■Vue基础

见文件vueBase1.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

<style type="text/css">

.cla1{

width: 100px;

height: 50px;

background-color: lawngreen;

}

.cla2{

border: solid 2px blue;

}

[v-cloak]{

display: none;

}

</style>

</head>

<body>

<!--总结开始

利用new Vue来创建Vue实例

实例中的选项:

el 选择器用来指定该实例需要挂载的元素

data 实例中定义的名值对数据都保存在这里

methods 实例中定义的方法都保存在这里

生命周期钩子，用来在特定时机触发指令如created, mounted, beforeDestroy

filters 设置对插值的过滤方法然后使用{{变量 | 过滤器名}}来实现过滤插值

computed 计算属性

指令：

v-bind的用法，用来动态更新HTML元素上的属性

v-bind绑定属性

v-bind绑定class

v-bind绑定内联样式

v-on的用法，给元素绑定事件

修饰符，这里略，详细见书p46

v-model的用法，用于双向数据绑定

v-pre的用法

v-html的用法

v-cloak的用法

v-if的用法

v-show的用法

v-if和v-show的区别

{{}}插值方法，里面的内容可以使用表达式

$el的用法

总结结束-->

<div id="app">

<input type="text" v-model="name"/> <!--v-model用来实现双向数据绑定-->

<p v-once>{{shuoming}}</p> <!--v-once表示只渲染一次，所以之后修改shuoming的值不会影响这里-->

<p v-cloak>{{shuoming}}</p> <!--v-cloak使系统加载时，不会出现未编译好的带大括号的内容。-->

<h2>你好，{{name}}</h2>

<p v-pre>这里的内容不会被编译：{{name}}</p> <!--v-pre属性表示{{}}内的内容不会被编译-->

<p>{{link}}</p>

<p v-html="link"></p> <!--v-html属性可使编译后的内容按照HTML规则输出-->

<h1>插值表达式计算后的日期：{{date.toLocaleString().slice(0,10)}}</h1> <!--使用{{}}插值，里面的内容可以使用表达式-->

<h1>过滤后的日期：{{date | jiequ}}</h1> <!--使用‘|’加过滤器名来实现过滤-->

<p>计算属性的结果：{{reversedText}}</p>

</div>

<!--使用v-bind来绑定属性及简写形式-->

<div id="app2">

<a v-bind:href="url">链接</a>

<a :href="url">简写形式</a>

<br/>

<img v-bind:src="imgUrl"/>

<img :src="imgUrl"/>

</div>

<!--使用v-bind指令绑定class-->

<div id="app3">

<!--对象语法，cla1为class名，c1为data里的变量，true是绑，false是不绑，便于之后操作-->

<div :class="{'cla1':c1, 'cla2':c2}">对象语法</div>

<!--数组语法，shu1为data里的变量，对应的值是class名-->

<div :class="[shu1, shu2]">数组语法</div>

<!--内联样式，chicun为data里的变量-->

<div :style="{'width':chicun +'px', 'height':chicun + 'px', 'background-color':colorr}">内联样式</div>

</div>

<!--v-if和v-show-->

<div id="app5">

<p v-if="status===1">使用v-if的例子1</p> <!--判断data里的变量status如果等于1时显示-->

<p v-if="panduanshow">使用v-if的例子2</p> <!--panduanshow是data里的变量，值是true时则显示，false时不显示-->

<p v-show="status===1">使用v-show的例子1</p>

<!--v-if是真正渲染，不显示就是不渲染出HTML元素。而v-show是给已有的元素加上display:none,下面的例子看控制台打印结果-->

<p id="p1" v-if="status===2">v-if和v-show的区别</p>

<p id="p2" v-show="status===2">v-if和v-show的区别</p>

</div>

<!--使用v-for遍历数组和对象-->

<div id="app6">

<ul>

<!--遍历数组-->

<li v-for="book in books">{{book.name}}</li>

<li v-for="(book, index) in books">{{index}} \* {{book.name}}</li>

<!--遍历对象-->

<li v-for="value in user">{{value}}</li>

<li v-for="(value, key, index) in user">{{index}} \* {{key}} : {{value}}</li>

</ul>

</div>

<!--使用v-on来绑定事件-->

<div id="app7">

<h1>点击次数：{{counter}}</h1>

<button v-on:click="handleAdd()">+1</button> <!--v-on简写形式是@，如下面的例子-->

<button @click="handleAdd(10)">+10</button>

<button @click="counter++">直接在@click里使用表达式的方法来实现+1</button>

<a href="http://www.apple.com" @click="handleClick('禁止打开', $event)">阻止默认行为的例子</a>

