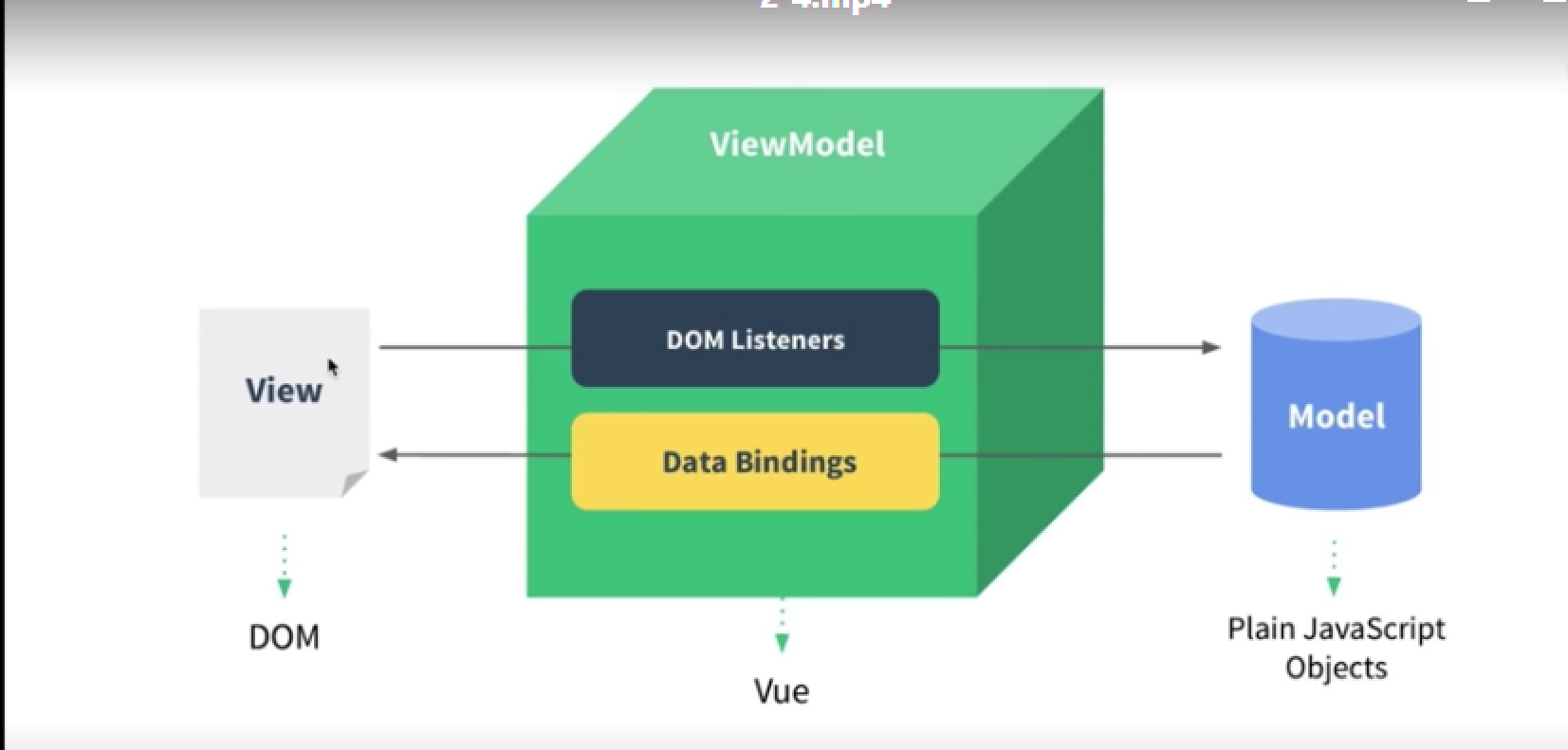
el限制vue实例接管处理的DOM的范围

data定义数据

MVP结构：M——模型层，V——视图，P——控制器（很大程度上都在做DOM的操作） => View ⬄ Presenter(核心) ⬄ Model

MVVM结构



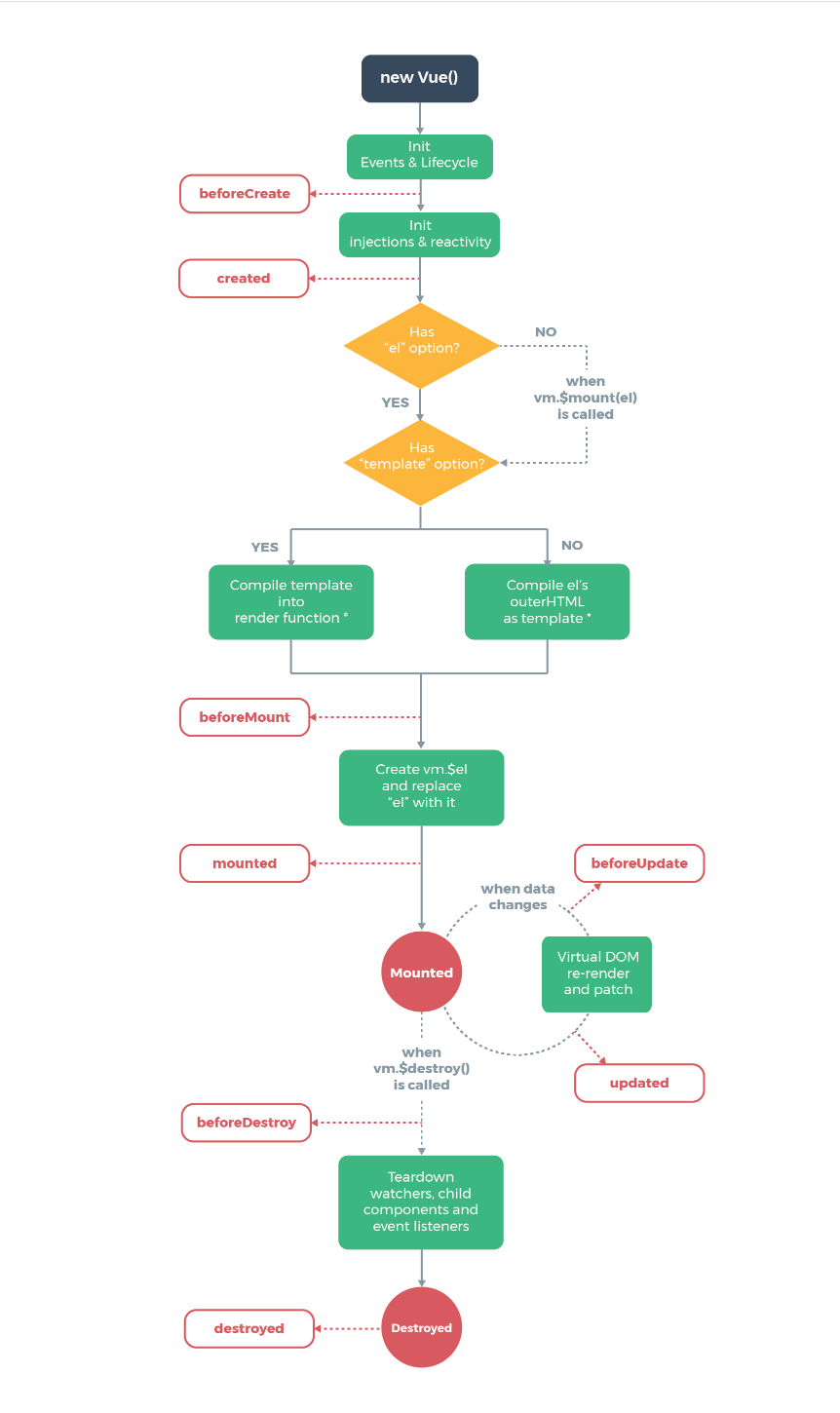
Data数据部分就是M层 页面展示部分是V层 数据变化而改变页面变化依靠是VM层(vue就是VM层)

