

vue组件化开发

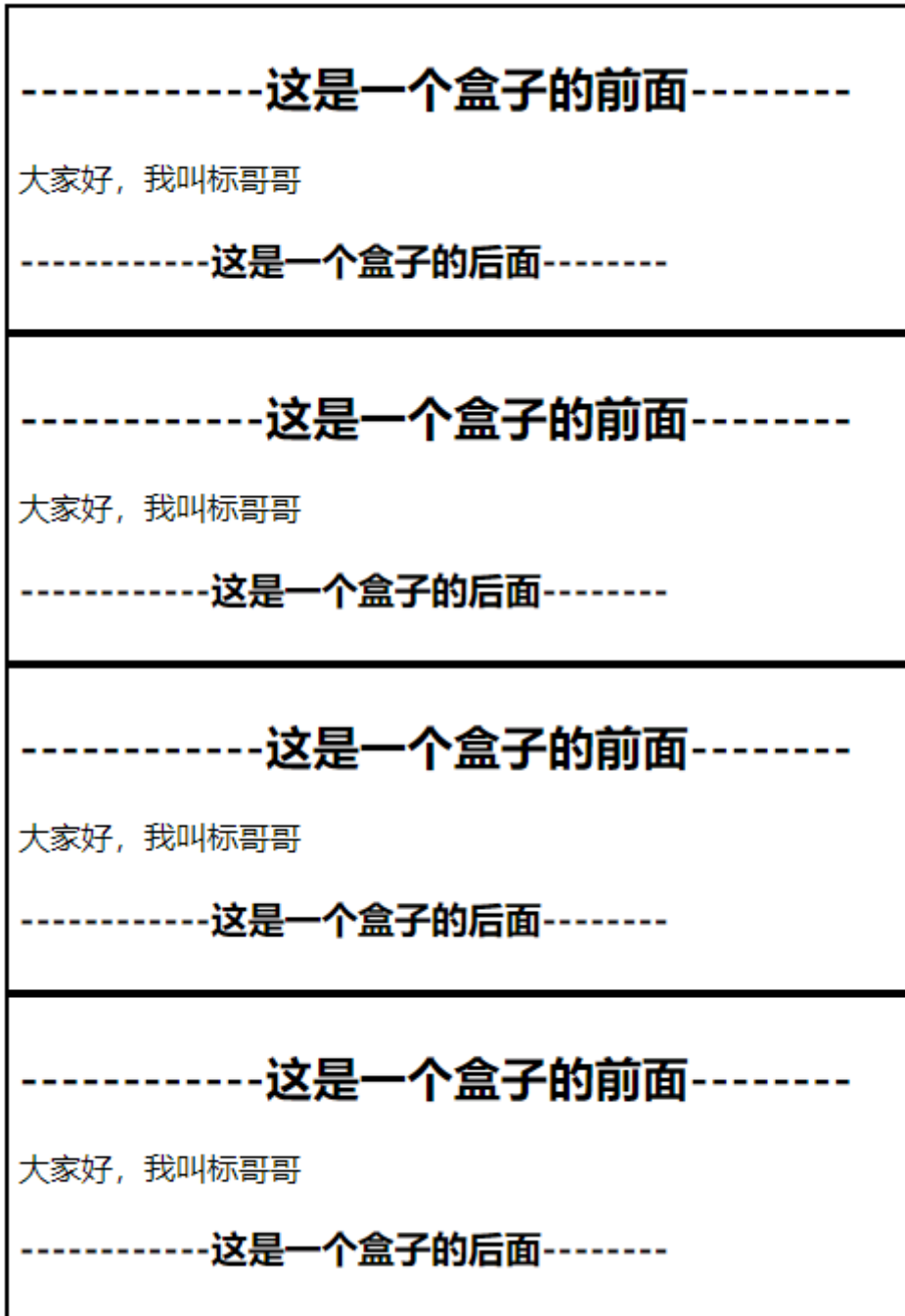
存在的问题

当我们在进行界面开始的时候，我们经常看到过一个问题，如下所示

```
1  <!DOCTYPE html>
2  <html lang="zh">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>目前存在的问题</title>
9      <style>
10         .box{
11             border: 2px solid black;
12             width: 450px;
13             padding: 5px;
14         }
15
16     </style>
17 </head>
18
19 <body>
20     <div id="app">
21         <div class="box">
22             <h2>-----这是一个盒子的前面-----</h2>
23             <p>大家好，我叫标哥哥</p>
24             <h3>-----这是一个盒子的后面-----</h3>
25         </div>
26         <div class="box">
27             <h2>-----这是一个盒子的前面-----</h2>
28             <p>大家好，我叫标哥哥</p>
29             <h3>-----这是一个盒子的后面-----</h3>
30         </div>
31         <div class="box">
32             <h2>-----这是一个盒子的前面-----</h2>
33             <p>大家好，我叫标哥哥</p>
34             <h3>-----这是一个盒子的后面-----</h3>
35         </div>
36         <div class="box">
37             <h2>-----这是一个盒子的前面-----</h2>
38             <p>大家好，我叫标哥哥</p>
39             <h3>-----这是一个盒子的后面-----</h3>
40         </div>
41     </div>
42 </body>
43 <script src="./js/vue.global.js"></script>
44 <script>
45     Vue.createApp({
46
```

```
47     }).mount("#app")
48   </script>
49
50   </html>
```

当我们需要重复的生成某些东西的时候，就会造成大量的代码冗余



我们现在想着的就是怎么样去简化上面的代码

关于virtual DOM概念

能够简化代码最好的办法就是封装。封装这个概念本身并不陌生，如果是JS的封装我们会想到函数 `function`，如果是CSS的封装我们会想到一个公共的class样式，但是好像并没有针对HTML的封装

为了实现这一种HTML方式的封装，所以我们后面提出一个 `virtual dom` 的概念，也就是虚拟DOM的概念，我们可以把要封装的HTML代码当成一个整体的DOM元素。思路如下