</div>

<!--表单与v-model-->

<div id="app8">

<input type="text" v-model="message" placeholder="请输入..."/>

<p>输入的内容是：{{message}}</p>

<!--单选按钮-->

<input type="radio" :checked="picked1"/>

<label>单独的单选按钮</label>

<br/>

<input type="radio" v-model="picked2" value="html" id="html"/>

<label for="html">组合的单选按钮：HTML</label>

<input type="radio" v-model="picked2" value="js" id="js"/>

<label for="js">组合的单选按钮：js</label>

<input type="radio" v-model="picked2" value="css" id="css"/>

<label for="css">组合的单选按钮：css</label>

<p>您的选择是：{{picked2}}</p>

<!--复选框-->

<input type="checkbox" v-model="checked1" id="checked1"/>

<label for="checked1">单独的复选框{{checked1}}</label>

<br />

<input type="checkbox" v-model="checked2" id="html" value="html"/>

<label for="html">组合的复选框HTML</label>

<input type="checkbox" v-model="checked2" id="js" value="js"/>

<label for="js">组合的复选框HTML</label>

<input type="checkbox" v-model="checked2" id="css" value="css"/>

<label for="css">组合的复选框HTML</label>

<p>您的选择是：{{checked2}}</p>

<!--选择列表-->

<select v-model="selected1">

<option>html</option>

<option value="js">JavaScript</option>

<option>css</option>

</select>

<p>您的选择是：{{selected1}}</p> <!--如果选中的option有value属性，则这里解析的是value的值，如果没有则解析标签内的text-->

<br />

<select v-model="selected2" multiple> <!--multiple属性可使选项变为多选-->

<option>html</option>

<option value="js">JavaScript</option>

<option>css</option>

</select>

<p>您的选择是：{{selected2}}</p> <!--如果选中的option有value属性，则这里解析的是value的值，如果没有则解析标签内的text-->

<br />

<!--动态输出option-->

<select v-model="selected3">

<option v-for="option in options" :value="option.value">{{option.text}}</option>

</select>

<p>您的选择是：{{selected3}}</p>

</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

var ap1 = new Vue({

// el:'#app',

el: document.getElementById('app'), //el选项可以使用CSS选择器，也可以直接写el:'#app'

data:{

name:'',

shuoming:'提示内容',

link:'<a href="#">这是一个链接</a>',

date:new Date(),

text:'123, 456, 789'

},

mounted:function(){ //生命周期钩子，如created, mounted, beforeDestroy

console.log(this.shuoming); //this指向调用这个钩子的Vue实例，这里就是对象app1

console.log(this.$el.innerText); //app1.$el可以访问Vue实例挂载的元素

},

filters:{

jiequ:function(value){ //value指的是要被过滤的数据，过滤结果要用return返回出去

return value.toLocaleString().slice(0,10);

}

},

computed:{ //计算属性，结果要用return返回出去

reversedText:function(){

return this.text.split(',').reverse().join('\*');

}

}

});

ap1.shuoming="你输入的名字会同时出现在下方"; //可以修改vue实例对象中的值

var ap2 = new Vue({

el:'#app2',

data:{

url:'https://github.com',

imgUrl:'https://timgsa.baidu.com/timg?image&quality=80&size=b9999\_10000&sec=1515653478116&di=86acdac224ae6c9eda78eba853a6b422&imgtype=0&src=http%3A%2F%2Fcdn.qiaobutang.com%2Fuploads%2Fbbs\_images%2F2016%2F9%2F20%2F10%2F57e0a3120cf2da59292896b2%2Foriginal.jpg'

}

});

var ap3 = new Vue({

el:'#app3',

data:{

c1:true, //用来控制对象语法设置的class

c2:true,

shu1:'cla1', //使用数组语法设置class

shu2:'cla2',

colorr:'blue',

chicun:100

}

});

var ap5 = new Vue({

el:'#app5',

data:{

status:1,

panduanshow:true

}

});