在vue中 凡是以$符号开头的都是vue的实例属性或实例方法

VUE实例的生命周期钩子详细笔记：

生命周期函数就是vue实例zai某一个时间点会自动执行的函数//

Vue实例中的生命周期函数，不放在methods里，它是vue自带的一类函数。



当数据被改变已经重新渲染之后，update执行。

当数据被改变还未重新渲染之前，beforeUpdate执行

在页面即将要挂载时，会产生beforeMount，此时还未被渲染

页面渲染

页面挂载完成后产生mounted方法。此时已被渲染

beforeDestroy，当组件即将被销毁时执行

Destroy,当组件被销毁后执行。

当代码中没有template时，它会把el中确定的范围作为template进行渲染。

自动执行created

自动执行beforeCreate

初始化事件和生命周期相关内容

Name：“<h1>hello</h1>

v-text ：不会进行html的解析；v-text=“name”=><h1>hello</h1>

v-html：会进行html的解析；v-html=“name”=>**hello**

{{插值表达式}}：不会进行html解析{{name}}=><h1>hello</h1>

三者都可以写js表达式例如“name+‘ lee’”=>hello lee

Computed//计算属性：内置缓存（如果相关联的变量没有再发生改变，则调用只进行一次，内置缓存）【方法也可以达到计算属性的目的，但是方法无法缓存】

Watch//监听器：类似计算属性，依赖的变量发生改变才会改变，也是会有缓存值。

当一个功能既可以通过watch methods和computed实现，会优先选择computed。

计算属性中的setter和getter：

computed: {

fullName: {

get: function () {

return this.firstName + " " + this.lastName

},

set:function(value){

var arr =value.split(" ");//打散成一个数组

this.firstName=arr[0];

this.lastName=arr[1];

}

}

}

通过get和set可以改变值。

Vue绑定：

1.class的对象绑定（单击hello world 变红事件）

<body>

<div id="app">

<div @click="handleDivClick" :class="{activated:isActivated}">

<!-- 显示与否取决于isActivated -->

Hello World</div>

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

isActivated: false

},

methods: {

handleDivClick: function () {

this.isActivated = !this.isActivated;

}

}

})

</script>

</body>

</html>

<style>

.activated{

color:firebrick

}

</style>

Class与数组绑定，数组内的变量决定了，class可以显示变量的内容，（可以写多个变量）：

<body>

<div id="app">

<div @click="handleDivClick" :class="[activated]">

<!-- 显示的class是activated里的内容 -->

Hello World</div>

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

activated: ""

},

methods: {

handleDivClick: function () {

if (this.activated === "activated") {

this.activated = "";

} else {

this.activated = "activated";

}

//this.activated =this.activated === "activated"?"":"activated"

}

}

})

</script>

</body>

</html>

<style>

.activated {

color: firebrick

}

</style>

Style的方法：

<div id="app">

<div :style="styleObj" @click="handleDivClick">

<div :style=[styleObj] @click="handleDivClick">

<!—数组和直接的方法 效果相同 代码基本一致-->

Hello World</div>

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

styleObj:{

color:"black"

}

},

methods: {

handleDivClick:function(){

this.styleObj.color=this.styleObj.color==="black"?"red":"black"

}

}

})

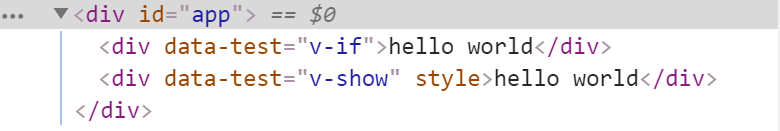
</script>

两类方法都是通过对象与数组的方式进行相关的绑定。

v-if和v-show两者都有能够在页面上进行条件渲染，v-if只要定义的值是false它就根本不存在于dom之上（会根据指令而删除），而v-show不论定义的内容是什么他都是存在于dom之上的，只是给了一个不显示的指令（display none）而已，

