vue框架(MVVM框架)

- MVVM： Model View ViewModel

vue使用

```
let vm = new Vue({
    el:'#挂载的元素名',
    data:{
        在new Vue中data的值是一个对象，对象里面就可以设置初始化的数据
    }
})
```

- 在new Vue中，data的值是一个**对象**

vue指令

v-if

- v-if=”数据|条件”
  - true为渲染在页面
  - false为不渲染页面
- v-else-if =”数据|条件”
- v-else

> 注意：前一兄弟元素必须有v-if 或 v-else-if
> 如果触发频繁会影响性能

v-show

- v-show = “条件”
- 成立：display:block; 否则为none

```
<div style="display: none;">12321321</div> == $0
```

```
        <div v-show="color=='red'">12321321</div>

        <!-- <div v-text="num"></div> -->
        <div v-html="dom"></div>
    </div>
<script src="./vue.js"></script>
<script>

    let v = new Vue({
        el:'.root',
        //等同于设置this.state
        data:{
            num:456,
            b:true,
            color:'red',
            dom:'<div>哈哈</div>'
        }
    });
```

- 如果满足条件，就显示出来，否则为display：none

v-text

- v-text相当于innerText

v-html

- v-html相当于innerHtml

v-for

- v-for=”(val,key) in arr” 数组 val就是数组中的值 val为数组中的值，key为数组中的索引
- v-for=”(val,key) in obj” 对象 val为对象中的值，key为对象中的名

v-on

- 不带on的事件名
- 缩写：@ 比如：@click
- 修饰符
  - 例如：
    - .13 或者.enter 回车
    - .stop 阻止冒泡
    - .prevent 阻止默认行为

v-bind 绑定行间属性

- 缩写： ：
- 如果属性为value=”“,src=”“,href=”“… 里面的值是静态的(写死那么不需要v-bind)
- 如果属性的值是通过数据得到，那么需要加上v-bind

```
v-bind:value
v-bind:href
v-bind:src
v-bind:style
```

v-bind 可以缩写为:

v-model 表单 双向数据绑定 （只要是表单就要立马想到是否使用v-model)

methods

- 事件函数应该写在methods中

```
new Vue({
    methods:{
         方法1,
         方法2
     }
})
```

- 事件中可以绑定函数
    - click=”fn” fn没有参数，默认的参数为event
    - click=”fn(key)” fn如果没有参数，那么key就是传进来的参数
    - 如果又要传参又想获取event，那么需要在函数中写**$event**
      -@click=”fn(key,$event)”

computed

- 计算属性
- 如果需要通过data的数据派生出别的结果这个时候就想到computed，并且还要让第一次运行，接下来每次修改接着运行

```
 computed:{
     fn(){
          return this.arr.every(item=>item)
      },
      all:{
          get(){
              return this.arr.every(item=>item)
          },
          set(val){
              val就是当修改数据时的值
          }
      }
  }
```

数据截持（defineProperty）

- 例:

```
let obj = {
       num:2
```

```
    }
   //让其consloe.log出来为true
  console.log(obj.n < 3 && obj.n >= 4);
```

- 做:
- get 读 set

```
<script>
    let obj = {
        num:2
    }

    let v = 1;
    //去设置属性
    Object.defineProperty(obj,'n',{
        // value:0,
        get(){
            console.log('拦截获取'); //读操作
           return v*=2; //读的时候进行拦截
        },
        // set(val){ //写属性的时候进行拦截
        //     // console.log(val);
        //     v = val;
        // }
        // writable:true, //是否能够修改这个属性，默认为不能修改
        // enumerable:true,//n这个属性是否能被枚举
        // configurable:true,//是否能被删除
        // writable:false
    });

    //在操作某个属性的时候，进行拦截

    // obj.n = 10;
    // delete obj.n;
    // console.log(obj.n);
    // obj.n = 30;
    // for(let attr in obj){
    //     console.log(attr);
    // }

    // console.log(obj.n);

    console.log(obj.n < 3 && obj.n >= 4); //true

</script>
```

watch

- 监听data数据的变化，只要变化就触发，第一次是不会触发的，只有改变了之后才触发
- 写法：

```
data:{
    arr:[]
},
watch:{
    要监听的数据名:函数或者对象
    比如:

    arr(newValue,oldValue){
        //当arr发生变化的时候做的事情
    }

    如果数据要深层监听要用对象的方式
    比如:
    arr:{
        handler:function(newValue,oldValue){
            //当arr发生变化的时候做的事情
        },
        deep:true
    }
}
```

组件

- 在vue中有一个**component**的方法来创建组件

```
 Vue.component({
    template:'<div>{{num}}</div>',
     data(){
         return {
             ary:[],
             num:0
         }
     }
 })
```

> 注意：
> 方法必须写在new Vue的上面

> 要记得插入子组件
> **data的值为函数，这个函数必须返回一个对象，对象的值就是初始化数据**
> 组件名字要么**小写**，要么烤串命名
> **组件顶层只能有一个元素**

子组件的数据流动

> vue是单向流动 -> 父级的数据传递给子级，如果需要通过操作子级修改数据，只能是父级修改（因为数据是从父级流向子级的，子级是不能把流动数据给父级的）

- 双向数据绑定 -> 数据驱动视图，视图操作数据

传递的方式

- 第一种：
  - 1.传递：
    - 通过在子组件上挂一个自定义的属性，如果传递的值是一个静态的，那么需要加上v-bind
      - 比如：pn=”1” 这个1就是静态的
    - 如果传递的值是一个动态的，那么这个属性必须是被v-bind所绑定
      - 比如： :pn=”num” 这个num是父级或者可能需要变化的
- 2.接收：
  - 父级定义一个改变数据的方法
  - 子组件内使用this.$emit 去定义一个自定义的事件
    - 比如：this.$emit(‘tongdao’,可以加参数);
  - 在子组件上绑定tongdao这个事件，并且把父组件的改变数据的方法添加到自定义事件上
- 第二种：
  - 把父级的数据变成子级自己的，跟父级断绝关系
    - 1.通过props去接收父级的数据
    - 2.把接收的数据存到自己的data中

```
{
    props:['nn'],
    data(){
        return {
            cnum:this.nn
        }
    },
    template:`<div>{{cnum}}</div>`
}
```

- 如果想父级的数据根据子级的数据变化，那么，父级需要定义的一个方法去接收子级传过来的数据
  - 比如:

```
getchildfn(val){
    this.num = val;
}
```

  - 然后通过事件绑定的方式把getchildfn传给子级

    自定义名字=”getchildfn”
  - 最后当修改子级数据的时候，创建一个自定义名字的事件，并且把数据传进去
    - 比如：

```
this.$emit('自定义名字',要传给父级的数据)
```