Vue数组处理学习：

### Vue中常用的数组方法

* [.filter()、.map()、.forEach()、.find()、.findIndex()、.some()、.every()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#filtermapforEachfindfindIndexsomeevery_2)
  + [.filter()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#filter_4)
  + [.map()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#map_20)
  + [.forEach()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#forEach_36)
  + [.find()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#find_52)
  + [.findIndex()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#findIndex_76)
  + [.some()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#some_95)
  + [.every()](https://blog.csdn.net/wang_xiao_ye/article/details/89385023#every_114)

.filter()、.map()、.forEach()、.find()、.findIndex()、.some()、.every()

本文仅为自己在项目中经常用到的一些数组方法的使用，以便随时查看。

**.filter()**

filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

是否改变原数组：否

是否对空数组进行检测：否

语法：

const arr= [32, 33, 16, 40];

const arr1 = arr.filter(item => item >= 18)

console.log(arr) // [32, 33, 16, 40]

console.log(arr1) // [32, 33, 40]

**.map()**

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后的值。

map() 方法按照原始数组元素顺序依次处理元素。

是否改变原数组：否

是否对空数组进行检测：否

语法：

const arr= [4, 9, 16, 25];

const arr1 = arr.map(item => item+2)

console.log(arr) // [4, 9, 16, 25]

console.log(arr1) // [6, 11, 18, 27]

**.forEach()**

forEach() 方法用于调用数组的每个元素，并将元素传递给回调函数。

注意: forEach() 对于空数组是不会执行回调函数的。

tips: forEach()中不支持使用break(报错)和return(不能结束循环)，有需要时可使用常规的for循环。

语法：

const arr= [4, 9, 16, 25];

const arr1 = [];

arr.forEach(item => arr1.push(item))

console.log(arr) // [4, 9, 16, 25]

console.log(arr1) // [4, 9, 16, 25]

**.find()**

find() 方法返回通过测试（函数内判断）的数组的第一个元素的值。

find() 方法为数组中的每个元素都调用一次函数执行：

当数组中的元素在测试条件时返回 true 时, find() 返回符合条件的元素，之后的值不会再调用执行函数。

如果没有符合条件的元素返回 undefined

注意: find() 对于空数组，函数是不会执行的。

注意: find() 并没有改变数组的原始值。

语法：

const arr= [4, 9, 16, 25];

const b = arr.find(item => item>10)

const c = arr.find(item => item<1)

console.log(arr) // [4, 9, 16, 25]

console.log(b) // 16

console.log(c) // undefined

**.findIndex()**

findIndex() 方法返回传入一个测试条件（函数）符合条件的数组第一个元素位置。

findIndex() 方法为数组中的每个元素都调用一次函数执行：

当数组中的元素在测试条件时返回 true 时, findIndex() 返回符合条件的元素的索引位置，之后的值不会再调用执行函数。

如果没有符合条件的元素返回 -1

注意: findIndex() 对于空数组，函数是不会执行的。

注意: findIndex() 并没有改变数组的原始值。

语法：

const arr= [4, 9, 16, 25];

const b = arr.findIndex(item => item>10)

const c = arr.findIndex(item => item<1)

console.log(arr) // [4, 9, 16, 25]

console.log(b) // 2

console.log(c) // -1

**.some()**

some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。

some() 方法会依次执行数组的每个元素：

如果有一个元素满足条件，则表达式返回true , 剩余的元素不会再执行检测。

如果没有满足条件的元素，则返回false。

注意： some() 不会对空数组进行检测。

注意： some() 不会改变原始数组。

语法：

const arr= [4, 9, 16, 25];

const b = arr.some(item => item>10)

const c = arr.some(item => item<1)

console.log(arr) // [4, 9, 16, 25]

console.log(b) // true

console.log(c) // false

.every()

every() 方法用于检测数组所有元素是否都符合指定条件（通过函数提供）。

every() 方法使用指定函数检测数组中的所有元素：

如果数组中检测到有一个元素不满足，则整个表达式返回 false ，且剩余的元素不会再进行检测。

如果所有元素都满足条件，则返回 true。

注意： every() 不会对空数组进行检测。

注意： every() 不会改变原始数组。

语法：

const arr= [4, 9, 16, 25];

const b = arr.every(item => item>10)

const c = arr.every(item => item>1)

console.log(arr) // [4, 9, 16, 25]

console.log(b) // false

console.log(c) // true

Vue filter:（过滤器） 私有和全局

概念：Vue.js 允许你自定义过滤器，**\*\*可被用作一些常见的文本格式化\*\***。过滤器可以用在两个地方：**\*\*mustache 插值和 v-bind 表达式\*\***。过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示；

