组件语法

： **:xxx="appname" :xxx就是一个变量，来直接传字符串值**

可以将组件中定义好的数据直接引用来传递赋值

```
    <!-- AS是一个组件，组件传值  加 ：  直接引用组件中已定义的数据值
         不加 ：  传递的就是一个普通的字符串  -->
    <AS :ns="appname"></AS>
    <AS ns="appname"></AS>
</template>
<script>
    export default {
        name: "A",    //组件名
        // 其它js方法及，生命周期函数，监听器，数据等
        data(){            // 组件所有数据都在data中创建
            return {
                appname:'我是你大页vue',
                showname:true,
```

单文件组件结构

.vue是vue单文件组件，一个单文件组件由三部分组成：<template>，<script>，<style>

```
<template>
    <!-- 写html  -->
</template>
<script>
    export default {
        name: "A",    //组件名
        props:{        //  申明外部可以给那些字段来传数据 用来做数据接收传递的
            name: {  //申明数据类型
                type: String,
                required: true    //必须的
            }
        },
        components:{       //注册别的组件
            // 导入组件名
        },
        // 其它js方法及，生命周期函数，监听器，数据等
    }
</script>
<style scoped>
    <!-- 写样式 -->
</style>
```

单文件组件中的数据定义及使用

在<script>中定义一个data()用来定义申明组件中需要的数据

```
<script>
  export default {
      name: "A",    //组件名
      // 其它js方法及，生命周期函数，监听器，数据等
      data(){           // 组件所有数据都在data中创建
          return {
              appname:'我是你大页vue',
              showname:true,
              goods:[
                  //  {text:'java',pice:4100},
                  //  {text:'android',pice:1500},
                  //  {text:'webss',pice:2100},
                  //  {text:'vue',pice:1200},
                  //   {text:'小程序',pice:1900}
              ],
          }
      },
  }
</script>
```

单文件组件中数据的使用：

1、在<template>中用 {{数据名}} 使用

```
<template>
  <!-- 写html 组件数据用{{}} 来显示使用 -->
    {{appname}}
</template>
```

2、在<script>中的其它函数里用 this.数据名 来使用

```
setName(){
    this.appname = '99999'
},
```

声明式渲染

```
<body>
<div id="app">
   <input type="text" v-model="name">
    {{name}}
</div>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
    // 申明式渲染就是在html页面中直接引用vue.js
    var app = new Vue({
        el:'#app',
        data:{
            name:'我是纯浏览器体验，测试的'
        }
    })
</script>
</body>
```

条件与循环

条件语句 v-if="xxxx"

 {{name}}

数组数据

goods:[{text:'java',pice:4100},

{text:'android',pice:1500},

{text:'web',pice:2100},

{text:'vue',pice:1200},

{text:'小程序',pice:1900}],

循环语句 v-for="(good,index)in 数据数组名"

<li v-for="(good,index)in goods" >

{{good.text}}

.....

数据绑定，样式绑定与事件

v-model="t"

<input type="text" v-model="t"/> 数据绑定

在**<script>**的方法中使用 **this.t** 来获取或赋值

```
addgoods(){
    //  判断输入框中是否有值   通过v-model="t"来获取到输入框对象
    if (this.t) {
      // ……
    }
    this.t = ""
}
```

:class="{active:c.active}" 样式绑定

```
tr.active{
    color:red;
}
```

@click 事件

@click="事件方法名**"**

<button @click="addgoods2()">添加</button>

组件的引用注册使用

创建组件：

新创建的组件，在组件的**<script>**中一定要**export default {}** 只有做了导出声明的组件才能导入到别的组件中使用

```
<script>
export default {
    name: "Cats",
}
```

导入组件：在需要使用的组件的**<script>**中导入新创建的组件

import 引用名 from './components/单组件名.vue'

```
import Cats from './components/Cats.vue'    //导入组件
```