```

1   <body>
2     <div id="app">
3       <user-info></user-info>
4     </div>
5
6     <template id="user-info">
7       <div class="box">
8         <h2>-----这是一个盒子的前面-----</h2>
9         <p>大家好，我叫标哥哥</p>
10        <h3>-----这是一个盒子的后面-----</h3>
11      </div>
12    </template>
13  </body>

```

我们要将需要封装的代码变成一个整体部分，然后将它转换成一个 `<user-info></user-info>` 的标签，最后去使用这个标签就可以了呢【这是一种思路】

```

<!DOCTYPE html>
<html lang="zh">
  <head>...</head>
  <body>
    <div id="app" data-v-app>
      <user-info></user-info>
    </div>
    <template id="user-info">...</template>
    <script src="./js/vue.global.js"></script>
    <script> Vue.createApp({ }).mount("#app") </script>
  </body>
</html>

```

默认情况下浏览器是不会有效果的，也不会展示任何内容，并且会报一个警告出来，如下

```

[Vue warn]: Failed to resolve component: user-info
If this is a native custom element, make sure to exclude it
from component resolution via compilerOptions.isCustomElement.
at <App>
vue.global.js:1622

```

它会告诉我们这个 `user-info` 不是一个元素

那么针对这个问题，我们可以把这个 `<user-info>` 去当一个DOM元素，然后再去调用这个DOM元素的时候我们就相当于调用下面的5行HTML代码，这样操作会非常方便

像类似于 `<user-info>` 这样的标签，我们就叫虚拟标签，也叫虚拟dom (virtual dom)

目前可以实现 `virtual dom` 的框架有很多

1. `vue`
2. `react`
3. `angular`

在上面的3个框架里面，`virtual dom` 我们都叫组件

组件的命名

组件名格式

在整个指引中，我们都使用 PascalCase 作为组件名的注册格式，这是因为：

1. PascalCase 是合法的 JavaScript 标识符。这使得在 JavaScript 中导入和注册组件都很容易，同时 IDE 也能提供较好的自动补全。
2. <PascalCase /> 在模板中更明显地表明了这是一个 Vue 组件，而不是原生 HTML 元素。同时也能够将 Vue 组件和自定义元素 (web components) 区分开来。