// 以下将输出"p1元素是不存在的",因为用v-if来禁止元素显示是不渲染出元素的

if (document.getElementById("p1")) {

console.log("p1元素是存在的");

}else{

console.log("p1元素是不存在的");

};

// 以下将输出"p2元素是存在的",因为用v-show来禁止元素显示是渲染出元素然后给元素加上css display:none

if (document.getElementById("p2")) {

console.log("p2元素是存在的");

}else{

console.log("p2元素是不存在的");

};

// 另外补充,v-show不能在<template>上使用

var ap6 = new Vue({

el:'#app6',

data:{

books:[

{name:'《Vue.js实战》'},

{name:'《JavaScript语言精粹》'},

{name:'《JavaScript高级程序设计》'}

],

user:{

name:'Alice',

gender:'女',

age:26

}

}

});

ap6.books.push({name:'《新加的书》'}); //数组动态更新后，视图也会随着更新

var ap7 = new Vue({

el:'#app7',

data:{

counter:0

},

methods:{

handleAdd:function(count){

var jishu=count || 1; //如没输入参数则为1

this.counter += jishu; //this指向当前Vue实例

},

handleClick:function(message, event){

event.preventDefault();

window.alert(message);

}

}

});

var ap8 = new Vue({

el:'#app8',

data:{

message:'',

picked1:true, //单独的单选按钮，通过true和false来控制

picked2:'js',

checked1:true, //单独的复选框，通过true和false来控制

checked2:['html', 'css'],

selected1:'html', //单选的选择列表

selected2:['html', 'css'],

selected3:'html',

options:[

{

text:'HTML',

value:'html'

},

{

text:'JavaScript',

value:'js'

},

{

text:'css',

value:'css'

}

]

}

});

</script>

</body>

</html>

■组件

组件的基本用法:

1. component注册全局组件
2. 在某个Vue实例中使用components选项注册局部组件
3. template里的DOM结构必须被一个元素标签包裹,否则无法渲染
4. 可以使用is属性来挂载组件
5. 组件里的data选项必须是函数，然后将数据return出去(return的是一个对象，类似Vue实例中的data对象)
6. 使用props使父组件向子组件传递数据

数据验证p74

●组件基础及props

见文件vueBase2Component1.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--总结开始

组件的基本用法

component注册全局组件

在某个Vue实例中使用components选项注册局部组件

template里的DOM结构必须被一个元素标签包裹,否则无法渲染

可以使用is属性来挂载组件

组件里的data选项必须是函数，然后将数据return出去(return的是一个对象，类似Vue实例中的data对象)

使用props使父组件向子组件传递数据

数据验证p74

总结结束-->

<!--组件基本用法-->

<div id="app1">

<my-component1></my-component1>

<my-component2></my-component2>

<table>

<tbody is="my-component2"></tbody> <!--可以使用is属性来挂载组件-->

</table>

<my-component3></my-component3>

</div>

<!--使用props传递数据-->

<div id="app2">

<my-component5 message="来自父组件的数据1"></my-component5>

<my-component6 mess-age="来自父组件的数据2"></my-component6> <!--HTML标签中不区分大小写，驼峰要写成-的形式，例如mess-age-->

<!--如果传递的是变量或表达式，则需要用v-bind的方法，如下例-->

<input type="text" v-model="parentMessage"/> <!--parentMessage是app2实例里的data，这里使用V-model进行双向绑定-->

<my-component7 :bianliang="parentMessage"></my-component7> <!--使用v-bind把变量parentMessage传递给my-component7的子元素-->

</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

// 最基本的组件如下,注意：template里的DOM结构必须被一个元素标签包裹,否则无法渲染

Vue.component('my-component1',{

template:'<h1>这里是组件的内容</h1>'

});

Vue.component('my-component3',{

template:'<button @click="counter++">{{counter}}</button>',

// 组件里的data选项必须是函数，然后将数据return出去(return的是一个对象，类似Vue实例中的data对象)

data:function(){

return{

counter:0

};

}

});

var ap1=new Vue({

el:'#app1',

// 在Vue实例内使用components选项可以创建局部组件,该组件只有在该实例作用域内才能有效使用

components:{

'my-component2':{

template:'<h2>这里是局部组件的内容</h2>'

}

}

});

