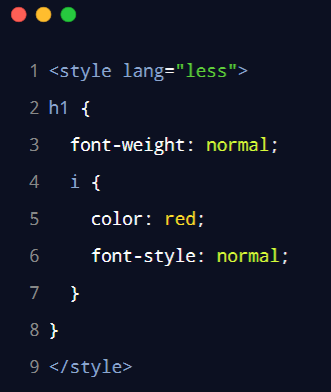


其中<style>标签上的lang="css"属性是可选的，它表示所使用的样式语言。默认只支持普通的css语法，可选值还有less、scss等。

**4.1 让style中支持less语法**

如果希望使用less语法编写组件的style样式，可以按照如下两个步骤进行配置:①运行npm install less -D命令安装依赖包，从而提供less语法的编译支持

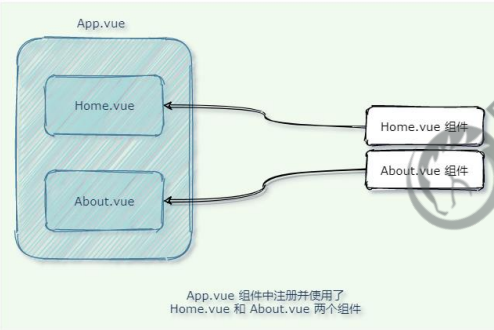
②在<style>标签上添加lang="less"属性，即可使用less语法编写组件的样式



**第五节 组件的基本使用**

**1.组件的注册**

组件之间可以进行相互的引用，例如:



vue中组件的引用原则:先注册后使用。

**1.1 注册组件的两种方式**

vue中注册组件的方式分为“全局注册”和“局部注册”两种，其中:

·被全局注册的组件，可以在全局任何一个组件内使用

·被局部注册的组件，只能在当前注册的范围内使用





**1.2 全局注册组件**



**1.3 使用全局注册组件**

使用app.component( )方法注册的全局组件，直接以标签的形式进行使用即可，例如:



**1.4 局部注册组件**



**1.5 全局注册和局部注册的区别**

·被全局注册的组件，可以在全局任何一个组件内使用

·被局部注册的组件，只能在当前注册的范围内使用

应用场景:

·如果某些组件在开发期间的使用频率很高，推荐进行全局注册;

·如果某些组件只在特定的情况下会被用到，推荐进行局部注册。

**1.6 组件注册时名称的大小写**

在进行组件的注册时，定义组件注册名称的方式有两种:

①使用kebab-case命名法（俗称短横线命名法，例如my-swiper和amy-search)

②使用PascalCase命名法（俗称帕斯卡命名法或大驼峰命名法,例如 MySwiper和 MySearch)

短横线命名法的特点:

·必须严格按照短横线名称进行使用

帕斯卡命名法的特点:

·既可以严格按照帕斯卡名称进行使用，又可以转化为短横线名称进行使用

注意:在实际开发中，推荐使用帕斯卡命名法为组件注册名称，因为它的适用性更强。

**1.7 通过name属性注册组件**

在注册组件期间，除了可以直接提供组件的注册名称之外，还可以把组件的name属性作为注册后组件的名称,示例代码如下:



**2.组件之间的样式冲突问题**

默认情况下，写在.vue组件中的样式会全局生效，因此很容易造成多个组件之间的样式冲突问题。导致组件之间样式冲突的根本原因是:

①单页面应用程序中，所有组件的DOM结构，都是基于唯一的index.html页面进行呈现的

②每个组件中的样式，都会影响整个index.html页面中的DOM元素

**2.1 思考：如何解决组件样式冲突的问题**

为每个组件分配唯一的自定义属性，在编写组件样式时，通过属性选择器来控制样式的作用域，示例代码如下:

![](NY7CW@}T]~L%JHFR~O8SF](data:image/png;base64,)

**2.2 style节点的scoped属性**

为了提高开发效率和开发体验,vue为style节点提供了scoped属性。从而防止组件之间的样式冲突问题:



**2.3 /deep/样式穿透**

如果给当前组件的style 节点添加了scoped属性，则当前组件的样式对其子组件是不生效的。如果想让某些样式对子组件生效，可以使用/deep/深度选择器。





注意:/deep/是vue2.x中实现样式穿透的方案。在vue3.x中推荐使用:deep( )替/deep/。

**3.组件的props**

为了提高组件的复用性，在封装vue组件时需要遵守如下的原则:

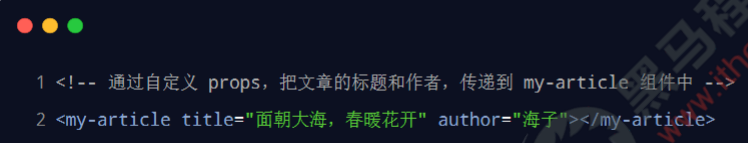
·组件的DOM结构、Style样式要尽量复用

·组件中要展示的数据，尽量由组件的使用者提供

为了方便使用者为组件提供要展示的数据，vue组件提供了props的概念。

**3.1 什么是组件的props**

props是组件的自定义属性，组件的使用者可以通过props把数据传递到子组件内部，供子组件内部进行使用。代码示例如下:



props的作用:父组件通过props向子组件传递要展示的数据。

props的好处:提高了组件的复用性。

**3.2 在组件中声明props**

在封装vue组件时，可以把动态的数据项声明为props自定义属性。自定义属性可以在当前组件的模板结构中被直接使用。示例代码如下:



**3.3 无法使用未声明的props**

如果父组件给子组件传递了未声明的props属性，则这些属性会被忽略。无法被子组件使用，示例代码如下:

**3.4 动态绑定props的值**

可以使用v-bind属性绑定的形式，为组件动态绑定props的值，示例代码如下:



**3.5 props的大小写命名**

组件中如果使用“camelCase(驼峰命名法)”声明了props属性的名称。则有两种方式为其绑定属性的值:



**4.Class与Style绑定**

在实际开发中经常会遇到动态操作元素样式的需求。因此，vue允许开发者通过v-bind属性绑定指令，为元素动态绑定class属性的值和行内的style样式。

**4.1 动态绑定HTML的class**

可以通过三元表达式，动态的为元素绑定class的类名。示例代码如下;



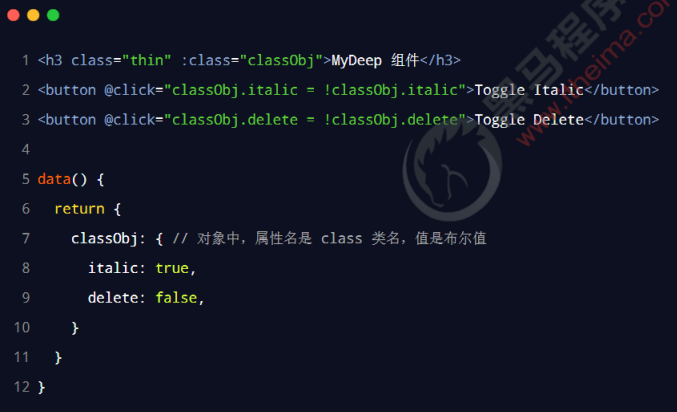
**4.2 以数组语法绑定HTML的class**

如果元素需要动态绑定多个class的类名，此时可以使用数组的语法格式:



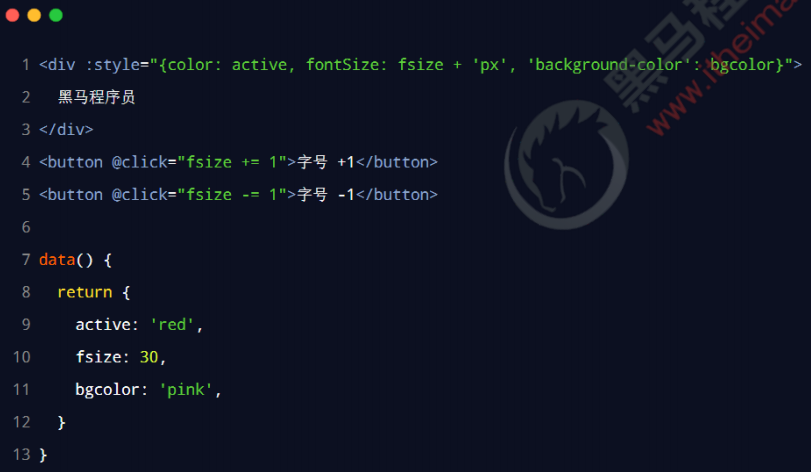
**4.3 以对象语法绑定HTML的class**

使用数组语法动态绑定class 会导致模板结构臃肿的问题。此时可以使用对象语法进行简化:



**4.4 以对象语法绑定内联的style**

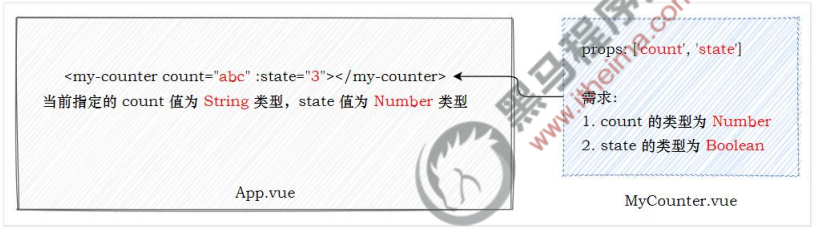
:style的对象语法十分直观——看着非常像CSS，但其实是一个JavaScript对象。CSS property名可以用驼峰式(camelCase)或短横线分隔(kebab-case，记得用引号括起来)来命名:



**第六节 props验证**

**1.什么是props验证**

指的是:在封装组件时对外界传递过来的props数据进行合法性的校验,从而防止数据不合法的问题。



使用数组类型的props节点的缺点∶无法为每个prop 指定具体的数据类型。

**2.对象类型的props节点**

使用对象类型的props 节点，可以对每个prop进行数据类型的校验，示意图如下:



**3.props验证**

对象类型的props节点提供了多种数据验证方案，例如:

①基础的类型检查

②多个可能的类型

③必填项校验

④属性默认值

⑤自定义验证函数

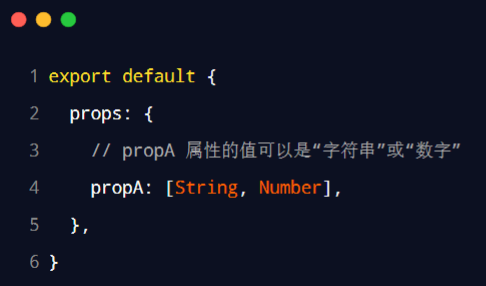
**3.1 基础的类型检查**

可以直接为组件的prop属性指定基础的校验类型，从而防止组件的使用者为其绑定错误类型的数据:



**3.2 多个可能的类型**

如果某个prop属性值的类型不唯一，此时可以通过数组的形式，为其指定多个可能的类型，示例代码如下:



**3.3 必填项校验**

如果组件的某个prop属性是必填项，必须让组件的使用者为其传递属性的值。此时，可以通过如下的方式将其设置为必填项:



**3.4 属性默认值**

在封装组件时，可以为某个prop属性指定默认值。示例代码如下:



**3.5 自定义验证函数**

在封装组件时，可以为prop属性指定自定义的验证函数，从而对prop属性的值进行更加精确的控制:



**第七节 计算属性**

**1.什么是计算属性**

计算属性本质上就是一个function函数，它可以实时监听data中数值的变化，并return一个计算后的新值,供组件渲染DOM时使用。

**2.如何声明计算属性**

计算属性需要以function函数的形式声明到组件的computed选项中，示例代码如下:



注意:计算属性侧重于得到一个计算的结果，因此计算属性中必须有return返回值!

