vue知识

(http://www.runoob.com/vue2/vue-events.html)

vue安装

> npm init -y 初始化
> npm info vue 查看版本信息
> npm install vue –save
> npm install bootstrap –save
> 安装初始化package.json主要用来描述自己的项目，记录安装过的文件有哪些，在当前文件下生成json
> npm install vue –save (–save或者-S代表项目依赖，–save-dev开发依赖)
> 安装后默认生成node_modules文件夹，但是上传到git上，node_modules是忽略掉的，拉下代码后，需要重新安装,执行npm-install 安装依赖

vue:专注前端的框架

- MVC:model view control 数据视图为单向数据绑定（M处理数据库专门写处理数据的JS，V专门DOM操作，C专写交互）
- MVC思想实现前端主流框架，是我们是他是一个不完整的MVC框架，他主要做的是前端view层优化，通过处理虚拟DOM，当真正处理好dom业务时，需要通过render再真正对dom做出修改。
- MVVM：model view viewmodel(双向绑定)，只有表单元素可以双向绑定，用于input textarea
- vue响应式思想：就是一个变另一个也跟着变

目录解析

目录/文件	说明
build	最终发布的代码存放位置。
config	配置目录，包括端口号等。我们初学可以使用默认的。
node_modules	npm 加载的项目依赖模块
src	这里是我们要开发的目录，基本上要做的事情都在这个目录里。里面包含了几个目录及文件： • assets: 放置一些图片，如logo等。 • components: 目录里面放了一个组件文件，可以不用。 • App.vue: 项目入口文件，我们也可以直接将组建写这里，而不使用 components 目录。 • main.js: 项目的核心文件。
static	静态资源目录，如图片、字体等。
test	初始测试目录，可删除
.xxxx文件	这些是一些配置文件，包括语法配置，git配置等。
index.html	首页入口文件，你可以添加一些 meta 信息或同统计代码啥的。
package.json	项目配置文件。
README.md	项目的说明文档，markdown 格式

vue模板语法

1、插值

- 文本：数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值：

```
<div id="app">
  <p>{{ message }}</p>
</div>
```

2、Html ：使用 v-html 指令用于输出 html 代码

```
<div id="app">
    <div v-html="message"></div>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    message: '<h1>菜鸟教程</h1>'
  }
})
</script>
```

3、HTML 属性中的值应使用 v-bind 指令。

```
<div id="app">
  <label for="r1">修改颜色</label><input type="checkbox" v-model="cla
ss1" id="r1">
  <br><br>
  <div v-bind:class="{'class1': class1}">
    directiva v-bind:class
  </div>
</div>

<script>
new Vue({
    el: '#app',
  data:{
      class1: false
  }
});
</script>
```

4、表达式

```
RUNOOB.COM
源代码:    点击运行

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title></title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></scrip
t>
</head>
<body>
<div id="app">
    {{5+5}}<br>
    {{ ok ? 'YES' : 'NO' }}<br>
    {{ message.split('').reverse().join('') }}
    <div v-bind:id="'list-' + id">菜鸟教程</div>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    ok: true,
    message: 'RUNOOB',
    id : 1
  }
})
</script>
</body>
</html>
```

5、指令：是带有 v- 前缀的特殊属性。

6、参数：指令后以冒号指明

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></scrip
t>
</head>
<body>
<div id="app">
    <pre><a v-bind:href="url"></a></pre>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    url: 'http://www.runoob.com'
  }
})
</script>
</body>
</html>
```

7、修饰符

Vue.js 为 v-on 提供了事件修饰符来处理 DOM 事件细节，如：event.preventDefault() 或 event.stopPropagation()。
Vue.js通过由点(.)表示的指令后缀来调用修饰符。

- @click.xxx
  .stop
  .prevent
  .capture
  .self
  .once
  <a v-on:click.stop="doThis">阻止冒泡行为
  <a v-on:click.stop.prevent="doThat"> 修饰符可以串联
  <div v-on:click.self="doThat">…只当事件在该元素本身（而不是子元素）触发时触发回调

8、按键修饰符

全部的按键别名：
.enter
.tab
.delete (捕获 “删除” 和 “退格” 键)
.esc
.space
.up
.down
.left
.right
.ctrl
.alt
.shift
.meta
click 事件至少触发一次，2.1.4版本新增 –>
<a v-on:click.once="doThis">
冒号绑定的class和原生的class不冲突，冒号的class会覆盖静态的class

<form v-on:submit.prevent="onSubmit">

```
8、input用户输入
在 input 输入框中我们可以使用 v-model 指令来实现双向数据绑定：
```

{{ message }}

new Vue({ el: '#app', data: { message: 'Runoob!' } })

9、铵钮事件：v-on监听事件，对用户的输入进行响应

```
v-on:click='fn'，函数执行时，带括号一般是传递参数，会丢失参数，不写括号默认传参为e,可以手动传递属性($event),例如fn($event)
```

{{ message }}

反转字符串

```
<script>
new Vue({
  el: '#app',
  data: {
    message: 'Runoob!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
</script>
```

v-bind 动态绑定

```
<div :class="['样式1',变量1]"></div>
 <div :class="{'样式1':true,"样式2":false}"></div>
 <div :style="{color:'red',background:'blue'}"></div>
 <div :style="[行内样式对象1,行内样式对象2]"></div>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title></title>
<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></scrip
t>
<style>
.text-danger {
    width: 100px;
    height: 100px;
    background: red;
}
.active {
    width: 100px;
    height: 100px;
    background: green;
}
</style>
</head>
<body>
<div id="app">
    <div v-bind:class="[errorClass ,isActive ? activeClass : '']">
</div>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    isActive: true,
    activeClass: 'active',
    errorClass: 'text-danger'
  }
})
</script>
</body>
</html>
```