// 以下是app2,使用props的例子

Vue.component('my-component5',{

props:['message'],

template:'<p>{{message}}</p>'

});

Vue.component('my-component6',{

props:['messAge'], //HTML标签中不区分大小写，驼峰要写成-的形式，例如mess-age

template:'<p>{{messAge}}</p>'

});

Vue.component('my-component7',{

props:['bianliang'],

template:'<div><p>从父亲那里接收到的数据：{{bianliang}}</p><p>{{counter2}}</p></div>',

data:function(){

return{

counter2:this.bianliang + "计算后的数据变成组件内部自己的数据，不会随输入内容而改变了"

};

}

});

var ap2 = new Vue({

el:'#app2',

data:{

parentMessage:''

}

});

</script>

</body>

</html>

●组件通信 $emit自定义事件

见文件vueBase3Component2.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--总结开始

组件通信

自定义事件（子组件用$emit()来自定义事件，父组件用$on()来监听子组件的事件）

app1例子梳理：

首先子组件使用$emit()来自定义事件,第一个参数是自定义事件的名称,第二个参数是要传递的值

父组件用 v-on:自定义事件名 ，来监听自定义事件

总结结束-->

<!--自定义组件的基本用法-->

<div id="app1">

<p>总数：{{total}}</p>

<my-component1 @increase="handleGetTotal" @reduce="handleGetTotal"></my-component1> <!--父组件用v-on:自定义事件名，来监听自定义事件-->

</div>

<!--自定义组件结合使用v-model-->

<div id="app2">

<p>总数：{{total}}</p>

<my-component2 v-model="total"></my-component2>

<p>总数：{{total2}}</p>

<my-component3 @input="handleGetTotal"></my-component3>

</div>

<!--使用自定义组件结合v-model来实现表单输入数据双向绑定-->

<div id="app3">

<p>总数：{{total}}</p>

<my-component5 v-model="total"></my-component5>

<button @click="handleReduce">-1</button>

</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

// 第一个例子:自定义组件的基本用法

Vue.component('my-component1',{

// 在template里可以使用\来换行

template:'\

<div>\

<button @click="handleIncrease">+1</button>\

<button @click="handleReduce">-1</button>\

</div>',

data:function(){

return{

counter:0

}

},

methods:{

handleIncrease:function(){

this.counter++;

// 使用$emit()来自定义事件,第一个参数是自定义事件的名称,第二个参数是要传递的值

this.$emit('increase',this.counter);

},

handleReduce:function(){

this.counter--;

this.$emit('reduce',this.counter);

}

}

});

var ap1=new Vue({

el:'#app1',

data:{

total:0

},

methods:{

handleGetTotal:function(fc){ //fc就是子组件使用$emit第二个参数传递的值

this.total=fc;

}

}

});

// 以上例子中,变量total和counter是两码事,只是它们的初始值一样等于0

// 然后点击加或减的按钮后,组件会通过click事件触发handleIncrease或handleReduce方法来计算并更新counter的值

// 每一次点击按钮,组件计算后都会将更新后的counter的值通过$emit传出去

// 父组件通过v-on监听$emit,每次传来新值都会触发handleGetTotal方法,将传来的新值赋值给total

// 这样最终视图上显示的总数total会随着每次点击按钮而更新

// 第二个例子:自定义组件结合使用v-model

Vue.component('my-component2',{

template:'<button @click="handleClick">+1</button>',

data:function(){

return{

counter:0

}

},

methods:{

handleClick:function(){

this.counter++;

this.$emit('input',this.counter)

}

}

});

Vue.component('my-component3',{

template:'<button @click="handleClick">+1</button>',

data:function(){

return{

counter:0

}

},

methods:{

handleClick:function(){

this.counter++;

this.$emit('input',this.counter)

}

}

});

var ap2=new Vue({

el:'#app2',

data:{

total:0,

total2:0

},

methods:{

handleGetTotal:function(fc){

this.total2=fc;

}

}

});

// 上面例子中,组件my-component2和my-component3是完全一样的

// 父组件监听组件my-component3的$emit类似第一个例子

// 而父组件监听my-component2使用了v-model的方法来实现同样的效果

// 第三个例子:使用自定义组件结合v-model来实现表单输入数据双向绑定

Vue.component('my-component5',{

props:['value'],

template:'<input :value="value" @input="updateValue">',

methods:{

updateValue:function(event){

this.$emit('input', event.target.value);

