1.其他

（1）this.$refs.xxx得到的是vue组件，this.$refs.xxx.$el得到的是vue组件下的子元素。

<search-box ref="searchBox"></search-box>

（2）父组件调用子组件方法；this.$refs.xxx.func();（也算一种父向子通信的方法）

子组件：

setQuery(query){  
 this.**query**=query;  
}

父组件：

selectItem(k){  
 this.$refs.searchBox.setQuery(k);  
}

1. 子组件向父组件通信

子组件：

selectItem(item){  
 // 派发事件，向父组件通信，select是事件名，item是参数  
 this.$emit('select',item);  
},

父组件：

<!--接收子组件派发出来的名为select事件，父组件处理函数时goToSingerDetail-->  
<list-view @select="goToSingerDetail" ></list-view>

goToSingerDetail(item){  
 this.**setSinger**(item);  
 this.$router.push({  
 **path**:'/singer/'+item.**id** })  
},

2.clientHeight和offsetHeight

(1)clientHeight网页(内容)可见区域高：document.body.clientHeight 即页面浏览器中可以看到内容的这个区域的高度，一般是最后一个工具条以下到状态栏以上的这个区域，与页面内容无关。注意：IE、Opera 认为 scrollHeight 是网页内容实际高度，可以小于 clientHeight。(2)offsetHeight网页可见区域高：document.body.offsetHeight (包括边线的宽)

3.better-scroll

(1).better-scroll只对容器下的第一个子节点进行滚动。

(2).Better-scroll横向滚动时要有宽度，否则不会滚动，纵向滚动时也要有高度

(3).DOM要生成了才能初始化better-scroll，否则会报错。

(4).由于fastclick和better-scroll（click属性）会发生冲突，导致点击后不能跳转，所以我们需要在img的class中添加needsclick，这样当我们点击图片时，点击事件就不会被拦截。

(5).要滚动的内容高度或者宽度必须要比容器的高度或者宽度大，否者无法滚动。

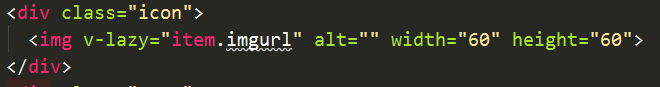
(6).better-scroll在y轴滚动的距离，不动时滚动距离为0，在顶端往下拉时滚动距离为正数，其他情况都为负数。

(7).better-scroll里面滚动的元素发生变化时要调用refresh重新计算高度，要等DOM全部渲染完成后才计算

4.图片懒加载（vue-lazyload）

用法：





官网例子



5.进入/离开 & 列表过渡

(a)这里是一个典型的例子：

<div id="demo">

<button v-on:click="show = !show">

Toggle

</button>

<transition name="fade">

<p v-if="show">hello</p>

</transition>

</div>

new Vue({

el: '#demo',

data: {

show: true

}

})

.fade-enter-active, .fade-leave-active {

transition: opacity .5s;

}

.fade-enter, .fade-leave-to /\* .fade-leave-active below version 2.1.8 \*/ {

opacity: 0;

}

(b)过渡的类名

在进入/离开的过渡中，会有 6 个 class 切换。

v-enter：定义进入过渡的开始状态。在元素被插入时生效，在下一个帧移除。

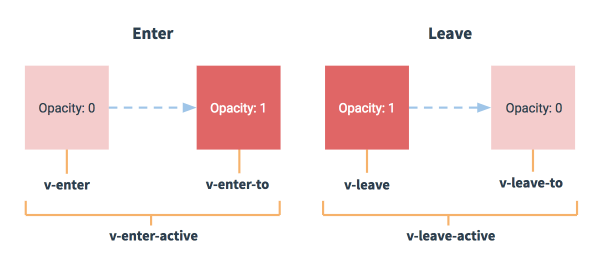
v-enter-active：定义过渡的状态。在元素整个过渡过程中作用，在元素被插入时生效，在 transition/animation 完成之后移除。这个类可以被用来定义过渡的过程时间，延迟和曲线函数。

v-enter-to: 2.1.8版及以上 定义进入过渡的结束状态。在元素被插入一帧后生效 (与此同时 v-enter 被删除)，在 transition/animation 完成之后移除。

v-leave: 定义离开过渡的开始状态。在离开过渡被触发时生效，在下一个帧移除。

v-leave-active：定义过渡的状态。在元素整个过渡过程中作用，在离开过渡被触发后立即生效，在 transition/animation 完成之后移除。这个类可以被用来定义过渡的过程时间，延迟和曲线函数。

v-leave-to: 2.1.8版及以上 定义离开过渡的结束状态。在离开过渡被触发一帧后生效 (与此同时 v-leave 被删除)，在 transition/animation 完成之后移除。