```
<!-- 完整语法 -->
<a v-bind:href="url"></a>
<!-- 缩写 -->
<a :href="url"></a>
```

v-bind:属性=”变量” -> :属性=”变量”
class style 对象和绑定 数组绑定

v-on 缩写

```
<!-- 完整语法 -->
<a v-on:click="doSomething"></a>
<!-- 缩写 -->
<a @click="doSomething"></a>
```

条件与循环

v-if与 v-show

- v-show=”flag” 如果为false,则display：none,用于频繁切换显示与隐藏，操作的样式
- v-if=“flag”如果为false,则removechild,用于判断是否存在或不频繁切换显示与隐藏,有阻断的作用，条件不成立，直接不执行后面的代码，操作的是结构。

v-for

v-for 指令需要以 item in items 形式的特殊语法， items 是源数据数组并且 item 是数组元素迭代的别名。

```
<div id="app">
  <ol>
    <li v-for="site in sites">
      {{ site.name }}
    </li>
  </ol>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    sites: [
      { name: 'Runoob' },
      { name: 'Google' },
      { name: 'Taobao' }
    ]
  }
})
</script>
```

计算属性

computed:我们可以使用 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。
computed 属性默认只有 getter ，不过在需要时你也可以提供一个 setter ：

watch属性

如果很简单的事 可以用watch ,computed不支持异步，watch支持异步

```
<div id="app">
    <input type="text" v-model="query">
    {{result}}
</div>
<script src="node_modules/vue/dist/vue.js"></script>
<script>
    var vm = new Vue({
        el:'#app',
        data:{
            query:'',
            //result:''
        },
       computed:{
            result(){
                setTimeout( () =>{
                    return '新结果'
                },2000);
                return "搜索结果"
            }
        },
        watch:{
            /*query(){ //输入时 先等待1s 在显示结果
                setTimeout(()=>{
                    this.result="搜索结果"
                },2000);
                this.result='加载中';
            }*/
            query:{ //输入时 先等待1s 在显示结果
                handler(){
                    setTimeout(()=>{
                        this.result="搜索结果"
                    },2000);
                    this.result='加载中';
                }
            }
        }
    })
</script>
```

表单

input 、textarea、select

修饰符

- .lazy
  在默认情况下， v-model 在 input 事件中同步输入框的值与数据，但你可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步

  ```
  <input v-model.lazy="msg" >
  ```

- .number 如果想自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值），可以添加一个修饰符 number 给 v-model 来处理输入值：

  ```
  <input v-model.number="age" type="number">
  ```

- .trim 如果要自动过滤用户输入的首尾空格，可以添加 trim 修饰符到 v-model 上过滤输入：

  ```
  <input v-model.trim="msg">
  ```

路由

前端路由&&不刷新页面：通过路径的变化显示不同的内容，有以下两种方式

- hash
- 浏览器自带的history

window.history.go（-1）
window.history.back(-1)
window.history.pushState({},null,'/aaa')路径可以是真实的也可以是假的
如果想使用history.pushState是需要后台支持的（页面不存在刷新会导致404），开发时采用hash的方式，上线可以改成history

promise的三种状态

- Promise本意是承诺，在程序中的意思就是承诺我过一段时间后给你一个结果。是异步操作，异步是指可能比较长时间才有结果的，例如网络请求、本地文件读取等。
- Promise 的三种状态
  Pending Promises 对象实例创建时候的初始状态
  Fulfilled 可以理解为成功的状态
  Rejected 可以理解为失败的状态
  then 方法就是用来指定Promise对象的状态改变时确定执行的操作，resolve时执行的第一个函数（onFulfilled），reject时执行第二个函数（onRejected）
  then中可以放置一个成功的回调和一个失败的回调
  reslove 成功

reject 失败
let promise=newPromise((resolve,reject)=>{
console.log(‘hello’);
})

- Promise.all实现并行
  -接受一个数组，数组内都是Promise实例，返回一个Promise实例，这个Promise实例都处于resolve状态时，返回的Promise实例会变为resolve状态。如果参数中任意一个实例处于reject状态，返回的Promise实例变为reject状态。
- Promise.race 实现选择
- Promise.resolve
- Promise.reject

数组常用的方法

- find 查找某个元素
  返回true则将当前遍历的元素返回，找到后立即返回不继续查找。
- some 查看是否有满足条件的元素
  返回true则结果为true，找到后立即返回不继续查找。
- map 遍历每一项，用返回值替换当前项

```
var arr=[{name:'jw',age:18},{name:'xfpx',age:8}];
var result=arr.map(function(item){
item.age+=10;
return item
});
console.log(result);
```

- filter
  返回true则表示当前项删除，返回false当前项保留

```
var arr=[{name:'jw',age:18},{name:'xfpx',age:8}];
var result=arr.filter(function(item){
item.age+=10;
return item
});
console.log(result);
```

- reduce
  返回值是经过叠加处理后的操作
  函数中的第一个参数为prev，第二个参数为next，求出价格中的总和,第一次的返回值为下一次的上一次prev

```
var arr=[{price:1},{price:2},{price:3}];
var result=arr.reduce(function(prev,next){
return prev+next.price;
},0);
console.log(reslut);
```

- forEach
  遍历数组，缺点：不能return循环
- for in
  缺点：会遍历所有属性，包括公有及私有的

```
var ary=[1,2,3];
arr.name='jw';
for(var key in arr){
console.log(typeof key);
console.log(arr[key]);
}
```

- for of
  好处：可以break,不会遍历数组以外的属性
  缺点：不能遍历普通对象

```
var ary=[1,2,3];
ary.ame='js';
for(var value of ary){
if(key==2){break};
console.log(ary);
}
```