/\* eslint-disable no-new \*/

禁止eslint对new声明进行检测

v-on修饰符：(v-on.xxx)

.prevent: 阻止默认事件

.stop： 阻止冒泡  
v-model修饰符：

.lazy: 当输入工具失去焦点时才更新显示区数据

.trim: 去掉数据两侧的空格

.number： 将数据转换为数字类型

Computed（计算属性）和methods区别：

1. 前者具有缓存机制，如果参与计算的值没有变化，会直接调用缓存的数据，而后者的需要每次重新调用

Prop：[“”] 用于接收外部向组件内部传的参数

子组件向父组件喊话需要通过事件-$emit，子组件通过

this.$emit(‘父组件监听事件’，参数)触发父组件待机的事件

Vue.component('father',{

template: '<div><son ③@son-father="showMoney"></son><br/><span v-if="showNum">当前余额：￥{{money}}</span></div>',

data(){

return {

showNum: false,

money: 0,

}

},

methods: {

④ showMoney(data){

let {money} = data;

this.showNum = true;

this.money = money;

console.log(data)

}

}

})

Vue.component('son',{

template: '<button ①@click="sayFather">展示</button>',

methods: {

sayFather(){

②this.$emit('son-father',{a:1,b:2,money:50});

}

}

})

同级传话

//传话器

①let EVENT = new Vue();

Vue.component('huahua',{

template: '<div>i say: <input tpye="text" ②@keyup="on\_change" v-model="i\_say"/></div>',

data(){

return {

i\_say: ''

}

},

methods:{

on\_change(){

③EVENT.$emit('huahua\_say\_something',this.i\_say);

}

}

})

Vue.component('cyj',{

template: '<div>huahua say: {{huahua\_say}}</div>',

data(){

return {

huahua\_say: ''

}

},

④mounted(){

let me = this;

⑤EVENT.$on('huahua\_say\_something',data=>{

me.huahua\_say = data;

})

}

})

①两个毫不相关的组件喊话需要传话者；

②在huahua组件发生变化时创建相关函数；

③由传话者EVENT创建一个huahua\_say\_something函数将huahua\_say带走

④mounted是一个组件挂载完毕后需要执行的工作

⑤让组件cyj在挂载完毕后监听（$on）huahua\_say\_something这个函数，并将带来的data传给cyj组件中的变量

Vue过滤器

<body>

<div id="box">

<input type="text" v-model='num'><br>

②{{num | showNum('快')}}

</div>

</body>

<script src="./bower\_components/vue/dist/vue.js"></script>

<script>

①Vue.filter('showNum',(val,suffix)=>{

val = val || 0;

suffix = suffix || '元'

return val+suffix;

})

没什么特别的，就是简单得对输出的数据加以修饰，如果内部逻辑太过于复杂，还是建议使用computed计算属性

Vue自定义指令

<div id="box">

<div ①v-pin='false' class='liked'></div>

</div>

</body>

<script src="./bower\_components/vue/dist/vue.js"></script>

<script>

②Vue.directive('pin',(③dom,attr)=>{

if(attr.value){

dom.style.position = 'fixed';

}

})

③第一个参数是挂载该指令的Dom元素，第二个是指令的相关信息，包括传入的数据等，然后通过回调函数来实现指令的功能

<div v-pin①.right.bottom='true' class='liked'></div>

①自定义指令传入的修饰符也可在上面的第二个属性attr中获取到

Mixins（混合）

①let dase = {

methods: {

aler(){

alert(this.name);

}

},

data(){

return {

name:'崔永杰'

}

}

}

Vue.component('show',{

template: '<button @click="aler"></button>',

②mixins: [dase],

data(){

return {

name: '猴子'

}

}

})

其实就是组件间公共的属性和方法，那个组件需要，使用

Mixins: [公共对象],引入即可，这样组件在查找完自己的东西之后，如果没找到，就会去公共对象里找

Slot 插槽

<div id="box">

①<show slot="content">哈哈</show>

</div>

</body>

<script src="./bower\_components/vue/dist/vue.js"></script>

<script>

Vue.component('show',{

template: `<button>

②<slot >

③<div slot="content">你好(默认内容)</div>

</slot>

</button>`,

})

①插槽就是可以向组件内部插入数据的一个入口(不加插槽，组件标签内部是无法写东西的)；

②通过在template内写<slot>标签来接收传入的数据；

③通过slot=””属性来接收不同的数据，同时可以声明默认内容

2.6具名插槽用法

<div id="box">

<show >

<template v-slot:content>

Nihao(会被传入到name为content的插槽中)

</template>

妈的(不指名会被插入default默认插槽中)

</show>

</div>

</body>

<script src="./bower\_components/vue/dist/vue.js"></script>

<script>

Vue.component('show',{

template: `

<div>

<button>

<slot name="content"></slot>

</button>

<slot></slot>

</div>`,

})

<template slot='b' slot-scope='b' //利用这个属性开始获取属性>

<div :class="b.a">

GK,nmlgb,{{b.class}}

{{b}} // 输出一个对象{b:, a:, c:…}

</div>

</template>

// 插槽内部还可以传入属性，可供上面的标签调用

<slot name='b' :b='publish\_data' a='active' c='哈哈'></slot>