(c)JavaScript 钩子

可以在属性中声明 JavaScript 钩子

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

v-on:enter-cancelled="enterCancelled"

v-on:before-leave="beforeLeave"

v-on:leave="leave"

v-on:after-leave="afterLeave"

v-on:leave-cancelled="leaveCancelled"

>

<!-- ... -->

</transition>

// ...

methods: {

// --------

// 进入中

// --------

beforeEnter: function (el) {

// ...

},

// 此回调函数是可选项的设置

// 与 CSS 结合时使用

enter: function (el, done) {

// ...

done()

},

afterEnter: function (el) {

// ...

},

enterCancelled: function (el) {

// ...

},

// --------

// 离开时

// --------

beforeLeave: function (el) {

// ...

},

// 此回调函数是可选项的设置

// 与 CSS 结合时使用

leave: function (el, done) {

// ...

done()

},

afterLeave: function (el) {

// ...

},

// leaveCancelled 只用于 v-show 中

leaveCancelled: function (el) {

// ...

}

}

这些钩子函数可以结合 CSS transitions/animations 使用，也可以单独使用。

当只用 JavaScript 过渡的时候， 在 enter 和 leave 中，回调函数 done 是必须的 。否则，它们会被同步调用，过渡会立即完成。

推荐对于仅使用 JavaScript 过渡的元素添加 v-bind:css="false"，Vue 会跳过 CSS 的检测。这也可以避免过渡过程中 CSS 的影响。

<transition name="normal"  
 @enter="enter"  
 @after-enter="afterEnter"  
 @leave="leave"  
 @after-leave="afterLeave"  
>

// el是DOM元素，done是回调函数，执行done后会自动执行钩子函数@after-enter="afterEnter"，done必须要执行，不执行，动画会没有效果  
enter(el,done){  
 var {x,y,scale}=this.\_getPosAndScale();  
 // 下面是利用create-keyframe-animation用js写css动画  
 // 动画  
 let animation={  
 0:{  
 **transform**: `translate3d(${x}px,${y}px,0) scale(${scale})`  
 },  
 60 :{  
 **transform**: `translate3d(0,0,0) scale(1.1)`  
 },  
 100:{  
 **transform**: `translate3d(0,0,0) scale(1)`  
 }  
 }  
 // 注册动画  
 animations.registerAnimation({  
 **name**:'move', //动画名称  
 animation, //动画  
 **parset**:{ //执行动画的一些参数  
 **duration**: 400, //执行动画的时间  
 **easing**: 'linear' //运动方式  
 }  
 })  
 // 运行动画，第一个参数是执行动画的DOM元素，第二个是注册了的动画名称，第三个参数是回调函数  
 animations.**runAnimation**(this.$refs.cdWrapper,'move',done);  
},  
afterEnter(){  
 // 取消注册了的动画  
 animations.unregisterAnimation(this.$refs.cdWrapper,'move');  
 this.$refs.cdWrapper.**style**.**animation**='';  
},  
// el是DOM元素，done是回调函数，执行done后会自动执行钩子函数@after-leave="afterLeave"，done必须要执行，不执行，动画会没有效果  
leave(el,done){  
 var {x,y,scale}=this.\_getPosAndScale();  
 // 设置离开时的过渡效果  
 this.$refs.cdWrapper.**style**.**transition**='all 0.4s'  
 // 全屏播放器图片移动到迷你播放器图片的位置  
 this.$refs.cdWrapper.**style**[transform]=`translate3d(${x}px,${y}px,0) scale(${scale})`;  
 // 过渡效果执行完毕后执行done  
 this.$refs.cdWrapper.addEventListener('transitionend',done)  
},  
afterLeave(){  
 // 清除过渡效果和动画  
 this.$refs.cdWrapper.**style**.**transition**='';  
 this.$refs.cdWrapper.**style**[transform]='';  
},  
// 播放器进行动画所需要的信息  
\_getPosAndScale(){  
 // 迷你播放器图片宽度  
 var targerWidth=40;  
 // 迷你播放器图片圆心距离屏幕左边的距离  
 var paddingLeft=40;  
 // 迷你播放器图片圆心距离屏幕下边的距离  
 var paddingBottom=30;  
 //全屏播放器图片(非圆心)距离屏幕上边的距离  
 var paddingTop=80;  
 // 全屏播放器图片宽度  
 var width=window.innerWidth\*0.8;  
 //全屏图片变成迷你图片要进行缩放的比例  
 var scale=targerWidth/width;  
 // 全屏图片移动到迷你图片的x轴位移,x轴正方向向右,y轴正方向向下,原点和全屏图片重合  
 var x= -(window.innerWidth/2-paddingLeft);  
 // 全屏图片移动到迷你图片的x轴位移  
 var y= window.innerHeight-paddingBottom-paddingTop-width/2;  
 return{  
 x,y,scale  
 }  
}

