一个组件的data选项必须是一个函数，并且他总会通过return来返回一个一个可维护的独立对象

为了更加方便的使用某些常用组件，可以再main.js中对部分组件进行全局注册，但一般不使用。

组件之间的传参：

**父组件向子组件传值**

**子组件<zi></zi>**

**在子组件中设置 props{**

**pushData:’’**

**}**

**然后在父组件中向子组件写入值**

**<zi :pushData=’data1’></zi>**

**对prop进行修改**

第一种 接受prop作为data选项的初始值，然后修改data的值。

例如：

Props：{

pushData:’’

},

Data(){

Return {desc:this.pushData }

}

第二种 使用计算属性computed对prop进行转换，这种方式多用于对prop进行二次转换的情况，比如把所有的英文字母转换为小写（感觉原理一样）

Props：{

pushData:’’

},

Computed：{

lowerDesc:function()

{

Return this.pushData.toLocaleLowerCase()；

}

当传值不是一个数值时，需要在prop中设置一下

Props：{

pushData:{

type:Object,

default:function(){return{title:’这是默认的标题,则his一个默认的内容’ } }

}

}

**子组件向父组件传值**

**组件之间的事件传递**

组件之间的事件传递多是通过$emit和$on来进行的。

首先是vm.$emit(‘’,’’);他用来触发当前实例上的事件，第一个‘’是方法，第二个‘’是参数

然后是vm.$on(event,callback),他用来在当前事例中创建一个自定义事件，事件可以由vm.$emit触发，第一个‘’是事件名称，第二个‘’是当事件被触发时执行的回调方法。

**☆**

$emit(‘’,’’); 就是触发主页面上一个方法，比如close，search，或者可以自定义，意思就是

子组件设置好触发某个特定事件并传入数据，然后在主页面触发，主页面@监听到时间触发。

**插槽**

<slot name=”title”></slot>

使用时

<template v-slot:search>

**动态组件**

<component v-bind:is="componentName"></component>

示例代码，进行了两次赋值

<template>

<div>

<ul class="fy\_list\_type">

<li :class="index === activeType ? 'fy\_type\_active' : ''" v-for="item,index in fyType" :key="index" @click="typeChoose(item,index)">

{{item.name}}

</li>

</ul>

<div>

<component v-bind:is="componentName"></component>

</div>

</div>

</template>

<script>

import fyMessage from './components/fyMessage';

import kyMessage from './components/kyMessage';

export default {

name: 'seeLogger',

components:{

fyMessage,

kyMessage,

},

data(){

return{

componentName: fyMessage,

activeType:0,

fyType:[{

name:'房源',

componentName: fyMessage,

},{

name:'客源',

componentName: kyMessage,

}]

}

},

methods:{

typeChoose(it,index){

this.componentName = it.componentName;

this.activeType = index;

}

}

}

</script>

**异步组件**

为了运行的速度更快，我们需要将非首屏的页面做成异步加载

异步组件可以让你按需加载，Vue允许你以一个工厂函数的方式定义你的组件，这个异步函数会异步解析你的组件定义，Vue只有在这个组件需要渲染的地方才会触发这个工厂函数，且会把结果缓存起来，供未来重渲染。

3种定义异步组件的方式

1.标准方式

就是先定义一个组件赋值给变量，然后通过 Vue.component('async-example', function (resolve, reject){} 来创建了一个工厂函数，这个函数包含 resolve、reject 两个参数。这两个参数表示两个回调方法，我们可以通过 resolve(resCom) 使程序去异步加载定义的 resCom 组件，也可以使用 reject('加载失败描述内容'); 来表示加载失败。这里的 setTimeout 仅作为模拟异步操作使用的。

var vm = new Vue({

el: '#app',

components: {

'async-example': function (resolve, reject) {

setTimeout(function () {

resolve(resCom);

// reject('加载失败描述内容');

}, 1000);

}

}

});

2promise方式

<script>

var resCom = {

template: '#async-example-view'

};

var promise = new Promise(function (resolve, reject) {

setTimeout(function () {

resolve(resCom);

}, 1000);

});

// Vue.component('async-example', function () {

// return promise;

// });

var vm = new Vue({

el: '#app',

components: {

'async-example': function () {

return promise;

}

}

});

</script>

在上面的代码中我们声明了一个 Promise，具体的写法与标准方式相同，然后在注册这个异步组件时，直接 return 回这个 Promise 即可。

3.高级异步组件定义

<script >

var resCom = {

template: '#async-example-view'

};

var promise = new Promise(function (resolve, reject) {

setTimeout(function() {

resolve(resCom);

}, 2000);

});

var LoadingComponent = {

template: '<div>异步组件加载中...</div>'

}

var ErrorComponent = {

template: '<div>异步组件加载失败</div>'

}

const AsyncComponent = function () {

return {

// 需要加载的组件 (应该是一个 `Promise` 对象)

component: promise,

// 异步组件加载时使用的组件

loading: LoadingComponent,

// 加载失败时使用的组件

error: ErrorComponent,

// 展示加载时组件的延时时间。默认值是 200 (毫秒)

delay: 200,

// 如果提供了超时时间且组件加载也超时了，

// 则使用加载失败时使用的组件。默认值是：`Infinity`

// PS: 组件加载超时时间，超时表示加载失败，会展示ErrorComponent。

// 比如在这里当我们把 Promise 中的 setTimeout 改为 4000的时候，则会展示 ErrorComponent

timeout: 3000

}

}

var vm = new Vue({

el: '#app',

components: {

'async-example': AsyncComponent

}

});

</script>

理解：感觉就是 首先定义好3个组件，成功、失败、加载中，将成功的组件赋值给promise，成功后返回。