Vue3

ES5循环方法：reduce（回调函数，初始值）：

当需要计算、统计数组中的每项的结果的时候使用它，这一次的运算，需要上一次的计算累积。

```
let arr = [1, 2, 3, 4, 5]

arr.reduce((a,b)=>{ // a 代表上一次的返回结果（如果没有return，那么就为undefined），b为本次循环的值

    console.log("a:"+a, 'b:'+b)
})
```

VS插件：VS Live Server -> 服务器。

生命周期函数（钩子函数）： 

难点：

- 生命周期函数，他们啥时候用
- 组件的传递（重点）

created：

- 是Vue数据初始化之后，调用的函数。（是new Vue( )下面的一个属性），里面的this指向 Vue 。
- 当要请求数据的时候，就走created。

这些框架的优势：

让前端更加关注数据，
当数据发生变化的时候,Vue自身会进行一些运算，对比出差异，更新与最近数据对应的视图。

watch：（指定数据的监听器）

当指定数据发生变化的时候触发，一开始是不会触发的，只有数据发生变化时才会触发。（Vue下面的一个属性）

如果

```
//  是函数的时候，只能监听一层 [一层]，深层（多层）({多层})就监听不到了，只能用下面的方式。
new Vue({
    el:"#app",
    data:[
        a:"张三"
    ]
})
    watch:{
        a(val,oldval){   //  此处的a  要和绑定的数据名称一样。（前几行data中的"a"）
            // val是改变后的值，oldval是改变前的值
            localStorage.srtItem('name', val) // 这是将新得到的值存到浏览器缓存中
        }
    }

function



// 是对象格式的时候  适用于多层数据内容嵌套时。
new Vue({
    el:"#app",
    data: {
        arr:
        //arr:[{checked:true},{checked:true},{checked:true}]
    }
    watch: {
        arr:{
            handler(val,oldval){
                localStorage.srtItem('name', val)
            },
            deep:true
        }
    }
})

function getStorage(name){
    let data = localStorage.getItem(name) || '[]';
```

```
    return
}
```

组件化开发

组件化开发：主要是把一个大的功能拆分成若干个小的功能，解决高耦合的问题，方便开发人员维护。（页面的每个部分都分为一个单独的vue组件，拥有自己的vue 文件，引入这个vue组件文件就能行了）

Vue的组件： component

component中的数据传递（容易晕）

Vue.component(组件的名称， 对象（描述组件的情况））–> 必须放在根实例的上面（参照，下面代码部分的前后顺序）

小技巧：创建组件时的名字和引入组件的名字，尽量一样。

- 比如：Vue.component(‘ppx’) –> 创建名字
- `<ppx></ppx>` –> 引入组件的名字。

组件名称命名规则：最好不要使用驼峰命名法（大小写组合）
如果要用驼峰命名，引入组件名字时，大写变小写，中间带 -

- 比如：Vue.component(‘ppX’) –> 创建名字
- `<pp-x></pp-x>` –> 引入组件的名字。

写组件的步骤：

- 1、Vue.component(‘ppa’，{ })
- 2、new Vue( )
- 3、引入：< 组件的名字 >

```
<div>
    <ppa></ppa>            // 第三步 引入组件。
</div>

Vue.component('ppa', {    // 第一步：创建组件 这里面的"ppa"为创建的组件名
    template:`
        <div id='box'>
            <div>放饭辣</div>
            <h2>晚餐</h2>
```

```
            </div>`
    data(){                 // 里面存放数据 在子组件中 data是一个函数，里面的
数据直接写在return中。
        return{
            val:"这个是子组件"
        }
    }
    methods:{               // 操作数据的方法放在这个 methods中。
        fn(){
            alert("11")
        }
    }
    computed
})

new Vue({                   // 根实例 第二步new Vue（）
    el:"#app",

})
```

> **在组件的 template 中，顶层只能有一个元素，**比如上面代码的 #box
> 子组件里面的data必须是一个函数，函数中返回一个**retrun后面放数据内容。**