(d)列表过渡

目前为止，关于过渡我们已经讲到：

单个节点

同一时间渲染多个节点中的一个

那么怎么同时渲染整个列表，比如使用 v-for ？在这种场景中，使用 <transition-group> 组件。在我们深入例子之前，先了解关于这个组件的几个特点：

不同于 <transition>，它会以一个真实元素呈现：默认为一个 <span>。你也可以通过 tag 特性更换为其他元素。

内部元素 总是需要 提供唯一的 key 属性值

列表的进入/离开过渡

现在让我们由一个简单的例子深入，进入和离开的过渡使用之前一样的 CSS 类名。

<div id="list-demo" class="demo">

<button v-on:click="add">Add</button>

<button v-on:click="remove">Remove</button>

<transition-group name="list" tag="p">

<span v-for="item in items" v-bind:key="item" class="list-item">

{{ item }}

</span>

</transition-group>

</div>

new Vue({

el: '#list-demo',

data: {

items: [1,2,3,4,5,6,7,8,9],

nextNum: 10

},

methods: {

randomIndex: function () {

return Math.floor(Math.random() \* this.items.length)

},

add: function () {

this.items.splice(this.randomIndex(), 0, this.nextNum++)

},

remove: function () {

this.items.splice(this.randomIndex(), 1)

},

}

})

.list-item {

display: inline-block;

margin-right: 10px;

}

.list-enter-active, .list-leave-active {

transition: all 1s;

}

.list-enter, .list-leave-to

/\* .list-leave-active for below version 2.1.8 \*/ {

opacity: 0;

transform: translateY(30px);

}

<transition-group name="list" tag="ul">  
 <!--添加transition-group后必须为每一个li添加一个key进行唯一标识-->  
 <li :key="index" ref="listItem" class="item" v-for="(item,index) in **sequenceList**" @click.stop="selectItem(item,index)">  
 </li>  
</transition-group>

&.list-enter-active, &.list-leave-active  
 transition: all 0.1s  
&.list-enter, &.list-leave-to  
 height: 0

(e)过渡模式

同时生效的进入和离开的过渡不能满足所有要求，所以 Vue 提供了 过渡模式

in-out：新元素先进行过渡，完成之后当前元素过渡离开。

out-in：当前元素先进行过渡，完成之后新元素过渡进入。

用 out-in 重写之前的开关按钮过渡：

<transition name="fade" mode="out-in">

<!-- ... the buttons ... -->

</transition>

off

只用添加一个简单的特性，就解决了之前的过渡问题而无需任何额外的代码。

in-out 模式不是经常用到，但对于一些稍微不同的过渡效果还是有用的。

6.Vuex

(1)对象state

// 仓库  
const state={  
 **singer**:{}  
}  
export default state;

(2)actions

// 异步操作或者是要执行多个mutations  
import \* as types from './mutations-type.js';  
  
// {commit,state}是vuex自动传进来的，commit用来调用mutations中的方法的，{list,index}是我们传递过来的参数  
export const setPlay=function({commit,state},{list,index}){  
 commit(types.SET\_PLAYING,true);  
 commit(types.SET\_PLAYLIST,list);  
 commit(types.SET\_FULLSCREEN,true);  
 commit(types.SET\_CURRENTINDEX,index);  
 commit(types.SET\_SEQUENCELIST,list);  
}

(3)getters

export const singer= state => state.**singer**;

(4)mutations

// 字符串常量  
export const SET\_SINGER='SET\_SINGER';

import \* as types from './mutations-type.js';  
// 修改state  
const mutations={  
 // 函数的一种写法,第一个参数是仓库state，第二个参数是我们传递进来的值  
 [types.SET\_SINGER](state,singer){  
 state.**singer**=singer;  
 }  
}  
export default mutations;