}

}

});

var ap3=new Vue({

el:'#app3',

data:{

total:0

},

methods:{

handleReduce:function(){

this.total--;

}

}

});

// 这个例子,在第二个例子自定义组件结合v-model的基础上,接收一个value属性

// 在有新的value时触发input事件,并将新的value通过$emit传出去

</script>

</body>

</html>

●中央事件总线bus和父链和子组件索引

见文件vueBase4Component3.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--总结开始

中央事件总线bus

首先创建一个名为bus的空实例

子组件内，特点时间会通过bus把事件on-message发射出去

在实例初始化时,监听来自bus实例的事件，一旦on-message有更新，就会执行相应的函数。

父链

$parent

$children

$refs

总结结束-->

<!--bus的基本用法-->

<div id="app1">

{{message}}

<component-a></component-a>

</div>

<!--父链$parent-->

<div id="app2">

{{message}}

<component-b></component-b>

</div>

<!--$children和ref-->

<div id="app3">

<button @click="handleRef">通过ref获取子组件实例</button>

<component-c ref="comA"></component-c> <!--使用ref属性给组件起一个名字，之后在父中-->

</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

// 第一个例子:bus的基本用法

var bus=new Vue(); //创建一个名为bus的空实例

Vue.component('component-a',{

template:'<button @click="handleEvent">传递事件</button>',

methods:{

handleEvent:function(){

// 通过bus把事件on-message发射出去

bus.$emit('on-message','来自组件component-a的信息');

}

}

});

var ap1=new Vue({

el:'#app1',

data:{

message:''

},

// 在实例初始化时,监听来自bus实例的事件，一旦on-message有更新，就会执行后面的函数。msg就是bus.$emit传来的值

mounted:function(){

var \_this=this; //\_this==this==ap1

bus.$on('on-message', function(msg){

\_this.message=msg;

});

}

});

// 第二个例子:父链$parent

Vue.component('component-b',{

template:'<button @click="handleEvent">通过父链直接修改数据</button>',

methods:{

handleEvent:function(){

// 访问父链后可以做任何操作,比如修改数据

this.$parent.message='来自组件component-b的信息';

}

}

});

var ap2=new Vue({

el:'#app2',

data:{

message:''

}

});

// 第三个例子:$children和ref

Vue.component('component-c',{

template:'<div>子组件</div>',

data:function(){

return{

message:'子组件的数据'

}

}

});

var ap3=new Vue({

el:'#app3',

methods:{

handleRef:function(){

// 通过$refs来访问指定的实例

var msg=this.$refs.comA.message;

console.log(msg);

}

}

});

</script>

</body>

</html>

●slot的用法

见文件vueBase5Component4.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--总结开始

使用slot分发内容

单个slot

具名slot

作用域插槽

模板里<slot msg="来自子组件的内容"></slot> msg将数据传到了插槽

父组件使用template标签，有个scope属性可以接收传来的数据。scope="props"，props是个自定义临时变量，可以用props.msg来访问数据

通过$slots访问slot

总结结束-->

<!--slot基础-->

<div id="app1">

<!--单个slot-->

<child-component1>

<p>分发的内容</p>

</child-component1>

<!--具名slot-->

<child-component2>

<h2 slot="header1">标题</h2>

<p>正文内容</p>

<div slot="footer1">底部内容</div>

</child-component2>

</div>

<!--作用域插槽-->

<div id="app2">

<child-component3>

<!--父组件使用template标签，有个scope属性可以接收传来的数据。scope="props"，props是个自定义临时变量，可以用props.msg来访问数据-->

<template scope="props">

<p>来自父组件的内容</p>

<p>{{props.msg}}</p>

</template>

</child-component3>

</div>

<!--作用域插槽代表性用例，列表组件-->

<div id="app3">

<my-list :bookn="books">

<template slot="book" scope="props">

<li>{{props.bookName}}</li>

</template>

</my-list>

</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

// 第一个例子:slot基础

Vue.component('child-component1', {

template:'\

<div>\

<slot>\

<p>如果父组件(child-component1)没有插入内容,我将作为默认出现</p>\

</slot>\

</div>'

});

