vue 第一周

数组的迭代方法

for

```
for (var i=0; i<5; i++)
  {
  x=x + "The number is " + i + "<br>";
  }
```

for in

- 语法：for（（val，key） in obj）{}
- obj表示一个对象，val则表示对象中的属性值和方法，key表示属性名和方法名。for…in…循环遍历对象内的属性和方法，

for of （数组，对象，字符串，数组对象）

- 语法：for (var value of myArray) {
  console.log(value);
  }

- 与forEach()不同的是，它可以正确响应break、continue和return语句
-

forEach

- 语法：array.forEach(function(item, index, arr))
- 参数：item 当前对象， index 当前对象索引， arr 当前对象所属的数组对象

map 映射

- map()方法返回一个新数组，数组中的元素为原始数组元素调用函数处理的后值。
- map()方法按照原始数组元素顺序依次处理元素。
- map不会对空数组进行检测
- map不会改变原始数组
- 语法：arr.map(function(currentValue，index，arr),thisValue)

- 参数说明 ：
  - currentValue 必须 当前元素值
  - index 可选 当前元素的索引值
  - arr 可选 当前元素属于的数组对象。

- 例子：

```
<p>点击按钮将数组中的每个元素乘以输入框指定的值，并返回新数组</p>
    <p>
        最小年龄:
        <input type="number" id="age" value="10" />
    </p>
    <button id="btn">点我</button>
    <p id="data">新数组</p>

    <script type="text/javascript">
        var btn = document.getElementById("btn");
        var data = document.getElementById("data");
        var age = document.getElementById("age");
        var numbers = [25,36,121,49];

        function myFunction(num,index,arr){
            console.log('arr',arr);
            console.log('index',index);
            console.log('num',num);
            return num * age.value;
        }

        btn.onclick = function(){
            data.innerHTML = numbers.map(myFunction);
        }
```

some

- 语法：arr.some((item,index)=>{
  console.log(‘q’);
  return index > 2
  });
- 返回值是个 布尔类型
  // 返回值若是 true : 所有的回调函数中，至少有一个 return 的值是 true
  // 返回值若是 false: 所有的回调函数返回值都是 false
- 迭代次数 当碰到return true时，后边的项不再迭代；

every

- 语法：arr.every((item,index)=>{});
- 返回值：只要有一个回调的返回值时false 那么结果就是false
  // 全是true时返回结果才是true；
- 迭代次数 当碰到 return false 时，迭代结束

filter 过滤

- 语法：arr.filter((item,index)=>{
  return index ！== 1；
  });
- 返回值是个新数组； 数组中的项决定于 回调函数的return的布尔值
- 若 当前迭代 让return true ;则把当前项放到新数组中；

reduce

- 语法：let res7 = arr.reduce((prev,next)=>{
  return prev + next
  })
- arr.reduce((prev,next)=>{
  console.log(prev,next);
  // 本次的prev 就是 上一次的回调的 return值
  // reduce 有两个参数 ；第一个参数是个回调函数 ；第二个参数可以不写；若写了，则是回调函数的第一个 prev 值；不写的话，回调函数的第一个 prev 是数组中的第一项
  // reduce的返回值是 最后一次回调的 return值
  return next
  })

vue全家桶 vue-router vuex vue-cli

安装包的三种方式

- 1、npm install –》简写 npm i
- 2、cnpm install –》 简写 cnpm i
  - 使用cnpm 的前提是 全局安装了 cnpm
  - 安装命令 npm install -g cnpm –registry=https://registry.npm.taobao.org
- 3、yarn add 包名
  - 使用这个而命令的前提是全局安装了 yarn
  - 安装命令 npm i -g yarn

vue 语法

```
<div id="app">
    <!-- 小胡子
        这里边可以直接写变量
        也可以写表达式
        但是不能写JS语句
     -->
    <!--name = '珠峰培训2018'  这个表达式的返回结果是 '珠峰培训2018'-->

    {{name = '珠峰培训2018'}}
    {{age && 'qqqq'}}
    <!-- 赋值表达式 是有返回值的  返回值就是 赋给变量的 那个值 -->
    <!-- 两次赋值 直接报错？？？ -->
    <h2>{{age > 3 ? name='qqq' : name='www'}}</h2>
    {{name}}
</div>
```

```
let vm = new Vue({
        el:'div',// 决定哪个元素是 vue的html 模板 ;值就是一个css 选择器; 但是只对第一个元素起作用
        data:{
            name:"珠峰培训2018"
        }
})
```

绑定 html模板的方法

- 1、el
- 2、$mount
- 3、直接写 template 属性

用法结构

```
let vm = new Vue({
        el:"#app",//需要操作的对象
        data:{//定义数据
            ary:[],
            name:""
        },
```

```
        created(){//钩子函数
            //实例被创建
            //this是指向当前这个实例
            this.getData();
        },
        methods:{//定义函数
            getData(){
//    axios请求数据              axios.get("./data.json").then((data)=>
{
                    console.log(data);
                    this.ary = data || [];
                }).catch((err)=>{
                    console.log(err);
                })
            }
        },
        filters:{
            //过滤器
            money(val){//val 就是 管道符| 前边的值
                return (val/100).toFixed(2)
            }
        }
    })
```

