爱创课堂前端培训

# VueJS

第3天课堂笔记（本课程共9天）

班级：北京前端训练营23期

讲师：张容铭

日期：2019年7月25日

张容铭老师

微博：@张容铭\_YYQH

QQ : 286031482

E-mail : yuye\_qinghe@qq.com

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

[VueJS 1](#_Toc2194)

[复习 3](#_Toc11383)

[一、Vue 4](#_Toc18366)

[1.1自定义指令 4](#_Toc18614)

[1.1.1使用自定义指令 4](#_Toc22100)

[1.1.2自定义指令 5](#_Toc8468)

[1.2组件 5](#_Toc13946)

[1.2.1使用组件 5](#_Toc25612)

[1.2.2动态组件 8](#_Toc26103)

[下午复习 9](#_Toc1149)

[1.3组件生命周期 10](#_Toc12169)

[1.3.1创建期 10](#_Toc60)

[1.3.2存在期 10](#_Toc10127)

[1.3.3销毁期 10](#_Toc30851)

[1.3.4 keep-alive 10](#_Toc14156)

[1.4组件通信 13](#_Toc4790)

[1.4.1父组件向子组件通信 13](#_Toc29464)

[1.4.2 $parent 14](#_Toc3142)

[1.4.3自定义事件 15](#_Toc6961)

[1.4.4子组件向父组件通信 17](#_Toc703)

[1.4.5兄弟组件间通信 20](#_Toc25620)

[1.6插槽 20](#_Toc14666)

[1.6.1使用插槽 20](#_Toc19066)

[1.6.2插槽作用域 21](#_Toc12066)

# 复习

事件 v-on:click=”fn” @

methods 参数 $event

修饰符 @click.修饰符 stop, prevent, once, self

鼠标 left, right, middle

系统键 shift, alt, ctrl, meta

键盘 esc, tab, space, enter, delete, a-z

绑定类 :class=”[] | {} | str”

绑定样式 :style=”[] | {} | str”

模板指令

v-if v-else， v-else-if

v-show

v-for=”item in data” v-for=”(item, index) in data” 数组， 对象， 数字 key

template

过渡 transition name=“demo”

.demo-enter, .demo-enter-to, .demo-enter-active, .demo-leave, .demo-leave-to, .demo-leave-active

transition, animation

before-enter, enter, after-enter, before-leave, leave, after-leave

appear

多元素过渡， mode out-in, in-out

transition-group key

# 一、Vue

## 1.1自定义指令

我们已经学过的指令：v-bind, v-model, v-text, v-html, v-once, v-cloak, v-on, v-if, v-else-if, v-else, v-show, v-for...