数据传递：（容易迷 ⭐⭐ ）

传递数据：第一种方法（子传子 html上的组件名传给组件数据）

- 1、在组件上挂一个自定义的属性。

  > 此处挂自定义属性的时候的注意点：
  >
  > - 正确的挂法：`<ppa sex="boy"><ppa>`
  > - 错误的挂法：`<ppa :sex="boy"><ppa>` 这个挂法是引用父级的数据时候的挂法，在前面有一个”：“这个是v-bind的简写。因为两者相近所以要注意。

- 2、子组件上设置一个props，用来接收数据。
  - props可以为一个数组，数组里面写自定义属性的名字即可。
  - 比如：`<ppa d='传进来的数据'></ppa>`

```
<div>
```

```
    <ppa d='传进来的数据'></ppa>
</div>
Vue.component("ppa",{
    props: ['d'],
    template:`<div>{{d}}</div>` // 这里的 d就是自定义属性的名
})
```

- - props（组件中接受数据的属性）可以为数组，也可以为对象，**数组的时候传字符串，使用对象的时候，是为了启用高级配置（传入数据类型的检测，自定义校检和设置默认值）**
    在传入数据的时候。类型检测会帮助你更快的锁定问题（故意报错），

```
d:{
    type: Number,    数据类型
    default: 11      如果子组件上没有传入这个数据，默认走default。
}
```

实例代码：

```
// 第一种方法: 引入组件时，传入数据，组件中接受

<div id="app">
        <hui d="传的"></hui>     // 引入组件，传入数据"d"
</div>
<script src="./vue.js"></script>
<script>

Vue.component('hui', {
    props:['d'],                // 这是组件中，  接受数据的位置
    template:`
       <div>
           <div>放饭辣</div>
           <h2>晚餐</h2>
           <div>{{a}}</div>
           <button>{{d}}</button>
        /div>`,
     data(){
          return {
              a:'666',
          }
     }
})
new Vue({
    el:'#app'
})
```

```
</script>
```

传递数据（父传子）父级数据传传递给子级：

- 1、在父级数据中有一个数据。”arr”
- 2、通过在子组件的身上加一个v-bind：自定义属性名 = 父级的数据
  `<ppa v-bind:data="arr"> -> 简写 <ppa :data="arr">`
- 3、子组件通过props去接受 `props:['data']`
- 4、使用：{{ d }}

```
// 父级数据传给子级。

<div id="app">
    <ppa :data="arr"></ppa>    // 2、引入子级时，引入父级数据 此处注意用
":"
</div>

let obj = {
    props:['data'],         // 3、子级拿到引入时传递进来的父级数据
    template:`
        <ul>
            <li v-for="(val, key) in data">{{val}}</li>  //  4、使用
父级传入的数据，data就是父级传入的数据。
        <ul>`
}

Vue.component('ppa', obj)

new Vue({
    el:"#app",
    data:{
        arr:[111, 222, 333]    //  1、父级的数据
    }
});
```

修改数据：子级修改父级数据

子级使用父级数据的基础上，修改数据：子改父（使用父级数据，子级将修改内容传给父级，父级修改人，然后子级再使用）

第一种方法（父级上面有一个函数（用来修改数据），在引入子级组件时使用一个自定义函数引入父级中这个用来修改数据的函数。）

- 1、在父级写一个改变数据的函数。（数据的改变根据这个函数内的操作改变）
- 2、在子组件上绑定一个自定义的事件，并且传入父级的数据改变函数。

    - 比如：`<ppa @changeboolean="changC">` （@changeboolean 是子组件定义的，changC是父组件中定义的）
- 3、在子组件内部监听这个自定义的事件
    - this.$emit(自定义的事件名，参数)
- 4、绑定，将这个函数与需要改变数据的元素绑定，