Vue.component('child-component2',{

template:'\

<div class="container">\

<div class="header">\

<slot name="header1">\

<h1>原生标题</h1>\

</slot>\

</div>\

<div class="main">\

<slot>\

<p>原生内容</p>\

</slot>\

</div>\

<div class="footer">\

<slot name="footer1">\

<p>原生底部</p>\

</slot>\

</div>\

</div>',

mounted:function(){

var aheader=this.$slots.header1; //可以通过$slot来访问

var amain=this.$slots.default; //$slots.default表示访问非具名节点

alert(aheader);

alert(amain);

}

});

var ap1=new Vue({

el:'#app1'

});

// 第二个例子:作用域插槽

Vue.component('child-component3',{

// 模板里<slot msg="来自子组件的内容"></slot> msg将数据传到了插槽

template:'\

<div class="container">\

<slot msg="来自子组件的内容"></slot>\

</div>'

});

var ap2=new Vue({

el:'#app2'

});

// 第三个例子:作用域插槽代表性用例，列表组件

Vue.component('my-list', {

props:{

bookn:{

type:Array,

default:function(){

return [];

}

}

},

template:'\

<ul>\

<slot name="book"\

v-for="book in bookn"\

:book-name="book.name">\

<p>{{book.name}}</p>\

</slot>\

</ul>'

});

var ap3=new Vue({

el:'#app3',

data:{

books:[

{name: '《Vue.js实战》'},

{name: '《JavaScript语言精粹》'},

{name: '《JavaScript高级程序设计》'}

]

}

});

</script>

</body>

</html>

●组件的高级用法

见文件vueBase6Component5.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--总结开始

组件的高级用法

递归组件

内联模板

动态组件<component>标签和is特性

异步组件p94

$nextTick的用法及异步更新队列的概念

X-Templates的用法

手动挂载Vue的实例

总结结束-->

<!--递归组件-->

<div id="app1">

<child-component1 :count="1"></child-component1>

</div>

<!--以上递归组件渲染后结果如下

<div id="app1">

<div class="child">

<div class="child">

<div class="child">

</div>

</div>

</div>

</div>-->

<!--内联模板-->

<div id="app2">

<child-component2 inline-template>

<div>

<h2>在父组件定义子组件的模板</h2>

<p>{{message}}</p>

<p>{{msg}}</p>

</div>

</child-component2>

</div>

<!--动态模板-->

<div id="app3">

<component :is="currentView"></component>

<button @click="handleChangeView('A')">切换到A</button>

<button @click="handleChangeView('B')">切换到B</button>

<button @click="handleChangeView('C')">切换到C</button>

</div>

<!--$nextTick的用法及异步更新队列的概念-->

<div id="app5">

<div id="div" v-if="showDiv">这是一段文本</div>

<button @click="getText">获取div内容</button>

</div>

<!--X-Templates的用法-->

<div id="app6">

<my-component1></my-component1>

<!--以下script标签里的内容将被认为是my-component1组件的模板

script标签里的type属性值必须是"text/x-template"

给script标签设定一个id，以便在my-component1的template选项中指定-->

<script type="text/x-template" id="my-component11">

<div>这里是组件的内容</div>

</script>

</div>

<!--手动挂载Vue的实例-->

<div id="mount-div"></div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

// 第一个例子:递归组件

// 1)组件需要加一个name选项

// 2)必须给一个条件来限制递归的数量

Vue.component('child-component1', {

name:'child-component1',

props:{

count:{

type:Number,

default:1

}

},

template:'\

<div class="child">\

<child-component1\

:count="count + 1"\

v-if="count < 3"></child-component1>\

</div>'

});

var ap1=new Vue({

el:'#app1'

});

// 第二个例子:内联模板

// 给组件标签使用inline-template属性,组件就会把它的内容当做模板,而不是把它当内容分发

// 注意,下面例子中,父组件和子组件中的数据(message, msg)都能访问

Vue.component('child-component2', {

data:function(){

return{

msg:'在子组件声明的数据'

}

}

});

var ap2=new Vue({

el:'#app2',

data:{

message:'在父组件声明的数据'

}

});

// 第三个例子:动态模板

// 使用<component>标签来动态挂载不同的组件

// 使用is特性来选择要挂载的组件

var ap3=new Vue({

el:'#app3',

components:{

comA:{

template:'<div>组件A</div>'

},

comB:{

template:'<div>组件B</div>'

},

comC:{

template:'<div>组件C</div>'

}

},

data:{

currentView:'comA'

},

methods:{

handleChangeView:function(component){

this.currentView='com' + component;

}

}

});

// 第四个例子:$nextTick的用法及异步更新队列的概念

var ap5=new Vue({

el:'#app5',

data:{

showDiv:false

},

methods:{

getText:function(){

this.showDiv=true;

this.$nextTick(function(){

var text=document.getElementById('div').innerHTML;

console.log(text);

});

// 如果不使用$nextTick而直接打印innerHTML会报错,因为this.showDiv=true;的同时打印,那时候还没有div元素呢

}

}

});

// 第五个例子:X-Templates的用法

Vue.component('my-component1', {

template:'#my-component11' //指定script type=text/x-template标签的id

});

var ap6=new Vue({

el:'#app6'

});

// 第六个例子:手动挂载Vue的实例

// Vue.extend是基础Vue构造器,创建一个"子类",参数是一个包含组件选项的对象

// $mount()可以手动挂载一个未挂载的实例

var MyComponent=Vue.extend({

template:'<div>Hello:{{name}}</div>',

data:function(){

return{

name:'Alice'

}

}

});

new MyComponent().$mount('#mount-div'); //用new创建一个新实例，并用$mount挂载到#mount-div中

</script>

</body>

</html>

●自定义指令基础

见文件vueBase7Directive1.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--总结开始

自定义指令基本用法

一个指令定义对象可以提供如下几个钩子函数 (均为可选)：

bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

inserted：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)。

update：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 (详细的钩子函数参数见下)。

componentUpdated：指令所在组件的 VNode 及其子 VNode 全部更新后调用。

unbind：只调用一次，指令与元素解绑时调用。

指令钩子函数会被传入以下参数：

el：指令所绑定的元素，可以用来直接操作 DOM 。

binding：一个对象，包含以下属性：

name：指令名，不包括 v- 前缀。

value：指令的绑定值，例如：v-my-directive="1 + 1" 中，绑定值为 2。

oldValue：指令绑定的前一个值，仅在 update 和 componentUpdated 钩子中可用。无论值是否改变都可用。

expression：字符串形式的指令表达式。例如 v-my-directive="1 + 1" 中，表达式为 "1 + 1"。

arg：传给指令的参数，可选。例如 v-my-directive:foo 中，参数为 "foo"。

modifiers：一个包含修饰符的对象。例如：v-my-directive.foo.bar 中，修饰符对象为 { foo: true, bar: true }。

vnode：Vue 编译生成的虚拟节点。移步 VNode API 来了解更多详情。

oldVnode：上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。

总结结束-->

<div id="app1">

<input type="text"/>

<input type="text" v-focus/>

<!--v-test:msg.a.b="message"详解：

msg是传给指令的参数。（arg：传给指令的参数，可选。例如 v-my-directive:foo 中，参数为 "foo"。）

.a.b是修饰符（modifiers：一个包含修饰符的对象。例如：v-my-directive.foo.bar 中，修饰符对象为 { foo: true, bar: true }。）

message是指令绑定的值（value：指令的绑定值，例如：v-my-directive="1 + 1" 中，绑定值为 2。）-->

<div v-test:msg.a.b="message"></div>

<!--如果需要绑定多个值，自定义指令可以传入一个JavaScript对象字面量-->

<div v-test1="{msg:'hello', name:'Alice'}"></div>

</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

<script type="text/javascript">

Vue.directive('focus', {

inserted:function(el){

el.focus();

}

});

Vue.directive('test',{

bind: function(el, binding, vnode){

var keys=[];

for(var i in vnode){

keys.push(i);

}

el.innerHTML=

'name:' + binding.name + '<br>' +

'value:' + binding.value + '<br>' +

'expression:' + binding.expression + '<br>' +

'argument:' + binding.arg + '<br>' +

'modifiers:' + JSON.stringify(binding.modifiers) + '<br>' +

'vnode keys:' + keys.join(',')

}

});

Vue.directive('test1', {

bind:function(el, binding, vnode){

console.log(binding.value.msg);

console.log(binding.value.name);

}

})

var ap1=new Vue({

el:'#app1',

data:{

message:'some text'

}

});

</script>

</body>

</html>