指令是对元素的拓展，使其具有一定的行为特征。

内置的指令是有限的，为了给元素拓展更多的功能，我们要自定义指令。

### 1.1.1使用自定义指令

共分两步

第一步 在页面中使用。

指令都是以v-开头的。

要遵守html规范，字母小写，横线分割单词

指令的属性值是js环境。

第二步 在js中，我们通过Vue.directive方法自定义指令

第一个参数表示指令名称

要省略v-前缀，并且要转化成驼峰式命名

第二个参数表示指令对象或者指令函数

如果是指令对象

bind 绑定时候执行的方法

update 更新时候执行的方法

unbind 解绑时候执行的方法

inserted 插入到页面中的时候，执行的方法

componentUpdated 组件更新的时候，执行的方法

如果是指令函数

相当于用指令函数代替上面的方法

不论是指令函数还是指令对象，参数常见的有四个

第一个是指令所在的DOM元素（源生的DOM元素）

第二个是指令对象，包括指令的一些信息

表达式，当前的值，上一个指令，指令名称等等

第三个是当前虚拟DOM对象

第四个是上一个虚拟DOM对象

注意：当多次使用指令的时候，一个指令的属性值发生改变，所有的指令的方法都会执行。

我们可以判断当前指令的属性值与上一个次的属性值是否相等

相当：说明数据没有改变，不用执行

不相等：说明数据发生改变，要执行

注意：跟filter一样，不要解构，并且在vue实例化之前定义。

### 1.1.2自定义指令

我们定义v-ickt-html指令。来模拟v-html指令

我们定义v-ickt-once指令。来模拟v-once指令

我们定义v-ickt-show指令。来模拟v-show指令

|  |
| --- |
| 1. <p> 2. <label for="">用户名</label> 3. <input type="text" v-model="username"> 4. <span v-ickt-check="username" test="^\w{4,8}$" error-text="用户名是4到8位的字母数字下划线"></span> 5. </p> 6. // 更新的时候检测 7. Vue.directive('icktCheck', { 8. // 更新 9. update(dom, obj) { 10. // 优化 11. if (obj.value !== obj.oldValue) { 12. // 数据检测 13. // 获取要求 14. let test = dom.getAttribute('test'); 15. // 创建正则 16. let reg = new RegExp(test); 17. // 校验数据 18. if(reg.test(obj.value)) { 19. // 合法，不显示错误提示 20. dom.innerHTML = ''; 21. } else { 22. // 不合法，显示提示内容 23. dom.innerHTML = dom.getAttribute('error-text') 24. } 25. } 26. } 27. }) |

## 1.2组件

在html中，组件是一段可以被复用的结构代码。

在css中，组件是一段可以被复用的样式代码。

在js中，组件是一段可以被复用的功能代码。

在vue中也有组件，组件是一段可以被复用的包含结构，样式和功能的代码。

vue中的组件是一个完整体。

### 1.2.1使用组件

在vue中使用组件分成三步

第一步 在html中使用组件。

要遵守html规范：字母小写，横线分割单词

注意：首字母不区分大小写。

第二步 定义组件类。通过Vue.extend方法定义。

extend方法是一个继承方法，说明组件继承了Vue类，

所以vue实例化对象也可以看成是一个组件。

extend方法接收一个参数对象，跟实例化vue是相同的。

methods 定义方法的

watch 监听数据的

computed 计算属性数据

...

也有一些特殊属性

data 定义模型数据的，是一个方法

返回值是对象，才是真正绑定的数据。

this指向组件实例化对象，但是无法访问模型中的数据，

template 定义模板的，与el属性类似

el属性：将该选择器对应元素的内容作为模板

template：属性值就是模板

属性值有两种方式

第一种：模板字符串（定义在js中了）

第二种：选择器，类似el属性

获取该选择器的内容作为模板，我们通常在模板元素中定义

html中，有两种方式定义模板

第一种 模板脚本标签 <script type=”text/template”>

第二种 html5提供的模板标签 template

vue建议我们用template标签定义

注意：定义的模块，最外层只能有且只有一个根元素

第三步 注册组件

注册的目的，将组件渲染到页面中

有两种注册方式

第一种 全局注册

Vue.component方法注册

第一个参数表示组件名称

驼峰式命名，首字母不区分大小写

第二个参数表示组件类

所有组件的模板中都可以使用该组件

第二种 局部注册

通过组件的components属性注册

key表示组件名称

驼峰式命名，首字母不区分大小写

value表示组件类

只有在该组件中，可以使用，其它的组件不能使用

注意：组件是一个完整独立的个体，彼此之间方法，数据等都不会共享。

父子关系

vue实例化对象可以看成是组件，

因此在vue实例化对象中使用的组件可以看成是vue实例化对象的子组件

vue实例化对象可以看成是父组件。

|  |
| --- |
| 1. <!-- 1 使用组件 --> 2. <home></home> 3. // 2 定义组件类 4. let Home = Vue.extend({ 5. // 模板 6. // template: '<h1>home component</h1>' 7. // 模板：选择器 8. template: '#tpl', 9. // 数据 10. data() { 11. // 此时数据还没有绑定呢，所以无法访问数据 12. console.log(this, this.msg) 13. return { 14. msg: 'home msg' 15. } 16. }, 17. // 计算属性数据 18. computed: { 19. // msg大写 20. dealMsg() { 21. return this.msg.toUpperCase() 22. } 23. }, 24. // 方法 25. methods: { 26. // 点击h1 27. clickH1() { 28. console.log(this, 'click home h1') 29. } 30. } 31. }) 32. // 3 注册 33. // 全局注册 34. // Vue.component('home', Home) 35. // Vue.component('Home', Home) 36. // 驼峰式命名 37. // Vue.component('icktHome', Home) 38. // Vue.component('IcktHome', Home) 39. // vue实例化对象 40. new Vue({ 41. // 局部注册 42. components: { 43. // Home: Home 44. // 省略 45. Home 46. }, 47. // 容器 48. el: '#app', 49. // 数据 50. data: { 51. msg: 'hello' 52. }, 53. // 计算属性数据 54. computed: { 55. // msg大写 56. dealMsg(v) { 57. return this.msg.toUpperCase(); 58. } 59. }, 60. // 方法 61. methods: { 62. // 点击h1 63. clickH1() { 64. console.log('click h1') 65. } 66. } 67. }) |