(5)Index

import Vue from 'vue';  
import Vuex from 'vuex';  
import \* as actions from './actions.js';  
import \* as getters from './getters.js';  
import state from './state.js';  
import mutations from './mutations.js';  
// 调试用  
import createLogger from 'vuex/dist/logger'  
Vue.use(Vuex)  
// 调试用  
const debug = process.**env**.**NODE\_ENV** !== 'production'  
export default new Vuex.Store({  
 actions,  
 getters,  
 state,  
 mutations,  
 **strict**: debug,  
 **plugins**: debug ? [createLogger()] : []  
});

(6)组件内使用(统一用this调用)

// 导出修改store的行为方法，必须在methods中导出，即可在组件内使用，this.setSinger(xxx),不用该方法导出相当于this.$store.commit('SET\_SINGER');  
...mapMutations({  
 **setSinger**:'SET\_SINGER'  
})

// 导出获取仓库state的方法,名称（字符串）要与与getters中的函数名一致，必须在computed中导出，即可在组件内使用，this.singer,不用该方法导出相当于return this.$store.getters.singer或者this.$store.state.singer;  
...mapGetters([  
 'singer'  
])

// 导出actions里面的方法，必须在methods中导出，参数是数组，名称要与actions中的方法名一样  
...mapActions([  
 'setPlay'  
])

7.混入

混入 (mixins) 是一种分发 Vue 组件中可复用功能的非常灵活的方式。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

例子：

// 定义一个混入对象

var myMixin = {

created: function () {

this.hello()

},

methods: {

hello: function () {

console.log('hello from mixin!')

}

}

}

// 定义一个使用混入对象的组件

var Component = Vue.extend({

mixins: [myMixin]

})

var component = new Component() // => "hello from mixin!"

选项合并

当组件和混入对象含有同名选项时，这些选项将以恰当的方式混合。

比如，数据对象在内部会进行浅合并 (一层属性深度)，在和组件的数据发生冲突时以组件数据优先。

var mixin = {

data: function () {

return {

message: 'hello',

foo: 'abc'

}

}

}

new Vue({

mixins: [mixin],

data: function () {

return {

message: 'goodbye',

bar: 'def'

}

},

created: function () {

console.log(this.$data)

// => { message: "goodbye", foo: "abc", bar: "def" }

}

})

同名钩子函数将混合为一个数组，因此都将被调用。另外，混入对象的钩子将在组件自身钩子之前调用。

var mixin = {

created: function () {

console.log('混入对象的钩子被调用')

}

}

new Vue({

mixins: [mixin],

created: function () {

console.log('组件钩子被调用')

}

})

// => "混入对象的钩子被调用"

// => "组件钩子被调用"

值为对象的选项，例如 methods, components 和 directives，将被混合为同一个对象。两个对象键名冲突时，取组件对象的键值对。

var mixin = {

methods: {

foo: function () {

console.log('foo')

},

conflicting: function () {

console.log('from mixin')

}

}

}

var vm = new Vue({

mixins: [mixin],

methods: {

bar: function () {

console.log('bar')

},

conflicting: function () {

console.log('from self')

}

}

})

vm.foo() // => "foo"

vm.bar() // => "bar"

vm.conflicting() // => "from self"

注意：Vue.extend() 也使用同样的策略进行合并。

全局混入

也可以全局注册混入对象。注意使用！ 一旦使用全局混入对象，将会影响到 所有 之后创建的 Vue 实例。使用恰当时，可以为自定义对象注入处理逻辑。

// 为自定义的选项 'myOption' 注入一个处理器。

Vue.mixin({

created: function () {

var myOption = this.$options.myOption

if (myOption) {

console.log(myOption)

}

}

})

new Vue({

myOption: 'hello!'

})

// => "hello!"

谨慎使用全局混入对象，因为会影响到每个单独创建的 Vue 实例 (包括第三方模板)。大多数情况下，只应当应用于自定义选项，就像上面示例一样。也可以将其用作 Plugins 以避免产生重复应用

import {mapGetters} from 'vuex';  
export const playlistMixin ={  
 **computed**: {  
 ...mapGetters([  
 'playList'  
 ])  
 },  
 mounted() {  
 this.handlePlaylist(this.**playList**)  
 },  
 // 组件激活的时候  
 activated() {  
 this.handlePlaylist(this.**playList**)  
 },  
 **watch**: {  
 playList(newVal) {  
 this.handlePlaylist(newVal)  
 }  
 },  
 **methods**: {  
 handlePlaylist() {  
 // 这里定义用户必须在具体组件内实现该方法，不然会报错  
 throw new Error('component must implement handlePlaylist method')  
 }  
 }  
}