在单文件组件和内联字符串模板中，我们都推荐这样做。但是，PascalCase 的标签名在 DOM 模板中是不可用的，详情参见 [DOM 模板解析注意事项](#)。

全局组件

vue3的全局组件与vue2的全局组件方式是不一样的

组件也叫 **component** ,组件可以把它看成是一个小型的vue,它也会接管某一个区域

```
1   <body>
2     <div id="app">
3       <aaa></aaa>
4       <aaa></aaa>
5       <aaa></aaa>
6     </div>
7
8     <template id="temp1">
9       <div class="box">
10        <h2>-----这是一个盒子的前面-----</h2>
11        <p>大家好，我叫标哥哥</p>
12        <h3>-----这是一个盒子的后面-----</h3>
13      </div>
14    </template>
15  </body>
16  <script src="../js/vue.global.js"></script>
17  <script>
18    const app = Vue.createApp({
19
20    });
21
22    //注册全局组件
23    app.component("aaa",{
24      // 代表组件接管的区域
25      template:"#temp1"
26    });
27    app.mount("#app");
28  </script>
```

在上面的代码里面，我们可以看到全局组件的注册我们使用的是下面的语法格式

```
1   app.component("组件名", 组件对象);
```

为什么这种方式叫全局组件呢？

```
03为什么叫全局组件.html > html > body
padding: 5px;
width: 450px;

</style>
</head>

<body>
  <div id="app">
    <user-info></user-info>
    <btn-box></btn-box>
  </div>

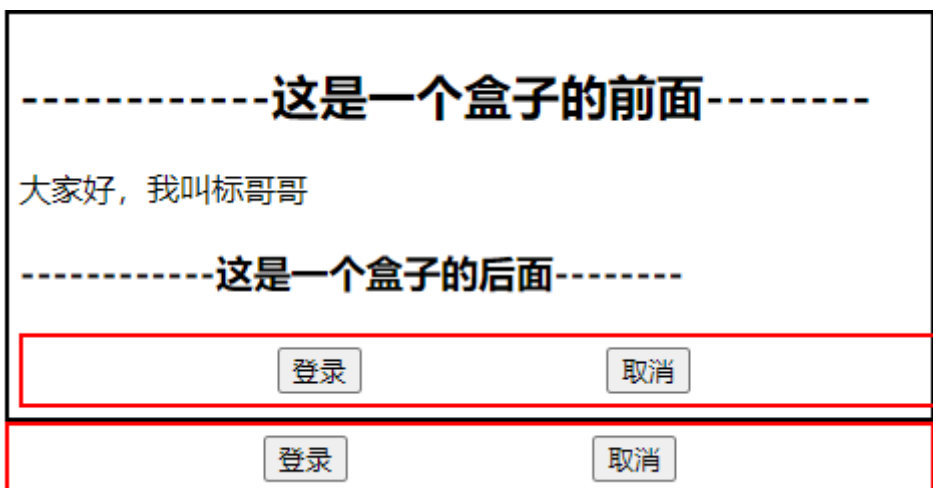
  <!-- 这是第一个组件接管的区域 -->
  <template id="temp1">
    <div class="box">
      <h2>-----这是一个盒子的前面-----</h2>
      <p>大家好，我叫标哥哥</p>
      <h3>-----这是一个盒子的后面-----</h3>
      <btn-box></btn-box>
    </div>
  </template>

  <!-- 这是第二个组件接管的区域 -->
  <template id="temp2">
    <div class="btn-box">
      <button type="button">登录</button>
      <button type="button">取消</button>
    </div>
  </template>
</body>
<script src="/js/vue.global.js"></script>
```

我们可以看到btn-box这个组件在内部与外部都使用了
这说明 全局组件 是在任何地方随意使用的

```
1 //注册全局组件
2 app.component("UserInfo",{
3   // 代表组件接管的区域
4   template:"#temp1"
5 });
6
7 app.component("BtnBox",{
8   template:"#temp2"
9 });
```