指令

- v-text 等同于{{}} v-text = “name”
- v-html 可以渲染字符串中的标签 v-html = “name”
- v-once 该标签只渲染一次
- v-cloak 解决小胡子显示的问题 结合css属性选择器用
- v-pre 有这个属性的元素 vue不会对元素内的内容编译
- v-model 输入框内容

改变视图内容

- 1.自定义一个无关变量，由这个变量触发视图更新
- 2.创建一个新对象，整个替换
- 3.提前写死（把需要的变量都写全）
- 4.vm.$set(targe,key,value)
- 想要触发视图更新 两个前提：1.该属性有get和set方法 2.该属性在html页面中有用到

v-for

- 是循环指定元素，属性用在哪个标签上就循环换哪个标签
- 可以循环 数组 对象 字符串 数字
- 例子：

```
<ul>
    <li v-for="(val,index) in ary">
        {{val}}{{`索引${index}`}}
    </li>
    <li v-for="(val,key,index) in obj">{{val}}{{`键${key}`}}{{`索引${index}`}}</li>
    <li v-for="val in 10">number:{{val}}</li>
</ul>
```

事件

- keyup.enter 键盘监听事件
  键盘事件中常用键：

  keydown.enter=’show()’ 回车执行

  keydown.up=’show()’ 上键执行

  keydown.down=’show()’ 下键执行

  keydown.left=’show()’ 左键执行

  keydown.right=’show()’ 右键执行
- v-on
    - click , mouseenter , mouseover , mouseleave

```
<!--<button v-on:click="fn">按钮</button>-->
<button @click="fn2">按钮1</button>
<!--不加括号默认把事件对象e传给对应的函数-->
<button @click="fn2()">按钮2</button>
<!--加括号时，括号里些什么就给函数传递什么参数，不会默认传事件对象e-->
```

```
let vm = new Vue({
    el:"#app",//优先级高于$mount
    data:{//data里的东西最终都挂在实例vm上
        name:"珠峰",
        /*fn(){//this指向window
            console.log(this);
        }*/
    },
    methods:{//this指向实例vm
        //methods中的属性名不能和data中的属性名重复
        fn2(){
            //console.log(this);
```

```
            console.log(arguments);
        }
    }
})
```

axios 请求数据

- axios 的get 和 post 怎么用？怎么传参 即可
- axios.defaults.baseURL = ” // 设置基础路径
- axios.get(url,{params:参数对象}).then((data)=>{}).catch((err)=>{console.log(err)})
- axios.post(url,参数对象).then().catch()

//请求拦截器
axios.interceptors.request.use((config)=>{
config.data.pc = true;
return config
},(err)=>{
return Promise.reject(err)
})
//响应拦截器
axios.interceptors.response.use((res)=>{
return res.data
},(err)=>{
return Promise.reject(err)
})

computed 计算属性

```
computed:{
// 计算属性
            name2(){
// 简写方式
// 只要name 不发生改变 那么 本函数就永远不会执行；
// DOM模板中使用的一直都是 name2的缓存值
// name2 完全依赖于 name;
                return this.name.split('').reverse().join('');
            },
            name3(){
//name3 是否依赖于 this.name
// 只要这个函数中用某些变量；  那么这个计算属性就依赖于这些变量
                return 123;
            },
```

```
            name4(){
                let a = this.str+'str';
                let b = this.name + 'name';
                return a+b;
            },
            name5:{
// 这个时全写的 内容 有一个get函数 和 一个set函数
// 简写只相当于有一个get函数
// 着两个函数 可以只写一个 get；但是不能只写一个set
                get(){
//get 的return值 就是 nam5的值
                    return this.name
                },
                set(val){
 // val 就是设置给 name5的值
                    console.log(val,'set')
                }
}
```

watch 侦听

```
   watch:{
// 侦听 data中的name属性
            name(cur,prev){
//cur  代表新值,prev 代表老值只有新旧两次值不一样时，才能触发该函数
                clearTimeout(this.timer);
                this.timer = setTimeout(()=>{
                    this.str = cur + '123'
                },1000);
this.str2 = cur.split('').reverse().join('');
            },
            question(cur,prev){
                clearTimeout(this.timer2);
                this.timer2 = setTimeout(()=>{
                    axios.get('https://yesno.wtf/api')
                        .then((data)=>{
                            console.log(data);
                            this.answer = data.data.answer;
                            this.pic = data.data.image;
                        }).catch((err)=>{
                        console.log(err)
                    })
                },1000);
            }
        }
```

v-if / v-show

- v-if 是用来决定 该标签是否要加载的 true 代表这个元素要加载；false 代表这个元素不加载；
- v-if v-else v-else-if 这些指令使用时 中间不能掺杂不相干的元素
- 用的时候 所在元素需要紧挨着
- v-show 是控制这个标签是否显示；控制的是CSS属性
- v-if 是控制这个标签要不要加载的;
  `<h2 v-show="isShow">v____show</h2>`

transition 过度动画

```
.fade-enter-active,.fade-leave-active{
/*整个过渡期间的类名下的样式*/}
.fade-enter{
/*动画开始第一帧时的样式*/}
.fade-enter-to{
/*动画的最后一帧时的样式；可以理解成最终的显示状态*/}
```

v-bind 用法

:class=” bg’”
:class=” bg==’bg1’ ? ‘bg2’ : ‘bg1’”
:class=”{bg1:flag, bg2:!flag}”
:class=”[bg]”
:class=”ary”