**3.计算属性的使用注意点**

①计算属性必须定义在computed节点中

②计算属性必须是一个function函数

③计算属性必须有返回值

④计算属性必须当做普通属性使用

**4.计算属性vs方法**

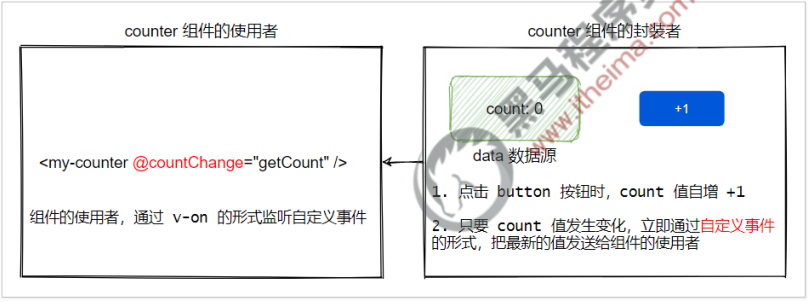
相对于方法来说，计算属性会缓存计算的结果，只有计算属性的依赖项发生变化时，才会重新进行运算。因此计算属性的性能更好:



**第八节 自定义事件**

**1.什么是自定义事件**

在封装组件时，为了让组件的使用者可以监听到组件内状态的变化，此时需要用到组件的自定义事件。



**2.自定义事件的3个使用步骤**

在封装组件时:

①声明自定义事件

②触发自定义事件

在使用组件时:

③监听自定义事件

**2.1 声明自定义事件**

开发者为自定义组件封装的自定义事件，必须事先在emits 节点中声明，示例代码如下:



**2.2 触发自定义事件**

在emits节点下声明的自定义事件，可以通过this.$emit('自定义事件的名称')方法进行触发，示例代码如下:



**2.3 监听自定义事件**

在使用自定义的组件时，可以通过v-on的形式监听自定义事件。示例代码如下:



**3.自定义事件传参**

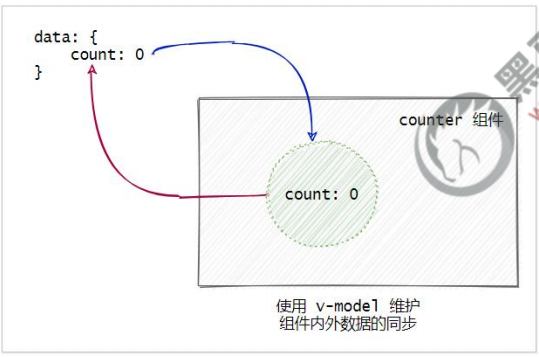
在调用this.$emit( )方法触发自定义事件时，可以通过第2个参数为自定义事件传参，示例代码如下:



**第九节 组件上的v-model**

**1.为什么需要在组件上使用v-model**

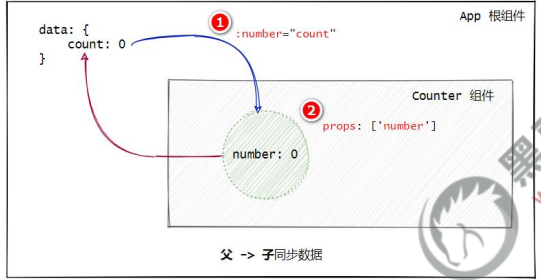
v-model是双向数据绑定指令，当需要维护组件内外数据的同步时,可以在组件上使用v-model指令。示意图如下:



·外界数据的变化会自动同步到counter组件中

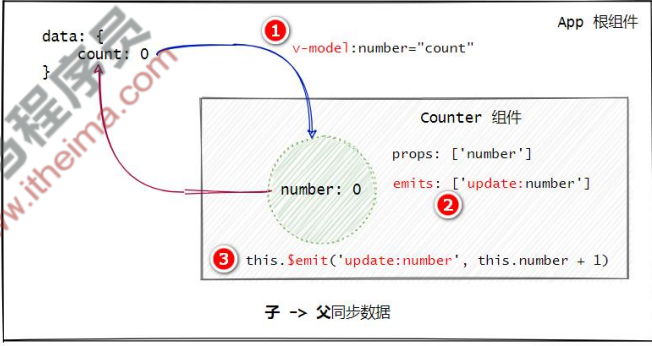
·counter组件中数据的变化，也会自动同步到外界

**2.在组件上使用v-model的步骤**



①父组件通过v-bind:属性绑定的形式，把数据传递给子组件

②子组件中，通过props接收父组件传递过来的数据



①在v-bind:指令之前添加v-model指令

②在子组件中声明emits自定义事件，格式为update:XXX

③调用$emit( )触发自定义事件，更新父组件中的数据