vue2.5入门



第一个vue:

<!DOCTYPE html>

<html>

<head>

<title>Vue入门</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<div id="root">{{msg}}</div>

<script type="text/javascript">

/\*创建vue实例\*/

new Vue({

el: "#root", /\*el 绑定dom\*/

data: {

msg: "Hello World"

}

})

</script>

</body>

</html>

第二个Vue：

<!DOCTYPE html>

<html>

<head>

<title>Vue入门</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<!--挂载点、模板、实例之间的关系-->

<!--挂载点内部的内容就是模板, 模板可以写到挂载点内部，也可以写到Vue实例内部-->

<div id="root">

<!-- <h1>{{msg}}</h1> -->

</div>

<script type="text/javascript">

new Vue({

el: "#root", /\*el:标签后面的就是挂载点\*/

template: '<h1 style="color:red;">{{msg}}</h1>', /\*模板，可以将挂载点内部的内容放到此处\*/

data : {

msg: "Hello World"

}

})

</script>

</body>

</html>

**【插值表达式】**：{{msg}}

**【数据占位符】**：{{data}}，对应data属性的key；

**【数据模板】**：v-text和v-html，对应data属性的key，使用方法同数据占位符，其中：v-text会直接输出html语句，v-html不会将html语句输出；

**【dom模板】**：template属性能输出dom模板；

**【绑定事件】**：v-on:click="方法名"，简写为 @click="方法名"。事件对应methods属性中的 方法名： function(){}。在事件方法中可以用实例中data：{}中的数据，this.数据名。

**【绑定dom属性】**：v-bind:property='data'，简写为 :property='data' 如 :title='data'，data对应为data属性的key；

<!DOCTYPE html>

<html>

<head>

<title>Vue入门</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<!--

v-text: 会进行转译

v-html: 不会进行转译

v-on: 用于绑定事件,可以简写为 @

-->

<div id="root">

<h1 style="color:red;">{{msg}}--{{number}}</h1>

<h1 style="color:blue;" v-text="number"></h1>

<h1 style="color:green;" v-html="number"></h1>

<div v-text="content"></div>

<br>

<div v-html="content"></div>

<div v-on:click="handleClick">{{msg}}</div>

</div>

<script type="text/javascript">

new Vue({

el: "#root",

data: { /\*数据项\*/

msg: "Hello World",

number: 123,

content: "<h1 style='color:red;'>Hello World</h1>"

},

methods: {

handleClick : function(){ //事件方法

this.content="world"

}

}

})

</script>

</body>

</html>

v-on事件绑定，可以简写成@

v-model是双向数据绑定

v-bind绑定属性，可缩写为:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue：属性绑定和双向数据绑定</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<div id="root">

<!-- <div title="This is Hello World">Hello World</div> -->

<!--

事件绑定：v-on: 可简写成@

属性绑定：v-bind: 可简写成:

-->

<div v-bind:title="title">Hello World</div>

</div>

<script type="text/javascript">

new Vue({

el : "#root",

data : {

title: "This is hello world"

}

})

</script>

</body>

</html>

数据单向绑定：数据决定页面的显示，但是页面无法决定数据里面的内容

使用v-bind进行单向数据绑定；

数据双向绑定：数据与页面的显示会同时变化，使用v-model进行双向数据绑定；

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue：属性绑定和双向数据绑定</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<div id="root">

<!-- <div title="This is Hello World">Hello World</div> -->

<!--

事件绑定：v-on: 可简写成@

属性绑定：v-bind: 可简写成:

双向数据绑定：v-model

-->

<div v-bind:title="title">Hello World</div>

<br>

<input type="text" name="" v-model="content">

<div>{{content}}</div>

</div>

<script type="text/javascript">

new Vue({

el : "#root",

data : {

title: "This is hello world",

content: "你好，Vue"

}

})

</script>

</body>

</html>

methods与computed的区别：

使用 methods 来替代 computed，效果上两个都是一样的，但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。

计算属性：

computed: {//return  data中的数据发生改变才会触发该方法--返回一个新的结果

    数据名 ： function(){

                        return ...