组件内使用

import {playlistMixin} from "../../common/js/mixin.js";

**mixins**:[playlistMixin],

// 实现mixins中的handlePlaylist方法  
handlePlaylist(list){  
 if(list.**length**>0){  
 this.$refs.**singer**.**style**.**bottom**='60px';  
 this.$refs.**list**.refresh();  
 }  
},

8.slice()方法和splice()方法

slice()，返回一个新的数组，该方法可从已有的数组中返回选定的元素。例如：arrObject(start，end)，start是必选的。规定从何处开始选取，如果是负数，则是从数组元素尾部选取，也就是说-1指最后一个元素，-2指倒数第二个元素；end是可选元素。规定从何处结束选取。该参数没有，表示从开始位置截取到数组末尾，如果是负数，表示从数组末尾向前截取元素。该方法不会修改原数组，如果想删除数组中某一段元素，使用splice()方法。

splice()，数组中添加/删除元素。例如：arrayObject.splice(index,howmany,item1,.....,itemX)。index是必选的，规定添加/删除项目的位置。howmany必须项，表示删除项目的个数，如果为0表示不删除元素。splice()可删除从index开始的howmany个元素，并可以用item元素替代删除的元素。item可选参数，表示新添加的项。

用法如：

(1)

var arr = new Array(5);

arr[0] = "amy";

arr[1] = "elice";

arr[2] = "divi";

arr[3] = "lvy";

arr[4] = "marry";

arr.splice(1, 0, "willian");

console.log(arr);

//输出：amy，willian,elice,divi,lvy，在数组第一个位置增加一个元素，其值是"willian"

(2)

var arr = new Array(5);

arr[0] = "amy";

arr[1] = "elice";

arr[2] = "divi";

arr[3] = "lvy";

arr[4] = "marry";

arr.splice(1, 2, "willian");

console.log(arr);

//输出：amy，willian,lvy，从数组第一个位置起删除两个元素，并用新的元素“willian”代替删除的元素。

1. 函数节流

算是固定写法

// 函数节流，即延迟一段时间在执行要执行的函数，func是要执行的函数，即要被延迟执行的函数，delay是要延迟多久  
export function debounce(func,delay) {  
 var timer;  
 return function (...args) {  
 if(timer){  
 clearTimeout(timer)  
 }  
 timer=setTimeout(()=>{  
 // apply()方法 接收两个参数，一个是函数运行的作用域（this），另一个是参数数组。  
 func.apply(this,args)  
 },delay)  
 }  
}

// 监听query的变化,这里不写在watch中是因为要进行函数节流，这样可以让用户在输入框中输入后延迟一段时间在发送请求，使得不用频繁发送请求  
this.$watch('query',debounce((newQuery)=>{  
 this.$emit('query',newQuery)  
},400))

10.数组

(a) filter() 方法

返回数组 ages 中所有元素都大于 18 的元素:

var ages = [32, 33, 16, 40];

function checkAdult(age) {

return age >= 18;

}

function myFunction() {

document.getElementById("demo").innerHTML = ages.filter(checkAdult);

}

输出结果为:

32,33,40

定义和用法

filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

注意： filter() 不会对空数组进行检测。

注意： filter() 不会改变原始数组。

(b)find() 方法

获取数组中年龄大于 18 的第一个元素

var ages = [3, 10, 18, 20];

function checkAdult(age) {

return age >= 18;

}

function myFunction() {

document.getElementById("demo").innerHTML = ages.find(checkAdult);

}

fruits 输出结果：

18

定义和用法

find() 方法返回传入一个测试条件（函数）符合条件的数组第一个元素。

find() 方法为数组中的每个元素都调用一次函数执行：

当数组中的元素在测试条件时返回 true 时, find() 返回符合条件的元素，之后的值不会再调用执行函数。

如果没有符合条件的元素返回 undefined

注意: find() 对于空数组，函数是不会执行的。

注意: find() 并没有改变数组的原始值。

11.路由懒加载

（1）第一种写法

const Singer=(resolve)=>{  
 import('../components/singer/singer.vue').then((singer) =>{  
 resolve(singer)  
 })  
}

1. 第二种写法

const home =(r)=>require(['./component/home/home.vue'],r);