- {bg1:flag, bg2:!flag} ：就是JS的普通对象
  对象的这种写法 属性名是要添加的类名； 属性值 是布尔值，决定要不要添加这个类名
- 数组的用法 把数组中的每一项都添加给该元素的类名

:style=”{background:’#ccc’,color:col}”
:style=”obj1”
:style=”{…obj1,…obj2}”
:style=”[obj1,obj2]”

绑定style的这种形式 对象的形式
属性名 是要加的 css属性
属性值 是要加的 css属性值 ；可以是变量；但是变量要有对应的值

事件的修饰符

- stop 阻止冒泡
  - @click.stop=”fn”
- prevent 阻止默认行为
  - @click.prevent=”fn” a标签的默认行为
- self 只有点击绑定的元素才能触发
  - `<div class="center" @click.self="fn1">`
    `<div class="inner" @click="fn2">`
    `</div>`
    `</div>`
- once 绑定的元素只能点击一次
  - @click.once=”fn”
- capture @click.capture —》 addEventListener(‘click’,fn,true) 事件要在捕获阶段触发
- passive 针对onscroll事件 不加这个修饰符；它是先执行事件；再看事件中有没有组织默认行为；没有阻止才会触发默认行为。 加这个修饰符， 他就不管事件中是否有阻止默认，都会直接出发默认行为；
- v-model.number=”name” 可以改变数据格式
- v-model.trim=”name”
  - trim 去除首位空格；即使在输入框看着输入了空格；但是数据层的数据仍然是没有空格的数据

自定义指令

`<h1 v-color-red="'red'">{{name}}</h1>`
`<h1 v-color-red>{{name}}</h1>`
directives:{
// 自定义指令
colorRed(ele,obj){
ele.style.color = obj.value || ‘red’;
}
}

深度watch

v-model=”obj.name”

```
data:{
```

```
        name:"123",
        obj:{
           name:456
        }
},
watch:{
      name(cur,prev){
//cur 改变之前的值，prev 改变之后的值
      },
      obj:{
         handler(cur,prev){
// handler 单词是固定的
      },
      deep:true
}
```

生命周期

钩子函数

- beforeCreate created
- beforeMount mounted
- beforeUpdate updated
- beforeDestroy destroyed

beforecreated：el和data 并未初始化
created:完成了 data数据的初始化，el没有
beforeMount：完成了 el 和 data 初始化
mounted ：完成挂载
data里的值被修改后，将会触发update的操作。
执行了destroy操作，后续就不再受vue控制了（改值不起作用）,但之前渲染的元素在页面上还存在

```
beforeCreate(){
console.group('beforeCreate 创建前状态===============》');
},
created(){
console.group('created 创建完毕状态===============》');
},
beforeMount(){
console.group('beforeMount 挂载前状态===============》');
},
mounted(){
console.group('mounted 挂载结束状态===============》');
```

```
},
beforeUpdate() {
console.group('beforeUpdate 更新前状态===============》');
},
updated(){
//视图更新时触发
console.group('updated 更新完成状态===============》');
},
beforeDestroy() {
console.group('beforeDestroy 销毁前状态===============》');
},
destroyed(){
console.group('destroyed 销毁完成状态===============》');
}
```

钩子函数用处

- beforecreate : 举个栗子：可以在这加个loading事件
- created ：在这结束loading，还做一些初始化，实现函数自执行
- mounted： 在这发起后端请求，拿回数据，配合路由钩子做一些事情
- beforeDestory： 你确认删除XX吗？
- destoryed ：当前组件已被删除，清空相关内容
  当然，还有更多，继续探索中……

获取元素

```
<ul>
        <li v-for="i in n" ref="li">{{i}}</li>
</ul>
```

```
mounted(){
// 获取元素的操作 一般都在 mounted 函数中
// 通过ref 获取元素 ；若是写死的元素；则只能获取最下边的那个元素
// 若是通过v-for循环出来的；那么都能获取到
// DOM 的渲染是异步的
// console.log(this.$refs)
}
```

- created:在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图。
  - 在created的时候，视图中的html并没有渲染出来，所以此时如果直接去操作html的dom节点，一定找不到相关的元素

- mounted:在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作。
    - 在mounted中，由于此时html已经渲染出来了，所以可以直接操作dom节点

dom的异步渲染 $nextTick

```
<ul>
<li v-for="i in n" ref="a">{{i}}</li>
</ul>
```

```
vm.n = 5;
vm.$nextTick(()=>{
console.log(vm.$refs.a)
})
```