注册组件：**components:{** 引用名 **}**

```
import  Cats from './components/Cats.vue'    //导入组件
import axios from 'axios'
export default {     // 组件暴露出去
  name: 'app',      // 组件名
  components:{       //注册已导入的组件
      Cats
  },
```

```
components:{       //注册已导入的组件
```

```
    Cats
},
```

使用组件：在<template>中使用组件，如使用html标签一样。

<组件引用名　要传递的数据**> </**组件引用名**>**

<Cats name="appname" :goods2="goods2"></Cats>

```
<template>
  <!-- 写html  -->
    {{appname}}
  <!-- 引用组件-->
  <Cats :ns="appname"></Cats>
</template>
```

组件中数据类型的约定

在新建中通过 **props:{}** 来申明将来使用这个组件要传什么数据进来

```
<script>
export default {
   name: "Cats",
    //  用来做数据接收传递的
   props:{
       //申明数据类型  申明外部可以给那些字段来传数据
       name:{
         type:String,      // 字符串类型
         required:true    //必须的
       },
        goods2:{
           type:Array     //数组类型
       }
   },
```

在新组件中，申明**props:{}**数据后的使用

```
<template>
  <!-- 写html  -->
    {{appname}}
  <!-- :name  必传的 :goods2可以不要 -->
  <Cats :name="appname" :goods2="goods2"></Cats>
  <Cats :name="appname" ></Cats>
</template>
```

axios网络请求

安装：npm install axios --Save

引入：在<script>中导入　import axios from 'axios'

使用：async/await语法

const res = await axios.get('http://47.93.241.158/Antes/api/customer/goods')

```
 async created(){
  const res = await axios.get(' http://47.93.241.158:8080/Antes/api/customer/goods')
}
await this.$axios({
    method:"post",
    url:"http://localhost:8080/Antes/api/customer/goods",
    headers:{
        token:token
    }
})
```

数据持久化一>监听器

```
//对谁要数据持久化方法       对 goods2 数组 里的数据进行持久化
setLocal(){
    window.localStorage.setItem('goods2',JSON.stringify(this.goods2))
},
```

```
 //   监听器
watch:{
    goods2:{
        handler(){
            this.setLocal()
        },
        deep:true
    }
},
```

```
//组件创建后   数据持久化处理   获取持久化数据对goods2数组赋值
this.goods2 = JSON.parse(window.localStorage.getItem('goods2'))||[]
```

组件数据传递

1、总线传递

```
Vue.prototype.$bus = new Vue()
```

2、**props:{}**申明传递

3、使用 **provide/inject** **------>** 多层嵌套数据传递

总线模式数据传递（订阅/发布）

1、在main.js中创建总线模式

```
//  总线模式   $bus   自定义名
Vue.prototype.$bus = new Vue()
```

2、发布数据

```
addgoods2(i){
    // 函数执行 将数组中数据good 发布出去
    const good = this.goods[i]
    this.$bus.$emit('addgoods3',good)
},
```

3、订阅获取数据

```
// 总线模式接收数据   good
this.$bus.$on('addgoods3', good=>{
    //以下对接收到的数据进行处理
    const ret = this.goods2.find(v=>v.text===good.text)
    if(ret){
        ret.count +=1
    }else{
        this.goods2.push({
            ...good,
            active:true,
            count:1
        })
    }
})
```

<script></script>结构中有的

```
组件生命同期函数，当组件创建后执行，需要一加载就执行方法在这里面写
created(){
    ...........
}
```

```
组件中所有响应的方法在这里面写
methods:{
    ............
}
```

```
用来做数据接收传递的
props:{
```

```
    ..............
},
```

```
定义组件内部数据的
data(){
  return {
    ..............
  }
},
```

```
// 程序自动执行计算方法都在这里面进行
computed:{
    ..............
},
```

```
//  监听器,监听数据对象编化的
watch:{
    ..............
},
```