### 1.2.2动态组件

在使用组件的时候，一类元素对应一个组件，

如果我们希望一个元素可以渲染多个组件，我们可以使用动态组件元素：component。

通过is属性定义渲染的组件。

想动态设置is属性值，要使用v-bind指令。

|  |
| --- |
| 1. <component :is="page"></component> |

### 下午复习

自定义指令

1 v-ickt-html

2 Vue.directive(name, { bind, update, unbind, inserted, componentUpdated } | fn)

dom, obj, vNode, oldVNode

value !== oldValue

自定义组件

1 home

2 Vue.extend({ methods, computed, watch, template, data() { return } })

3 Vue.component(key, value) components: {}

动态组件 component :is=””

## 1.3组件生命周期

为了说明组件创建，存在，销毁的过程，vue提供了组件生命周期技术。

共分三大周期：创建期，存在期，销毁期

### 1.3.1创建期

在页面中渲染组件的时候，我们就要创建这个组件，此时组件会进入存在期，共分四个阶段：

beforeCreate 组件即将创建，此时数据，自定义事件等还没有绑定

created 组件创建完成：此时组件具有了数据以及自定义事件等，还没有渲染。

beforeMount 组件即将渲染：此时要确定容器元素以及编译模板。

mounted 组件渲染完成：此时组件已经渲染到页面中（上树）。

### 1.3.2存在期

当组件中的数据发生改变，组件就会进入存在期，共分两个阶段。

beforeUpdate 组件即将更新：此时数据更新了，视图尚未更新。

updated 组件更新完成：此时数据与视图都已经更新了

一次数据更新的结束，存在期仍然继续，等待数据下一次更新。

### 1.3.3销毁期

当组件从页面中删除的时候，组件进入销毁期。共分两个阶段。

beforeDestroy 组件即将销毁：此时数据，自定义事件，子组件等仍然存储。

destroyed 组件销毁完成：此时数据，自定义事件，子组件都被移除了。

组件一旦被销毁，就再也无法访问该组件。

### 1.3.4 keep-alive

当组件从页面中删除的时候，组件进入销毁期，如果不想让组件被销毁。可以在外部包裹keep-alive组件。

此时组件从页面中删除，但是没有被销毁。而是缓存在内存中，需要的时候，从内存中获取。

注意：隐藏的时候，会保留最后一个状态，显示的时候会将最后一个状态显示。

为了说明组件显示或者隐藏，为组件拓展了两个方法：

activated 组件被激活，显示出来了。

deactivated 组件被禁用，消失不存在了。

以上所有周期方法，this都指向组件实例化对象，没有参数的。

创建期与销毁期的方法，在组件的一生中只执行一次

存在期的方法可以多次执行。



|  |
| --- |
| 1. let Home = Vue.extend({ 2. // 模板 3. // template: '<h1>home component</h1>' 4. // 模板：选择器 5. template: '#tpl', 6. // 数据 7. data() { 8. // 此时数据还没有绑定呢，所以无法访问数据 9. // console.log(this, this.msg) 10. return { 11. msg: 'home msg' 12. } 13. }, 14. // 计算属性数据 15. computed: { 16. // msg大写 17. dealMsg() { 18. return this.msg.toUpperCase() 19. } 20. }, 21. // 方法 22. methods: { 23. // 点击h1 24. clickH1() { 25. console.log(this, 'click home h1') 26. } 27. }, 28. // 创假期 29. beforeCreate() { 30. console.log(111, 'beforeCreate', this, arguments) 31. }, 32. created() { 33. console.log(222, 'created', this, arguments) 34. }, 35. beforeMount() { 36. console.log(333, 'beforeMount', this, arguments) 37. }, 38. mounted() { 39. console.log(444, 'mounted', this, arguments) 40. }, 41. // 存在期 42. beforeUpdate() { 43. console.log(555, 'beforeUpdate', this, arguments) 44. }, 45. updated() { 46. console.log(666, 'updated', this, arguments) 47. }, 48. // 销毁期 49. beforeDestroy() { 50. console.log(777, 'beforeDestroy', this, arguments) 51. }, 52. destroyed() { 53. console.log(888, 'destroyed', this, arguments) 54. }, 55. // 使用keep-alive 56. activated() { 57. console.log(999, 'activated', this, arguments) 58. }, 59. deactivated() { 60. console.log(123, 'deactivated', this, arguments) 61. }, 62. }) |