> **注意：**此处改变数据的函数名容易弄混，
>
> - 此处会有三种函数名：1、父级的函数名，2、引入时的函数名，3、子级的函数名，此处何时使用哪种函数名容易混淆，导致无法进行操作。
> - 1、父级的函数名：当在父级定义这个函数时使用、以及在引入组件向子级传这个数据时使用，
> - 2、引入的函数名：当在引入组件，向子级传这个函数时使用，以及在子级接收这个函数时使用。
> - 3、子级的函数名：在子级中接收后，定义子级中的函数时，和于元素绑定时使用子级的函数名。
>
> **总结：三个函数名是分阶段，每换一个阶段就换一个适应当时环境的函数名**

实例代码：

```
// 第一种方法
<body>
    <div id="app">
        <h2>父级的</h2>
            <input type="text" v-model="val" @keyup.13="add">
            {{arr}}
        <hr>

        <h2>子组件</h2>
        <list :data="arr" @changboolean="changC"></list>  // 2、使用自定义函数，传入父级用来修改数据的函数。
    </div>
<script src="./vue.js"></script>
<script>
    let obj = {
        template:`
            <div>
                <ul>
                    <li v-for="(val, key) in data">
                        <input
                            type="checkbox"
                            @change="change(val.id, $event)"  // 4、函数与元素绑定。--> 操作元素-改变数据
                            >
```

```
                        <span>{{val.txt}}</span>
                    </li>
                </ul>
            </div>`,
        methods:{
            change(id, ev){
                this.$emit('changboolean', id, ev.target.checked);
// 3、使用"this $emit"接收父级修改数据的函数，后面的参数是需要传入的参数
            }
        },
        props:{   // 接受传入的数据
            data:{
                type:Array,
                default:[]
            }
        }
    }

    Vue.component('list', obj)

    new Vue({
        el:'#app',
        data:{
            val:'',
            arr:[
                {
                    id:0,
                    txt:'马国耀',
                    checked: false
                },
                {
                    id:1,
                    txt:'张三',
                    checked: false
                },
                {
                    id:2,
                    txt:'李四',
                    checked: false
                }
            ]
        },
        methods:{
            add(){
                this.arr.push({
                    id:+new Date,
                    txt:this.val,
                    checked:false
                });
```

```
                this.val = '';
            },
            changC(id,bool){        // 1、父级中用来修改数据的函数
                this.arr.forEach(e=>{
                    if (e.id === id) {
                        e.checked = bool;
                    }
                })
            }
        }
    })
</script>
```

第二种方法：将父级数据复制过来一份使用（数据为复合类型需要用深度克隆）

如果要让子级有功能，那么可以把父级传进来的数据变成子级的，子级改变自己的数据，是不会影响父级的数据的。（这样父级和子级的数据都是独立的，互不影响），如果想要互相的连动，就还需要用事件函数相互绑定改变数值。方式和第一种绑定的类似。

操作步骤：

- 1、父级上有一个数据
- 2、接受数据： ：data =”arr”
- 3、 props：[‘data’]
- 4、在data的return中，重新给一个变量名，并赋值为传入的这个数据
- 4、data(){
  return{
  cd：this . data
  }
  }
- 5、使用自己的数据：`<div>{{cd}}</div>`

以上操作子级和父级的数据不会绑定，是相对独立的，如果需要绑定的那就就用第一种方法吧。

- **注意：如果父级传进来的数据是复合类型的，变成子级的数据时，需要深拷贝一下。不然改变子级数据，会影响到父级。**
  深度拷贝： JSON.parse(JSON.stringify(this.data)) –> data 是props中的数据名

注意：有时会想到直接在子级上修改数据，这是不允许的！！或许数据会变更，但是控制台会报错，这个想法是错误的。

> 数据：
> 双向数据绑定：数据和视图（绑定）
>
> - v-model –> 将数据和视图绑定到一块。

单向数据流动（父级数据和子级数据的流动）

- 数据从父级流向子级，数据本身还是父级的，如果操作子级要改变父级的数据，只能通过子级告知父级要操作那个数据，然后让父级去修改数据。（不允许子级直接修改父级的数据 –>子级传递修改信息，父级修改后再传回子级。）