通过一个v-if指令通过一个返回值决定这个标签是否在页面上展示





v-if与v-else必须紧密结合的使用不能分开。

<div v-if="show==='a'">this is a</div>

<div v-else-if="show==='b'">this is b </div>

<div v-else>this is other </div>

Key值：当给某个元素加上key值时，vue就不会利用以前的这个标签，所以就不会将之前的内容连带标签一起使用过来。（加上key值后vue可以区分每个标签）

<div v-if="show" key="username">用户名：<input></div>

<div v-else key="password">邮箱：<input></div>

列表渲染：

<div v-for="(item,index) of list":key="item.id" >{{item.text}}---{{index}}</div>

<!-- 不推荐index作为key值，在频繁操作DOM相对应元素时比较费性能，可能会没办法复用DOM结点 -->

<!-- 使用key值唯一但不要使用index才能使key值性能发挥最好 -->

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

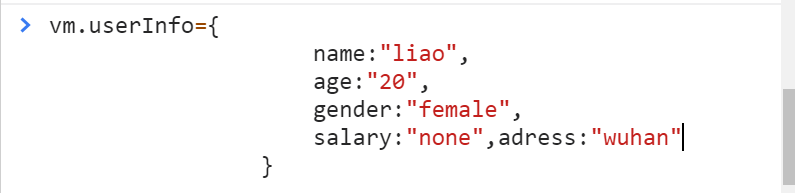
list: [{

id:"010120201",

text:"hello"

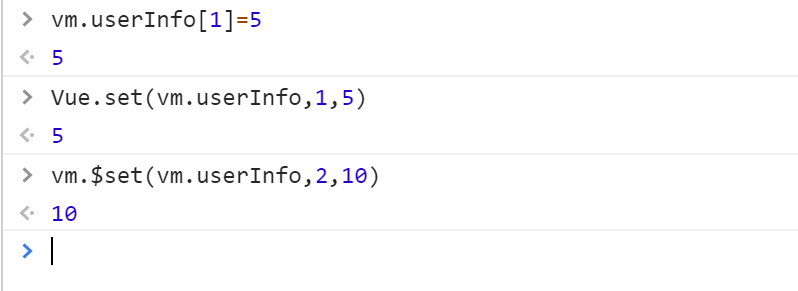
},

当我们向数组里添加数据的时候，不能只通过往数组里添加下标的方式操作数组，只能通过vue提供的七个编译数组的方法才能够实现改变数组渲染页面的效果：

1. pop-删除2.push-添加3.shift-数组第一项删除4. Unshift-数组的第一项加内容 5.splice-数组截取 6sor-数组排序 7.reverse-数组取反
2. 直接将代码粘贴，改动里头的内容，也可达到直接渲染的目的。（改变数组的引用）

template占位符，可以包裹元素并且不被渲染到页面上。

Vue中的set方法：



直接改动userInfo里头的下标不能将之渲染到页面上。

Vue.set(vm.userInfo,1,5) ：利用set方法将userinfo里的下标1的数字2改为数字5.

Vm.$set(vm.userInfo,2,10) ：利用实例set方法将下标为2的数字3改为数字10.。

实现数组改变页面随之变动的方法有以上三种：vue编译数组七种方法，改变数组的引用，利用vue的set方法。

组件使用中的细节：

使用is标签解决模板上的bug问题：

<table>

<tbody>

<tr is="row"></tr>

<tr is="row"></tr>

<tr is="row"></tr>

<!-- is代表此处tr内容为row组件，既能保证tr里是row组件，又能满足h5的规范 -->

</tbody>

</table>

在h5中规定table中有tbody，tbody中有tr，所以为了使组件row能够显示并不出现bug利用is即可解决问题。

在非根组件中不能直接写data:{}(如下)

Vue.component('row',{

data:{

content:'this is a row'

},

template:'<tr><td>{{content}}</td></tr>'

})

应改为

Vue.component('row', {

data: function(){

return {

content: 'this is a row'

}

},

template: '<tr><td>{{content}}</td></tr>'

})

子组件定义data必须是函数而不能是对象，因为子组件不像根组件只调用一次，每个子组件应该有自己的数据而不应该与其他的子组件的数据互相影响。

Ref引用：

<div id="app">

<counter ref="one" @change="handleChange"></counter>

<counter ref="two" @change="handleChange"></counter>

<div>{{total}}</div>

</div>

<script>

Vue.component('counter', {

data: function () {

return {

number: 0

}

},

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

methods: {

handleClick: function () {

this.number++

this.$emit('change')