                    }

}

侦听器：

watch:{//data中的数据发生改变时，触发该方法

     data中的数据名 ： function(){

                        方法体

                    }

}

1. computed：可以对已有的data属性进行计算，返回一个需要的值，并且在data属性值发生变更的时候也会相应的变化
2. watch：可以对data和computed的任意属性值进行侦听，当侦听到变化的时候，可以执行自定义的任何函数

<!DOCTYPE html>

<html>

<head>

<title>Vue-计算属性和监听器</title>

<meta charset="utf-8">

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<div id="root">

姓：<input type="text" name="firstName" v-model="firstName" />

名：<input type="text" name="lastName" v-model="lastName" />

<div><h1 style="color:red;">您好！{{firstName}}{{lastName}},欢迎你来到Vue世界！</h1></div>

<div><h1 style="color:green;">您好！{{fullName}},欢迎你来到Vue世界！</h1></div>

</div>

<script type="text/javascript">

new Vue({

el: "#root", //挂载点

data: { //数据

firstName: '',

lastName: ''

},

computed: { //计算属性

fullName : function(){

return this.firstName + ' ' + this.lastName

}

}

})

</script>

</body>

</html>

计算属性和监听器：

<!DOCTYPE html>

<html>

<head>

<title>Vue-计算属性和监听器</title>

<meta charset="utf-8">

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<div id="root">

姓：<input type="text" name="firstName" v-model="firstName" />

名：<input type="text" name="lastName" v-model="lastName" />

<div><h1 style="color:red;">您好！{{firstName}}{{lastName}},欢迎你来到Vue世界！</h1></div>

<div><h1 style="color:green;">您好！{{fullName}},欢迎你来到Vue世界！</h1></div>

<div style="color: blue; font-size: 40px;">计数器：{{count}}</div>

</div>

<script type="text/javascript">

new Vue({

el: "#root", //挂载点

data: { //数据

firstName: '',

lastName: '',

count: 0

},

computed: { //计算属性

fullName : function(){

return this.firstName + ' ' + this.lastName

}

},

watch: { //监听器

firstName : function(){

this.count++

},

lastName : function(){

this.count++

}

}

})

</script>

</body>

</html>

v-if 是否存在，通过删除或加载DOM元素来实现显隐。

v-show 是否显示, 其实就是控制 display: block || none，通过设置DOM的display属性来实现显隐；

如果一个DOM需要频繁的显隐，那么使用v-show更好，这样效率高。如果是一次性的显隐，那么直接使用v-if即可；

v-if和v-show】：均用来控制dom显示/隐藏，当值为true则显示，为false则隐藏。

if 和 show的一大区别：

在于if是通过销毁dom的方式实现隐藏，而show是通过隐藏dom的方式实现的，具体需要灵活使用。        若只需不频繁切换隐藏显示时用v-if可以，其会使元素从dom中清除，每次切换都得重新渲染，加载。      若需频繁切换时用v-show，它会自动加上display：none，并没有从dom中清除，这样就不用重新渲染，性能好。

v-for 循环, 添加 :key 提升渲染效率/性能, 要求每一项循环的key 不一样, 也可用 :key="index", 如果对列表进行频繁的变更, 比如排序时, 使用 index 就不适合了

<li v-for="(item index) of list" :key="item">{{item}}</li>

​

v-for=“item of list”，v-for表示循环list，依次输出item ，：key可以提升遍历效率，但key 值不能相同；key值相同时，用index做区别

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue指令—— v-if,v-show,v-for</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<div id="root">

<div style="color: red;font-size: 30px;" v-if="show">hello world</div>

<br>

<div style="color: green;font-size: 30px;" v-if="show">hello world</div>

<button style="width: 80px; height: 30px;" v-on:click="handleClick">toggle</button>

<br>

<ul>

<li v-for="item of list">{{item}}</li>

</ul>

<ul>

<!-- v-for=“item of list”，v-for表示循环list，依次输出item ，：key可以提升遍历效率，但key 值不能相同；key值相同时，用index做区别-->

<li v-for="item of list" :key="item">#{{item}}</li>

</ul>

<ul>

<li v-for="(item, index) of list" :key="index">&{{item}}</li>

</ul>

</div>

<script type="text/javascript">

new Vue({

el: "#root",

data: {

show: true,

list: [1, 2, 3, 4]

},

methods: {

handleClick: function(){

this.show= !this.show;

}

}

})

</script>

</body>

</html>

实例代码：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue-TodoList</title>

<script type="text/javascript" src="./vue.js"></script>

</head>

<body>

<!--

v-model: 实现双向数据绑定

v-for: 实现循环列表

v-on: 实现事件绑定

this.list.push(): 向列表中添加内容数据

-->

<div id="root">

<div>

<input type="text" name="name" v-model="inputValue" >

<button v-on:click="handleSubmit">提交</button>

<ul>

<li v-for="(item, index) of list" :key="index">{{item}}</li>

</ul>

</div>

</div>

<script type="text/javascript">

new Vue({

el: "#root", //挂载点

data: { //数据

inputValue: '',

list: []