当全局组件注册完成以后，可以在任何地方使用，所以它叫全局组（这个观点可以把它联想到全局变量）



局部组件

局部组件的功能与全局组件的功能是一样的，只是使用的范围有限制而已（联想成局部变量）

```
1  <!DOCTYPE html>
2  <html lang="zh">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>局部组件</title>
9      <style>
10         .box {
11             border: 2px solid black;
12             width: 450px;
13             padding: 5px;
14         }
15     </style>
16 </head>
17
18 <body>
19     <div id="app">
20         <user-info></user-info>
21     </div>
22
23     <template id="temp1">
24         <div class="box">
25             <h2>-----这是一个盒子的前面-----</h2>
26             <p>大家好，我叫标哥哥</p>
27             <h3>-----这是一个盒子的后面-----</h3>
28         </div>
29     </template>
30 </body>
31 <script src="../js/vue.global.js"></script>
32 <script>
33     // 组件大写
34     let UserInfo = {
35         template: "#temp1"
36     }
37
38     Vue.createApp({
39         // 在这里注册局部组件
40         components: {
41             // UserrInfo: UserInfo
42             UserInfo
43         }
44     }).mount("#app")
45 </script>
46 </html>
```

局部组件本质上面也是一个对象，只是这个对象要注册在某一个内部的 `components` 里面才可以

为什么叫局部组件

局部组件在使用之前一定要先在内部注册一下，否则不能使用

```
1
2   <body>
3     <div id="app">
4       <user-info></user-info>
5       <btn-box></btn-box>
6     </div>
7
8     <template id="temp1">
9       <div class="box">
10        <btn-box></btn-box>
11        <h2>-----这是一个盒子的前面-----</h2>
12        <p>大家好，我叫标哥哥</p>
13        <h3>-----这是一个盒子的后面-----</h3>
14      </div>
15    </template>
16
17    <template id="temp2">
18      <div class="btn-box">
19        <button type="button">登录</button>
20        <button type="button">取消</button>
21      </div>
22    </template>
23  </body>
24  <script src="../js/vue.global.js"></script>
25  <script>
26    // 组件大写
27
28    let BtnBox = {
29      template: "#temp2"
30    }
31
32    let UserInfo = {
33      template: "#temp1",
34      components:{
35        BtnBox
36      }
37    }
38
39    Vue.createApp({
40      // 在这里注册局部组件
41      components: {
42        // UserrrInfo: UserInfo
43        UserInfo,
44        BtnBox
45      }
46    }).mount("#app")
47  </script>
```

在上面的代码里面，我们分别在 `UserInfo` 及 `Vue` 的 `components` 下面了 `BtnBox` 的组件，这样在这2个地方都可以使用这个 `BtnBox` 的组件了

组件中的数据

组件本身也是小型的vue,所以它的内部的原理与我们前面所学习的是一样的, 它的内部也会有 **data** 数据, 也会有 **methods** 方法, 也会有 **watch** 监听等, 其中最重要的还是组组的数据

1. 组件自身的数据

```
1  <body>
2    <div id="app">
3      <aaa></aaa>
4      <aaa></aaa>
5    </div>
6    <template id="temp1">
7      <div class="box">
8        <h2>大家好 {{nickName}}</h2>
9      </div>
10   </template>
11 </body>
12 <script src="./js/vue.global.js"></script>
13 <script>
14   let aaa = {
15     template: "#temp1",
16     data() {
17       return {
18         nickName: "小姐姐"
19       }
20     }
21   }
22   Vue.createApp({
23     components: {
24       aaa
25     }
26   }).mount("#app")
27 </script>
```

大家好 小姐姐

大家好 小姐姐

2. 组件接收的外部数据



其实在大多数的开发场景下面，我们可以看到，组件的布局是相同的，但是数据是不一样的

组件是可以接收外部传递进去的数据的。数据在传递的时候使用的是**自定义属性传递**

这也是为什么之前跟同学们说过“冒号的属性，@的事件“

```
1 <aaa msg="标哥哥"></aaa>
```

如果我们现在采用上面的方式来完成以后，这个时候我们就可以看到组件上面有一个自定义属性叫 `msg`，这个属性值就是“标哥哥”。在组件的内部就可以实现值的接收

数组语法的接收

```
1 let aaa = {
2   template: "#temp1",
3   // 数组语法
4   props: ["msg", "sex"]
5 }
```