}

}

})

var vm = new Vue({

el: "#app",

data: {

total:0

},

methods:{

handleChange:function(){

this.total=this.$refs.one.number+this.$refs.two.number

}

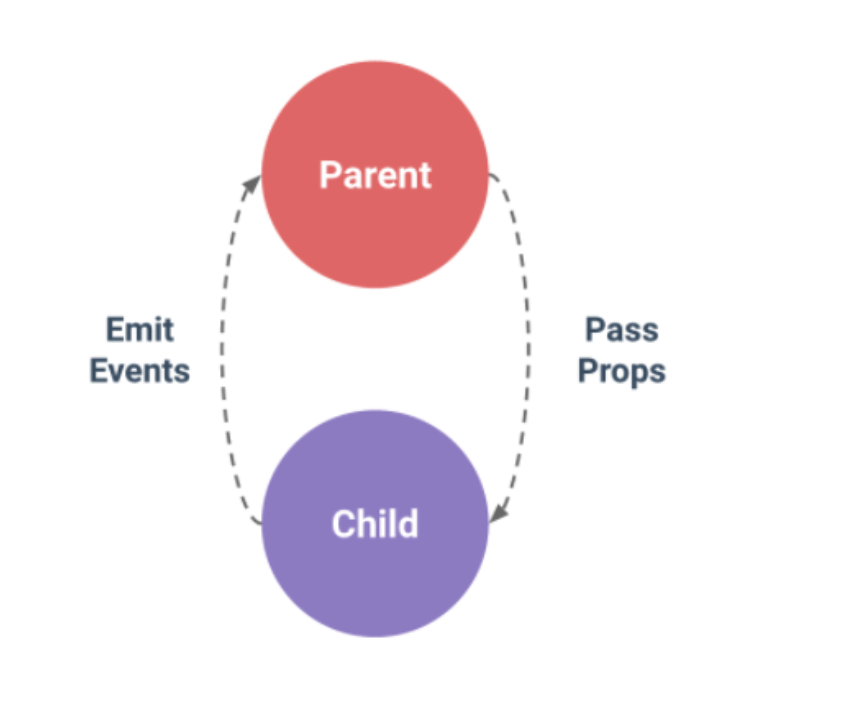
}

})

</script>

父子组件的传值内容：

单项数据流：父组件可以随意向子组件传递参数，但是子组件不可以随意修改父组件传递过来的参数。



<div id="app">

<counter :count="3" @change="handleChange"></counter>

<counter :count="2" @change="handleChange"></counter>

<!-- 加了：代表count穿的值为数字，不加：则传的值为字符 -->

<div>{{total}}</div>

</div>

<script>

var counter = {

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

props: ['count'],

data: function () {

return {

number: this.count

}//拷贝一个副本，修改副本里的值而不是直接修改父组件里的值

},

methods: {

handleClick: function () {

this.number=this.number+2;

this.$emit('change',2)//2即为step

}

}

}

var vm = new Vue({

el: "#app",

components: {

counter: counter

},

data: {

total: 5

},

methods: {

handleChange:function(step){

this.total+=step

}

}

})

</script>

组件参数校验与非props特性：

参数校验即为，当父组件传递参数给子组件，子组件可以加一些约束条件给参数。

Vue.component('child',{

template:'<div>{{content}}</div>',

props:{

content:[Number,String]//接受数字和字符

}

})

props:{

content:{

type:String

}

}

content:{

type:String,

required:true//必须传值

}

content:{

type:String,

required:false,

//必须传值

default:'default value'//当没有传值时，会显示defau里头的内容

validator: function (value) {

return (value.length > 5)//要求传入值的长度必须大于5

}

}

Props特性：当父组件向子组件传值时，父子组件有对应关系，即父组件中声明了content，子组件中的props也声明了content，这时就可以在子组件中直接用插值表达式直接使用。

非props特性：父组件向子组件传递属性，子组件并没有props内容，即没有声明父组件传的值的内容

非props特性会显示到dom上，即content=“hell“会显示到dom中

给组件绑定原生事件：

在给组件绑定一个@click=“handleClick”事件时，这个事件并非原生事件而是一个自定义事件，给div这类型元素绑定@click事件时才属于原生事件。如果要直接绑定这个事件只需要

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

非父子组件之间的传值问题：

（观察者模式）发布订阅模式——总线机制 （bus）

组件单项数据流，强制改content导致出错

<body>

<div id="app">

<child content="dell"></child>

<child content="lee"></child>

</div>

<script>

Vue.prototype.bus=new Vue()

//每一个vue实例上都会挂载一个bus属性

Vue.component('child', {

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

props:{

content:String

},

data:function(){

return{

selfContent:this.content

}

},

methods:{

handleClick:function(){

this.bus.$emit('change',this.selfContent)