## 1.4组件通信

组件是完整独立的，因此数据，方法等是不会共享的。

想实现组件之间数据共享，我们就要使用组件间通信技术。

通信具有方向性。

父组件向子组件通信

子组件向父组件通信

兄弟组件间通信。

### 1.4.1父组件向子组件通信

父组件向子组件通信分成两步

第一步 为子组件元素添加属性数据。

要遵守html命名规范：字母小写，横线分割单词。

属性值是默认是字符串，想传递变量，要使用v-bind指令，动态传递。

第二步 在子组件中，通过props属性接收数据。

命名规范：转化成驼峰式命名。

属性值有两种情况

第一种：简单接收数据

属性值是数组，每一个成员代表一个属性数据。

第二种：对接收的数据做校验

属性值是对象

key表示接收的属性名称。

value

是构造函数：String，Number等，代表属性数据的类型

是数组，每一个成员是一个构造函数，多种类型限制。

是对象

type 定义数据类型

required 是否是必须

default 默认值

可以是数据

可以是函数，函数返回的值就是默认值

validator 校验方法

返回值是布尔值，表示是否校验通过

true 校验通过

false 校验不通过

props中定义的属性与data中定义的模型数据一样，添加给vue实例化对象自身，并且设置了特性，

我们可以在模板中使用，也可以在组件中使用。

### 1.4.2 $parent

在子组件实例化对象中，有一个$parent属性，代表其父组件，

我们可以通过$parent访问父组件中的数据，但是工作中不常用，会有耦合问题。

|  |
| --- |
| 1. <!-- 1 传递属性数据 --> 2. <home color="red" :parent-msg="msg"></home> 3. // 定义组件类 4. let Home = Vue.extend({ 5. // 2 接收数据 6. // props: ['color', 'parentMsg'], 7. // 属性校验 8. props: { 9. // 设置类型 10. color: String, 11. // 多种类型 12. // num: [Number, String], 13. num: { 14. // 类型 15. type: Number, 16. // 必填的 17. // required: true, 18. // 默认值 19. // default: 10 20. default() { 21. return 200 22. } 23. }, 24. parentMsg: { 25. // 类型 26. type: String, 27. // 校验方法 28. validator(value) { 29. // console.log(arguments, this) 30. // value长度要大于4位 31. return value.length > 4; 32. } 33. } 34. }, 35. // 数据 36. data() { 37. return { 38. ickt: 'icketang' 39. } 40. }, 41. // 模板 42. template: ` 43. <div> 44. <h1>child: {{color}}--{{parentMsg}}--{{num}}</h1> 45. <h1>parent data: {{$parent.msg}}</h1> 46. </div> 47. `, 48. // 组件创建完成 49. created() { 50. console.log(this) 51. } 52. }) |

### 1.4.3自定义事件

vue也实现了观察者模式。提供了注册事件，发布事件，注销事件等方法。并且将这些方法拓展给vue类的原型上。

$on(type, fn) 订阅消息

type 消息类型

fn 消息回调函数

参数由$emit提供。

$once 单次订阅

$emit(type, ...arg) 发布消息

type 消息类型

arg 从第二个参数开始，表示传递的数据

$off(type, fn) 注销消息

type 消息类型

fn 消息回调函数

注意：组件是完整独立的，因此事件，数据，方法等是不会共享的。

所以当前组件订阅的事件，只能由当前组件发布。

但是，子组件可以通过$parent访问父组件，所以可以通过$parent来向父组件发布消息。

注意：工作中不常用，因为具有耦合问题。

|  |
| --- |
| 1. let Home = Vue.extend({ 2. // 数据 3. data() { 4. return { 5. msg: 'hello' 6. } 7. }, 8. // 模板 9. template: ` 10. <div> 11. <input type="text" v-model="msg" /> 12. <h1>{{msg}}</h1> 13. </div> 14. `, 15. // 组件创建完成 16. created() { 17. // console.log(this) 18. // 子组件中发布消息 19. // this.$emit('ickt', this.msg, 200, false) 20. // 通过父组件再发布消息 21. this.$parent.$emit('ickt', this.msg, 200, false) 22. } 23. }) 24. // vue实例化对象 25. new Vue({ 26. // 注册 27. components: { Home }, 28. // 容器 29. el: '#app', 30. // 数据 31. data: { 32. msg: '' 33. }, 34. // 组件创建完成 35. created() { 36. // 订阅消息 37. this.$on('ickt', (msg, ...arg) => { 38. console.log(msg, arg) 39. // 存储数据 40. this.msg = msg; 41. }) 42. // 父组件可以发布 43. // setTimeout(() => { 44. // // 发布消息 45. // this.$emit('ickt', 'parent data', true, 100) 46. // }, 2000) 47. } 48. }) |