对象语法的接收

```
1 let aaa = {
2   template: "#temp1",
3   props: {
4     msg: {
5       type: String,
6       required: true
7     },
8     sex: {
9       type: String,
10      default: "人妖"
11    }
12  }
13 }
```

在使用对象语法接收的时候，我们可以使用一些描述信息，如 `type` 去指定接收的类型，如 `default` 指定默认值，如 `required` 指定这个值必须传递进来

3.关于组件属性传值的注意事项

这里主要讲解非 `string` 类型传值的时候注意事项，如 `number` 类型，`boolean` 类型的传值

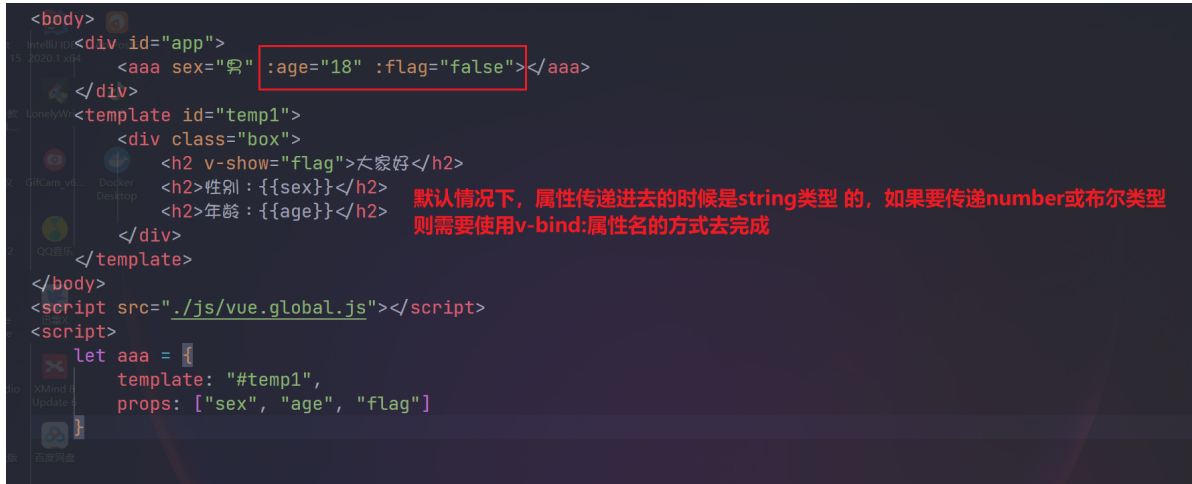
第一种情况：传递属性名的有驼峰怎么办？

```
<body>
  <div id="app">
    <aaa sex="男" nickname="标哥哥"></aaa>
  </div>
  <template id="temp1">
    <div class="box">
      <h2>大家好</h2>
      <h2>性别：{{sex}}</h2>
      <h2>昵称：{{nickname}}</h2>
    </div>
  </template>
</body>
<script src="./js/vue.global.js"></script>
<script>
  let aaa = {
    template: "#temp1",
    props: ["sex", "nickname"]
  }
```

在传递的属性名上面不能使用驼峰命名，如果非要使用驼峰命名，应该使用转义规则

```
<body>
  <div id="app">
    <aaa sex="男" nick-name="标哥哥"></aaa>
  </div>
  <template id="temp1">
    <div class="box">
      <h2>大家好</h2>
      <h2>性别：{{sex}}</h2>
      <h2>昵称：{{nickname}}</h2>
    </div>
  </template>
</body>
<script src="./js/vue.global.js"></script>
<script>
  let aaa = {
    template: "#temp1",
    props: ["sex", "nickname"]
  }
```

第二种情况：关于传递的值的类型的问题



4.全局数据

全局数据可以解决跨组件调用的问题，如父级给孙子组件，如兄弟之前相互给数据，如子级组件给父级。面向这种情况的时候我们可以使用全局数据完成
目前的全局数据方案有以下几种