//bus指的是每个实例上都有bus属性，bus又是一个实例所以有$emit方法

}

},

mounted:function(){

var this\_=this

//监听bus的change事件

this.bus.$on('change',function(msg){

this\_.selfContent =msg

// alert(msg) //弹出两次弹窗，每个child都会执行change

})

}

})

var vm = new Vue({

el: "#app",

data: {

total: 5

},

methods: {

handleClick: function () {

alert('click')

}

}

})

</script>

</body>

在vue中使用插槽：

倘若不是具名插槽，当出现<div class='header' slot='header'>header</div>

<div class='footer' slot='footer'>footer</div>

则<slot ></slot>

<div class='child'>child</div>

<slot ></slot>

两个slot插槽会都显示两个div，但是如果使用具名插槽则不会出现这种情况：

<slot name='header'></slot>

<div class='child'>child</div>

<slot name='footer'></slot>

作用域插槽：

父组件调用子组件的时候给子组件传了一个插槽，叫做作用域插槽，必须是一个template开头结尾的内容，插槽要声明从作用域接收到的信息都放在一个叫做props属性里，然后还要给出一个模板：

<child >

<template slot-scope="props">

<!-- 作用是当子组件用slot时会向slot中传一个item，用child时就可以用item的数据 -->

<h1>{{props.item}}</h1>

</template>

</child>

Vue.component('child', {

template: `<div>

<ul>

<slot v-for="item of list" :item=item>{{item}}</slot>

</ul>

</div>`,

data:function(){

return{

list:[1,2,3,4]

}

}

})

动态组件和v-once：

动态组件：

<div id="app">

<component :is="show"></component>

<!-- <child-one v-if="show==='child-one'">

</child-one>

<child-two v-if="show==='child-two'">

</child-two>-->

<button @click="handleClick">change</button>

</div>

<script>

Vue.prototype.bus=new Vue()

//每一个vue实例上都会挂载一个bus属性

Vue.component('child-one', {

template: `<div>

child-one

</div>`,

data:function(){

return{

list:[1,2,3,4]

}

}

})

Vue.component('child-two',{

template:`<div>child-two</div>`

})

var vm = new Vue({

el: "#app",

data: {

show:'child-one'

},

methods: {

handleClick:function(){

this.show=this.show==='child-one'?'child-two':'child-one'

}

}

})

</script>

</body>

v-once:

Vue.component('child-one', {

template: `<div v-once>//v-once可以有效提高静态内容的展示效率

child-one

</div>`,

data:function(){

return{

list:[1,2,3,4]

}

}

})

Vue.component('child-two',{

template:`<div v-once>child-two</div>`

})

动画效果：

Transition的相关：当一个元素被transition包裹后，vue会自动的分析这个元素的css样式并且构建一个动画的流程，在动画即将被执行的时候会出现两个class名：enter enter-active.

Css过渡动画：

<style>

.activated {

color: firebrick

}

.fade-enter{

opacity: 0

}

.fade-enter-active{

transition: opacity 3s;

}

.fade-leave-to{

opacity: 0;

}

/\* .fade-leave{

opacity: 0;

} \*/

.fade-leave-active{

transition: opacity 3s;

}

</style>

Keyframe放大缩小字体动画效果：

<style>

@keyframes bounce-in{

0%{

transform: scale(0);

}

50%{

transform: scale(0.5);

}

100%{

transform: scale(1);

}

}

.fade-enter-active{

transform-origin: left center;

animation:bounce-in 1s;

}

.fade-leave-active{

transform-origin: left center;

animation:bounce-in 1s reverse;

}

Animate Css:

自定义class的方式，必须包括animate，要将相关动画名字加上。

过渡动画与动画特效同时使用：

<transition name="fade"

type="transition"//手动设置按照transition动画的时长来作为动画时长

:duration="{enter:10000,leave:5000}"//设置动画进出场时长

appear//让刷新的动画效果呈现要加这一步

enter-active-class="animated hinge fade-enter-active"//可以不仅用animated的动画还可以用transition自身的动画

leave-active-class="animated tada fade-leave-active"

appear-active-class="animated tada"//设置刷新页面的动态效果>