4、2父子组件传值

父子件向子组件传值，利用props

<div id="app">

<counter v-bind:counter="0"></counter>

<counter v-bind:counter="1"></counter>

</div>

Js

var counter = {

props: [counter],//接受父组件传过来的值。

template: '<div>{{ counter }}</div>'

}

var app = new Vue({

el: '#app',

component: {

counter: counter

}

})

不能修改父组件传过来的值，因为这样会影响其他的子组件。

解决方案

修改代码如下

var counter = {

props: [counter],

template: '<div>{{ counter }}</div>'

data: function() {

return {

number: this.counter

}

}

}

这样就可以通过修改自己的numbe变量而不影响其他子组件。注意的是，这里的data必须为一个函数，不能是一个对象。

4、3组件校验

Vue.component('child'{

props: {

content: String

}

})

表示从父组件接受的值得类型必须为String否则报错

Vue.component('child'{

props: {

content: [ String,Number]

}

})

表示从父组件接受的值得类型为String或者Number

Vue.component('child'{

props: {

content: {

type: String,

required: true//表示必须传入值

}

}

})

表示从父组件接受的值得类型必须为String，而且必须传入值

传入的值约束

Vue.component('child'{

props: {

content: {

type: String,

validator: function(value) {

return (value.length >5)//传入的值的长度要大于5否则报错。

}

}

}

})

4、4给组件绑定原生事件

<child @click=”handleClick”></child>,监听的自定义事件非原生。如果改成

<child @click.native=”handleClick”></child>

Var app = new Vue({

el: ‘ #app’,

methods: {

handleClick: function(){

alter(‘click’)

}

}

})

4、5非父子组件之间的传值

<div id="app">

<child content="Del"></child>

<child content="Li"></child>

</div>

Js

Vue.prototype.bus = new Vue(); //给prototype绑定一个bus，目的是为了非父子组件间的传值

var child = {

data: function() {

return {

selfContent: this.content

}

}, //解决子组件修改父组件传来的值而不影响其他组件

props: {

content: String //父组件传过来的值必须是String类型

},

template: '<div @click="handleClick">{{ selfContent }}</div>',

methods: {

handleClick: function() {

this.bus.$emit('change', this.selfContent) //触发change事件，并携带了 this.selfContent内容，这里不是很懂

}

},

mounted: function() { //mounted生命周期钩子，当该组件挂载时执行函数

var this\_ = this //改变this的引用

this.bus.$on('change',function(msg){ //监听bus的改变，$on监听change事件

this\_.selfContent = msg;

})

}

}

var app = new Vue({

el: '#app',

components: {

child: child

},

})