1. HTML元素：  
<td>{{item.ctime | dataFormat('yyyy-mm-dd')}}</td>  
2. 私有 `filters` 定义方式：  
filters: { // 私有局部过滤器，只能在 当前 VM 对象所控制的 View 区域进行使用  
    dataFormat(input, pattern = "") { // 在参数列表中 通过 pattern="" 来指定形参默认值，防止报错  
      var dt = new Date(input);  
      // 获取年月日  
      var y = dt.getFullYear();  
      var m = (dt.getMonth() + 1).toString().padStart(2, '0');  
      var d = dt.getDate().toString().padStart(2, '0');  
      // 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日  
      // 否则，就返回  年-月-日 时：分：秒  
      if (pattern.toLowerCase() === 'yyyy-mm-dd') {  
        return `${y}-${m}-${d}`;  
      } else {  
        // 获取时分秒  
        var hh = dt.getHours().toString().padStart(2, '0');  
        var mm = dt.getMinutes().toString().padStart(2, '0');  
        var ss = dt.getSeconds().toString().padStart(2, '0');  
        return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;  
      }  
    }  
  }  
```  
> 使用ES6中的字符串新方法 String.prototype.padStart(maxLength, fillString='') 或 String.prototype.padEnd(maxLength, fillString='')来填充字符串；

### 全局过滤器

```

// 定义一个全局过滤器

Vue.filter('dataFormat', function (input, pattern = '') {

var dt = new Date(input);

// 获取年月日

var y = dt.getFullYear();

var m = (dt.getMonth() + 1).toString().padStart(2, '0');

var d = dt.getDate().toString().padStart(2, '0');

// 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日

// 否则，就返回 年-月-日 时：分：秒

if (pattern.toLowerCase() === 'yyyy-mm-dd') {

return `${y}-${m}-${d}`;

} else {

// 获取时分秒

var hh = dt.getHours().toString().padStart(2, '0');

var mm = dt.getMinutes().toString().padStart(2, '0');

var ss = dt.getSeconds().toString().padStart(2, '0');

return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;

}

});

注意：当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用！

## 键盘修饰符以及自定义键盘修饰符

### 1.x中自定义键盘修饰符【了解即可】

```

Vue.directive('on').keyCodes.f2 = 113;

```

### [2.x中自定义键盘修饰符](https://cn.vuejs.org/v2/guide/events.html#键值修饰符)

1. 通过Vue.config.keyCodes.名称 = 按键值来自定义案件修饰符的别名：

```

Vue.config.keyCodes.f2 = 113;

```

1. 使用自定义的按键修饰符：

```

<input type="text" v-model="name" @keyup.f2="add">

```

## [自定义指令](https://cn.vuejs.org/v2/guide/custom-directive.html)

1. 自定义全局和局部的 自定义指令：

```

// 自定义全局指令 v-focus，为绑定的元素自动获取焦点：

Vue.directive('focus', {

inserted: function (el) { // inserted 表示被绑定元素插入父节点时调用

el.focus();

}

});

// 自定义局部指令 v-color 和 v-font-weight，为绑定的元素设置指定的字体颜色 和 字体粗细：

directives: {

color: { // 为元素设置指定的字体颜色

bind(el, binding) {

el.style.color = binding.value;

}

},

'font-weight': function (el, binding2) { // 自定义指令的简写形式，等同于定义了 bind 和 update 两个钩子函数

el.style.fontWeight = binding2.value;

}

}

```

1. 自定义指令的使用方式：

```

<input type="text" v-model="searchName" v-focus v-color="'red'" v-font-weight="900">

```

## 键盘修饰符以及自定义键盘修饰符

### 1.x中自定义键盘修饰符【了解即可】

```

Vue.directive('on').keyCodes.f2 = 113;

```

### [2.x中自定义键盘修饰符](https://cn.vuejs.org/v2/guide/events.html#键值修饰符)

1. 通过Vue.config.keyCodes.名称 = 按键值来自定义案件修饰符的别名：

```

Vue.config.keyCodes.f2 = 113;

```

1. 使用自定义的按键修饰符：

```

<input type="text" v-model="name" @keyup.f2="add">

```

## [自定义指令](https://cn.vuejs.org/v2/guide/custom-directive.html)

1. 自定义全局和局部的 自定义指令：

```

// 自定义全局指令 v-focus，为绑定的元素自动获取焦点：

Vue.directive('focus', {

inserted: function (el) { // inserted 表示被绑定元素插入父节点时调用

el.focus();

}

});

// 自定义局部指令 v-color 和 v-font-weight，为绑定的元素设置指定的字体颜色 和 字体粗细：

directives: {

color: { // 为元素设置指定的字体颜色

bind(el, binding) {

el.style.color = binding.value;

}

},

'font-weight': function (el, binding2) { // 自定义指令的简写形式，等同于定义了 bind 和 update 两个钩子函数

el.style.fontWeight = binding2.value;

}

}

```

1. 自定义指令的使用方式：

```

<input type="text" v-model="searchName" v-focus v-color="'red'" v-font-weight="900">

```