### 1.4.4子组件向父组件通信

作用：子组件中的数据，传递给父组件去使用。

子组件向父组件通信有两种方式：模拟DOM事件，方法传递

#### 1.4.4.1模拟DOM事件

定义DOM事件

@click=”fn”

模拟DOM事件

@ickt-demo=”fn”

实现通信分成四步：

第一步 在父组件的模板中，为子组件元素绑定模拟的DOM事件

@ickt-demo=”fn”

注意：

1 事件名称建议横线分割单词，字母小写。

2 不要添加参数集合

如果没有添加参数集合，可以获取所有的数据。

如果添加了参数集合，默认无法获取通信数据，即使传递$event也只能获取第一个参数数据，其它的数据丢失了。

第二步 在父组件中，定义事件回调函数。

第三步 在子组件中，通过$emit方法发布消息，并且传递子组件中的数据。

注意：消息名称必须一致

唯一一个不需要驼峰式命名转换的。

第四步 在父组件中，接收数据，并存储数据。

|  |
| --- |
| 1. <div id="app"> 2. <h1>parent: {{msg}}</h1> 3. <!-- 子组件 --> 4. <!-- 1 模拟DOM事件绑定 --> 5. <home @ickt-demo="fn"></home> 6. <!-- 不要添加参数集合 --> 7. <!-- <home @ickt-demo="fn($event)"></home> --> 8. </div> 9. let Home = Vue.extend({ 10. // 数据 11. data() { 12. return { 13. msg: 'hello' 14. } 15. }, 16. // 模板 17. template: ` 18. <div> 19. <input type="text" v-model="msg" /> 20. <h1>{{msg}}</h1> 21. </div> 22. `, 23. // 监听数据变化 24. watch: { 25. // 监听msg 26. msg(value, oldValue) { 27. console.log(value, this.msg) 28. // 监听数据变化，并向父组件发布消息 29. // this.$emit('ickt-demo', this.msg); 30. this.$emit('ickt-demo', value) 31. } 32. }, 33. // 组件创建完成 34. created() { 35. // 3 子组件中发布消息 36. // 注意：不要驼峰式命名转化 37. this.$emit('ickt-demo', this.msg, 200, true); 38. } 39. }) 40. // vue实例化对象 41. new Vue({ 42. // 注册 43. components: { Home }, 44. // 容器 45. el: '#app', 46. // 数据 47. data: { 48. msg: '' 49. }, 50. // 方法 51. methods: { 52. // 2 定义方法 53. fn(msg) { 54. // 4 接收数据，并存储数据 55. console.log(arguments) 56. // 存储数据 57. this.msg = msg; 58. } 59. }, 60. created() { 61. // 父组件发布消息无意义。 62. this.$emit('ickt-demo', 100, 200, true) 63. console.log(1111) 64. } 65. }) |

#### 1.4.4.2传递方法

子组件向父组件通信，也可以通过传递属性方法的形式来实现。

共分四步

第一步 为子组件元素传递父组件方法

第二步 子组件通过props属性接收方法

第三步 子组件执行方法，并传递数据

第四步 父组件接收数据，并存储数据。

|  |
| --- |
| 1. <div id="app"> 2. <h1>parent: {{msg}}</h1> 3. <!-- 子组件 --> 4. <!-- 1 向子组件传递方法 --> 5. <home :send-message="receiveMessage"></home> 6. </div> 7. let Home = Vue.extend({ 8. // 2 接收属性 9. props: ['sendMessage'], 10. // 数据 11. data() { 12. return { 13. msg: 'hello' 14. } 15. }, 16. // 模板 17. template: ` 18. <div> 19. <input type="text" v-model="msg" /> 20. <h1>{{msg}}</h1> 21. </div> 22. `, 23. // 监听数据的变化 24. watch: { 25. // msg的变化 26. msg(value) { 27. // 执行方法 28. this.sendMessage(value) 29. } 30. }, 31. // 创建完成 32. created() { 33. // console.log(this) 34. // 3 执行方法，传递数据 35. this.sendMessage(this.msg, 100, true) 36. } 37. }) 38. // vue实例化对象 39. new Vue({ 40. // 注册 41. components: { Home }, 42. // 容器 43. el: '#app', 44. // 数据 45. data: { 46. msg: '' 47. }, 48. // 方法 49. methods: { 50. // 父组件定义的方法，this指向父组件。 51. receiveMessage(msg) { 52. // 4 接收数据，存储数据 53. console.log(arguments) 54. this.msg = msg; 55. } 56. } 57. }) |