1. `vuex`
2. `pina`
3. Vue3 自带的 `provide` 和 `inject`

这些技术需要在 SFC 的组件下面讲解

组件的事件及方法

组件本身也算是一个小型的vue，所以它的内部一定会有数据和事件，那么，现在就来看一下组件内部的事件及方法

```
1
2  <body>
3    <div id="app">
4      <h1>组件外部</h1>
5      <aaa nick-name="张珊"></aaa>
6    </div>
7    <template id="temp1">
8      <div class="box">
9        <h2>大家好，这个人叫{{nickName}}</h2>
10       <button type="button" @click="sayHello">按钮</button>
11     </div>
12   </template>
13
14 </body>
15 <script src="../js/vue.global.js"></script>
16 <script>
17   let aaa = {
18     template: "#temp1",
19     props: ["nickName"],
20     methods: {
21       sayHello() {
22         console.log("大家好，我叫"+this.nickName);
23       }
24     }
25   }
26 </script>
```

```

23         }
24     }
25 }
26
27 Vue.createApp({
28     components: {
29         aaa
30     }
31 }).mount("#app")
32 </script>

```

在上面的代码里面，我们可以看到一点，组件内部的事件可以在组件内部处理，这是没有问题的

1. 父级组件调用子级组件的方法【第二种 \$refs】

当父级组件需要调用子级组件的方法的时候，我们可以通过 `$refs` 来找到这个组件，只要找到这个组件以后就可以调用这个子级组件内部的方法了

```

1 <aaa nick-name="张珊" ref="aaa"></aaa>

```

最后在使用的时候，直接通过 `this.$refs.aaa.方法名()` 就可以调用里面的方法

```

1 <!DOCTYPE html>
2 <html lang="zh">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>组件内的数据</title>
9     <style>
10         .box {
11             border: 2px solid black;
12             padding: 5px;
13         }
14
15         .box2 {
16             border: 2px solid green;
17             padding: 5px;
18         }
19     </style>
20 </head>
21
22 <body>
23     <div id="app">
24         <h1>组件外部</h1>
25         <button type="button" @click="m1">外部的按钮</button>
26         <aaa nick-name="张珊" ref="aaa"></aaa>
27     </div>
28     <template id="temp1">
29         <div class="box">
30             <h2>大家好，这个人叫{{nickName}}</h2>
31             <button type="button" @click="sayHello">按钮</button>
32         </div>
33     </template>
34

```

```

35 </body>
36 <script src="./js/vue.global.js"></script>
37 <script>
38     let aaa = {
39         template: "#temp1",
40         props: ["nickName"],
41         methods:{
42             sayHello(){
43                 console.log("大家好, 我叫"+this.nickName);
44             }
45         }
46     }
47     Vue.createApp({
48         methods:{
49             m1(){
50                 //想调用aaa组件内部的`sayHello`方法
51                 // console.log(this.$refs);
52                 this.$refs.aaa.sayHello()
53             }
54         },
55         components: {
56             aaa
57         }
58     }).mount("#app")
59 </script>
60
61 </html>

```

数据流的单向性

数据流的单向性指的是数据只能从外部流向内部，外部改变了，内部也改变了

```

1 <!DOCTYPE html>
2 <html lang="zh">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>组件内的数据</title>
9     <style>
10         .box {
11             border: 2px solid black;
12             padding: 5px;
13         }
14
15         .box2 {
16             border: 2px solid green;
17             padding: 5px;
18         }
19     </style>
20 </head>
21
22 <body>
23     <div id="app">

```

```

24     <h1>组件外部----{{nickName}}</h1>
25     <button type="button" @click="nickName='帅小伙'">外部改变nickName</button>
26     <aaa :nick-name="nickName"></aaa>
27
28   </div>
29   <template id="temp1">
30     <div class="box">
31       <h2>大家好，这个人叫---{{nickName}}</h2>
32     </div>
33   </template>
34
35 </body>
36 <script src="./js/vue.global.js"></script>
37 <script>
38   let aaa = {
39     template: "#temp1",
40     props: ["nickName"],
41   }
42   Vue.createApp({
43     data() {
44       return {
45         nickName: "标哥哥"
46       }
47     },
48     components: {
49       aaa
50     }
51   }).mount("#app")
52 </script>
53
54 </html>

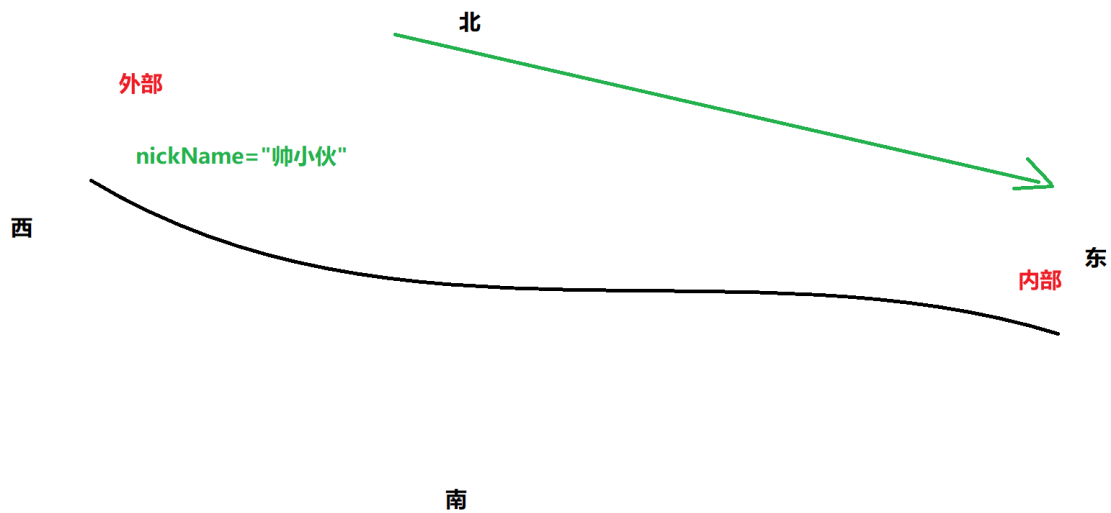
```

组件外部----帅小伙

外部改变nickName

大家好，这个人叫---帅小伙

当我们去点击按钮以后，我们发现外部的数据变了，内部的数据也变化，这是因为数据是有流行向的，由外向内进行传递，**这个过程不可逆**



数据流的单向性就注定了只能是外边改变，里面再改变，不能是里面改变，外边改变。当我们尝试在内部更改这个值的时候就报错了

```
<body>
  <div id="app">
    <h1>组件外部----{{nickName}}</h1>
    <button type="button" @click="nickName='帅小伙'">外部改变nickName</button>
    <aaa :nick-name="nickName"></aaa>
  </div>
  <template id="temp1">
    <div class="box">
      <h2>大家好，这个人叫---{{nickName}}</h2>
      <button type="button" @click="changeNickName">我在内部改变nickName</button>
    </div>
  </template>
</body>
<script src="./js/vue.global.js"></script>
<script>
  let aaa = {
    template: "#temp1",
    props: ["nickName"],
    methods: {
      changeNickName() {
        this.nickName = "岳圣哲";
      }
    }
  }
  Vue.createApp({
```

结果出现了问题

组件外部----帅小伙

外部改变nickName

大家好，这个人叫---帅小伙

我在内部改变nickName

控制台 欢迎 元素 >> + 1 1 1

top 筛选器 默认级别 1

You are running a development build of Vue. [vue.global.js:10938](#)
Make sure to use the production build (*.prod.js) when deploying for production.

[Five Server] connecting... [fiveserver.js:1](#)

[Five Server] connected. [fiveserver.js:1](#)

⚠ [Vue warn]: Attempting to mutate prop "nickName". Props are readonly.

uid: 1, vnode: {...}, type: {...}, parent: {...}, appContext: {...}, ...

破坏数据流的单向性

1. 利用对象的堆栈原理

vue当中进行组件数据传递的时候使用的是 `const` 的原理在锁值，所以当值接收以后，是不可以再更改栈的内容的，但是可以更改堆的内容

而且我们还知道一点，对象在传递的时候传递的是地址（浅拷贝）

```
1  <!DOCTYPE html>
2  <html lang="zh">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>组件内的数据</title>
9      <style>
10         .box {
11             border: 2px solid black;
12             padding: 5px;
13         }
14
15         .box2 {
16             border: 2px solid green;
17             padding: 5px;
18         }
19     </style>
20 </head>
21
22 <body>
23     <div id="app">
24         <h1>组件外部---{{userInfo.nickName}}</h1>
25         <button type="button" @click="userInfo.nickName = '帅小伙子'">外部改变
26         nickName</button>
27         <aaa :user-info="userInfo"></aaa>
28     </div>
29     <template id="temp1">
30         <div class="box">
31             <h2>大家好，这个人叫---{{userInfo.nickName}}</h2>
32             <button type="button" @click="changeUserInfo">我在内部改变
33             nickName</button>
34         </div>
35     </template>
36
37 </body>
38 <script src="./js/vue.global.js"></script>
39 <script>
40     let aaa = {
41         template: "#temp1",
42         props: ["userInfo"],
43         methods: {
44             changeUserInfo() {
45                 this.userInfo.nickName = "岳圣哲";
46             }
47         }
48     }
```



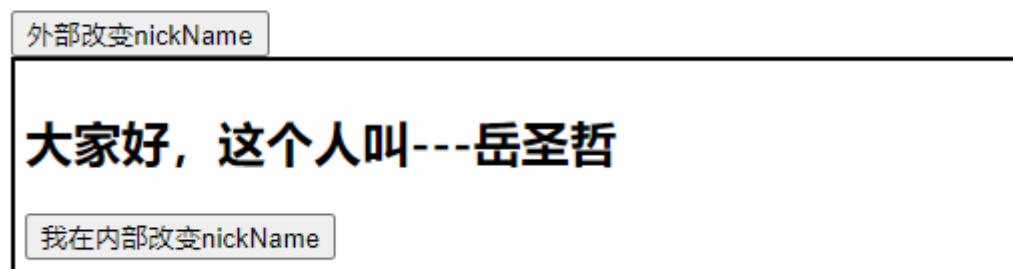
```

48     Vue.createApp({
49       data() {
50         return {
51           userInfo: {
52             nickName: "标哥哥"
53           }
54         }
55       },
56       components: {
57         aaa
58       }
59     }).mount("#app")
60   </script>
61
62   </html>

```

在上面的代码里面，我们本来传递过去的是一个基本数据类型，但是现在我们把它改成了 `userInfo` 对象，它是一个对象类型，对象在传递的时候是地址传递（浅拷贝），同时即使内部使用 `const` 去修改 `userInfo`，也可以更改 `userInfo` 内部的东西

组件外部----岳圣哲



这一种实现方式实现起来比较简单，但是vue的官方推荐我们使用另一种方式

2.利用自定义事件

在vue的内部其实也考虑到了内部改变外部数据的这一种场景，所以它专门提供了一种解决方法，这种解决方案是自定义事件解决方法


```

35         changeMyName(){
36             //通知它爸改名子
37             //触发了某一个事件，emit:发出;射出;散发
38             // 这里它触发了xyz这一个自定义事件
39             this.$emit("xyz");
40         }
41     }
42 }
43
44
45 Vue.createApp({
46     data() {
47         return {
48             daughterName: "杨妞"
49         }
50     },
51     methods:{
52         changeDaughterName(){
53             console.log("我女儿在找我")
54             this.daughterName = "杨柳彤";
55         }
56     },
57     components: {
58         aaa
59     }
60 }).mount("#app")
61 </script>
62
63 </html>

```

标哥的女儿叫：杨柳彤

我是标哥的女儿，我叫---杨柳彤

女儿要改名子

组件的插槽

1. 普通插槽
2. 具名插槽
3. 作用域插槽

vue的组件及生命周期

1.生命周期及钩子函数

2.跨生命周期的调用

3.v-if及v-show的区别

4. keep-alive的使用

5.vue1与vue2生命周期的对比

6.vue生命周期图