},

methods: { //事件

handleSubmit: function(){

this.list.push(this.inputValue) //添加内容

this.inputValue = '' //内容提交之后，恢复为空

}

}

})

</script>

</body>

</html>

|  |
| --- |
| <ul>      <todo-item v-for="(item, index) of list" :key="index" :content="item"></todo-item>  </ul> |

***全局组件***

通过Vue.component定义的组件是全局组件, 可以在任何地方使用

|  |  |
| --- | --- |
|  | Vue.component('todo-item', {      props: ['content'], //这个组件接收从外部传进来的叫 content的属性       template: '<li>{{content}}</li>'   }); |

***局部组件:***必须在new Vue()内注册声明

|  |  |
| --- | --- |
|  | var TodoItem = { template: '<li>item</li>' } |

|  |  |
| --- | --- |
|  | new Vue({      el: '#root',      data: {          list: []      }     // components: { 'todo-item': TodoItem } //注册局部组件  }); |

1. 通过Vue.component可以创建全局组件，在页面的任何地方都可以使用全局组件
2. 通过new Vue({components:{"componentName":"component"}})可以声明局部组件，其中局部组件的创建格式为var componentName = {template:""}
3. 通过props来接收父组件传递的参数

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue-TodoList</title>

<script type="text/javascript" src="./vue.js"></script>

<style type="text/css">

</style>

</head>

<body>

<!--

v-model: 实现双向数据绑定

v-for: 实现循环列表

v-on: 实现事件绑定

this.list.push(): 向列表中添加内容数据

-->

<div id="root">

<div>

<input type="text" name="name" v-model="inputValue" style="width: 250px;height: 30px;border: solid;">

<button v-on:click="handleSubmit">提交</button>

<ul>

<!-- <li v-for="(item, index) of list" :key="index" style="color: blue;font-size: 30px;">{{item}}</li> -->

<!-- 使用组件 -->

<todo-item v-for="(item, index) of list" :key="index" :content="item" style="color: blue;font-size: 30px;">

</todo-item>

</ul>

</div>

</div>

<script type="text/javascript">

//定义全局组件

Vue.component('todo-item', {

props: ['content'],

template: '<li>{{content}}</li>'

})

new Vue({

el: "#root", //挂载点

data: { //数据

inputValue: '',

list: []

},

methods: { //事件

handleSubmit: function(){

this.list.push(this.inputValue) //添加内容

this.inputValue = '' //内容提交之后，恢复为空

}

}

})

</script>

</body>

</html>

​在Vue里 ，每一个Vue组件，都是一个Vue实例  【实例=组件】；

Vue实例有的功能，Vue组件都拥有。

每个组件都是vue的实例；

每个实例的组成部分：

props、template、data、methods...

每一个组件都有一个template模板，如果没有写template，默认为挂载点下面的所有DOM标签作为模板。

【父组件与子组件】

父组件： new Vue({})

子组件： 'todo-item'

\*   父组件-->子组件通信：属性形式

\*   父组件中通过属性向子组件传递值，子组件接受属性的内容并再子组件上进行内容显示

\*   子组件可以绑定事件，实现子组件的删除必须删除父组件的相关数据，

\*

\*   子组件和-->父组件通信：发布订阅模式

\*   子组件被点击需要通知父组件把数据删除，

\*   传递一个$emit，触发一个自定义事件，传递一个值，

\*   父组件在模板里可以监听子组件向外传递的出发的delete事件，

\*   如果父组件监听到子组件的事件，执行delete事件

\*

\* 总结一下大概的逻辑过程：

\* （1）：父组件通过props属性向子组件传值，子组件获得父组件传来的内容和索引。

\* （2）：子组件通过$emit向父组件抛出触发事件名称（delete）和触发事件的list索引值。

\* （3）：父组件通过监听对应事件名称（@delete）触发函数handleDelete。函数通过子组件抛出的索引值对应删除list

父组件通过属性的形式向子组件传递数据；

子组件通过发布订阅模式向父组件传递数据；子组件发布一个事件，父组件通过监听（订阅）这个事件，就可以实现子组件向父组件传递数据。

​ <!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue-TodoList</title>

<script type="text/javascript" src="./vue.js"></script>

<style type="text/css">

</style>

</head>

<body>

<!--

父组件通过属性的形式向子组件传递数据；

子组件通过发布订阅模式向父组件传递数据；子组件发布一个事件，父组件通过监听（订阅）这个事件，就可以实现子组件向父组件传递数据。

-->

<div id="root">

<div>

<input type="text" name="name" v-model="inputValue" style="width: 250px;height: 30px;border: solid;">

<button v-on:click="handleSubmit">提交</button>

<ul>

<!-- <li v-for="(item, index) of list" :key="index" style="color: blue;font-size: 30px;">{{item}}</li> -->

<!-- 使用组件 -->

<todo-item v-for="(item, index) of list"

:key="index"

:content="item"

:index="index"

v-on:delete="handleDelete"

style="color: blue;font-size: 30px;">

</todo-item>

</ul>

</div>

</div>

<script type="text/javascript">

//定义全局组件

Vue.component('todo-item', {

props: ['content','index'],

template: '<li @click="handleClick">{{content}}{{index}}</li>',

methods: {

handleClick: function(){

//删除内容

this.$emit('delete', this.index)

}

}

})

new Vue({

el: "#root", //挂载点

data: { //数据

inputValue: '',

list: []

},

methods: { //事件

handleSubmit: function(){

this.list.push(this.inputValue) //添加内容

this.inputValue = '' //内容提交之后，恢复为空

},

handleDelete: function(index){

console.log(index)

this.list.splice(index, 1)

}

}

})

</script>

</body>

</html>

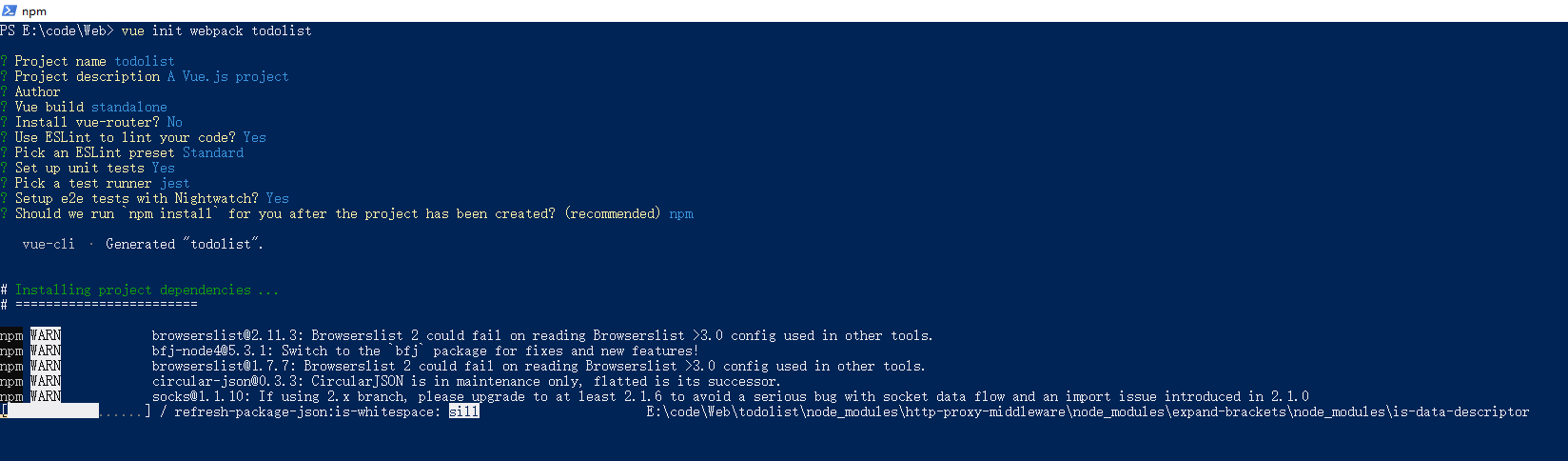
用脚手架工具创建项目

1、npm install --global vue-cli          全局安装vue脚手架工具  vue-cli

2、vue init  webpack   my-project   创建一个基于webpack模版的新项目

3、cd my-project   进入到项目目录

4、npm run dev   运行项目

​ 

build: webpack配置文件

config: 项目开发环境/线上环境的配置文件

node\_modules: 项目的依赖环境

src: 项目源代码目录

static:项目的静态资源

index.html: 项目的根实例

src目录下的文件说明：

main.js文件是项目的入口文件

App.vue是一个组件

components: 组件库

<style scoped></style>

scoped 定义了作用域, 仅对当前局部组件有效