### 1.4.5兄弟组件间通信

在同一父组件中的两个子组件，这两个子组件互为兄弟组件。

想实现兄弟组件间的通信，只需要两步

第一步 在一个子组件中，向父组件通信

第二步 在父组件中，向另一个子组件通信。

两个兄弟组件没有直接的关系，但是都与父组件有关系。

|  |
| --- |
| 1. <div id="app"> 2. <h1>parent: {{msg}}</h1> 3. <!-- 第一个子组件 --> 4. <!-- 传递方法 --> 5. <!-- <child :send-msg="receiveMsg"></child> --> 6. <!-- 自定义事件 --> 7. <child @ickt="receiveMsg"></child> 8. <!-- 第二个子组件 --> 9. <other :parent-msg="msg"></other> 10. </div> |

## 1.6插槽

允许我们在组件模板中，访问组件元素内部的其它元素。

### 1.6.1使用插槽

使用插槽分成两步。

第一步 在组件元素内。为其它元素设置插槽名称。

通过slot属性设置。

第二步 在组件模板中，通过slot组件，使用这些元素。

通过name属性指令插槽名称

如果没有设置name属性，默认会使用剩余的元素。

使用插槽的时候，默认会引入slot属性所在的元素，

不想引入该元素，我们可以使用template模板元素。

使用template模板元素的时候，在新版本中，建议用v-slot指令代替slot属性

此时为插槽定义名称的语法是冒号语法。

v-slot:名称

注意：该指令不能给普通元素使用，如div，h1，span等。

只能给组件和template元素使用。

|  |
| --- |
| 1. <div id="app"> 2. <h1>parent</h1> 3. <!-- 使用子组件 --> 4. <home> 5. <!-- 1 设置插槽名称 --> 6. <template slot="header"> 7. <h1>header</h1> 8. </template> 9. <!-- 普通元素不能使用 --> 10. <!-- <div v-slot:demo></div> --> 11. <div> 12. <h1>body</h1> 13. </div> 14. <!-- 新版本中，建议使用v-slot指令 --> 15. <template v-slot:footer> 16. <h1>footer</h1> 17. </template> 18. </home> 19. </div> 20. <!-- 模板 --> 21. <template id="tpl"> 22. <div> 23. <!-- 2 使用插槽 --> 24. <slot name="header"></slot> 25. <!-- 没有指定名称 --> 26. <slot></slot> 27. <h1>home component</h1> 28. <slot name="footer"></slot> 29. </div> 30. </template> |

### 1.6.2插槽作用域

作用域是指：在插槽中使用的数据存储在哪里。

默认在插槽中，使用的数据是父组件中的数据（插槽使用的元素，是在父组件模板中定义的）。

在插槽元素中，使用子组件中的数据分成两步。

第一步 为slot组件，传递子组件的数据

动态传递数据，使用v-bind指令

第二步 为v-slot指令赋值，定义接收数据的命名空间（作用域名称）

接收的数据，驼峰式命名

此时在元素中，通过该命名空间，可以使用传递的数据，

v-slot指令语法糖是#。

已经学过的语法糖

: => v-bind:

@ => v-on:

# => v-slot:

|  |
| --- |
| 1. <div id="app"> 2. <h1>parent</h1> 3. <!-- 使用子组件 --> 4. <home> 5. <!-- 2 声明作用域命名空间 --> 6. <!-- <template v-slot:header="childScope"> --> 7. <!-- 语法糖 --> 8. <template #header="childScope"> 9. <h1>header -- {{msg}}</h1> 10. <h1>{{childScope}}</h1> 11. <h1>{{childScope.color}}--{{childScope.childMsg}}</h1> 12. </template> 13. </home> 14. </div> 15. <!-- 1 传递属性 --> 16. <slot name="header" color="red" :child-msg="msg"></slot> |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |