1. **运用的技术**

vue全家桶(vuex, vue-router, vue-cli) + axios +jsonp + scss +vue-lazy(图片懒加载)

1. **不要在main.js中引入variable.scss, mixin.scss等变量文件(在.vue文件中使用无效)，因为main.js和vue文件是分开处理scss的，vue-loader单独处理.vue中的scss，需要全局导入在vue文件中，例如：**

css: {

loaderOptions: {

sass: {

prependData: `

@import '@/common/scss/variable.scss';

@import '@/common/scss/mixin.scss';

`

}

}

},

1. **vue.config.js中使用代理解决前端请求**

有两种方式：

**(1). proxy代理**

proxy: {

'/api/getPicList': {

// 代理的是/api/getPicList中的 /

target: 'https://c.y.qq.com/musichall/fcgi-bin/fcg\_yqqhomepagerecommend.fcg',

// 跨域需要，因为有些网站防止爬虫会对origin作出限制，

// 为了能爬到数据，必须加这个

// 把本机的host改为target地址

changeOrigin: true,

// 必须覆盖为空, 因为真实接口上面没有api/getPicList

pathRewrite: {

'^/api/getPicList': ''

}

}

}

注意：

**不需要在proxy里面添加url参数，前端部分会自动添加的，**例如：

/api/getPicList?name=xxx&age=33 =>

'https://c.y.qq.com/musichall/fcgi-bin/fcg\_yqqhome

pagerecomme?name=xxx&age=33

如果不pathRewrite的话，那么会是这样

'https://c.y.qq.com/musichall/fcgi-bin/fcg\_yqqhome

pagerecomme/api/getPicList?name=xxx&age=33

真实接口错误，没有/api/getPicList这个路径，所以必须要设置pathRewrite

**(2). before中间件(推荐)**

before: app => {

app.get('/getSingerSongs', (req, res) => {

const url = 'https://u.y.qq.com/cgi-bin/musicu.fcg';

axios.get(url, {

headers: {

'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8;',

},

// 前端部分url上的问号查询参数

params: req.query,

}).then((response) => {

// 注意不能res.json(response), 只能是response.data

res.json(response.data);

}).catch(e => console.log(e));

});

},

// 太多路由请求时候，放在外部函数，更方便

before: app => {

// 在外部函数中集中处理

handleRouters(app, axios, express.Router());

},

1. **flex布局**

**纵向布局：**

最后一个元素flex: 1; 要加overflow:hidden，配合scroll滚动

**横向布局：**

一般左图 固定值，右侧文案 flex: 1;

**横向滚动**时候， 可以在ul元素上用 **display: inline-flex:** 来代替

position:absolute + display:inline-block + no-wrap, 父元素div用

overflow:hidden

1. **对于父组件向子组件传递数据，如果是动态属性，自定义属性前要有”:” ,如果是固定值，就不需要在前面加”:”, 例如：**

<music-list :data=”mvList” title=”第一次”>

**动态属性的值都是变量，静态属性的值都是写死的值。**

1. **Fast Click 是一个简单、易用的库，专为消除移动端浏览器从物理触摸到触发点击事件之间的300ms延时 为什么会存在延迟呢？**

从触摸按钮到触发点击事件，移动端浏览器会等待接近300ms，原因是浏览器会等待以确定你是否执行双击事件

**何时不需要？**

Android 系统中，在头部 meta 中设置 width=device-width 的Chrome32+ 浏览器不存在300ms 延时，所以，也不需要

1. **数据为什么有时候放在data()里，有时又放在created里面那？**

放在data里面的数据会被watch, 即添加了 **getter 和 setter,** 放在**created**里面不会被watch。

例如：

移动端触屏事件, 用 this.touch = {} 里面放触屏时候的x坐标，y坐标, 不需要watch，

所以放在created钩子函数内

const touches = e.touches[0]; // 第一个手指

this.touch.startX = touches.pageX;

1. **对于vue组件, this.$refs.xxx拿到的是组件，可以调用组件内的方法，如果想拿到dom元**

**素，必须this.$refs.xxx.$el.style.xxx**

例如：

<song-list ref=”songList”></song-list>

this.$refs.songList.show(); // 调用song-list组件的show方法

this.$refs.songList.$el.style.width = ‘50px’; // 设置song-list组件的dom元素的宽度

1. **v-html可以转义字符，处理特定接口有用。**

比如数据中有&lt; 或者 &#60, 在template模板中，可以用v-html转义成 > ，

在js中可以用以下方法处理：

(1). 用正则 + fromCharCode解码

function fromCharCode(str) {

// fromCharCode里的参数是 数字

return str.replace(/&#(\d+);/g,

(match, submatch) => String.fromCharCode(submatch));

}

(2). innerHTML也能对其解码

function fromCharCode(str) {

const div = document.createElement('div');

div.innerHTML = str;

return div.textContent || div.innerText;

}

const str = '&#60;我的&#62;';

fromCharCode(str); // '<我的>'

1. **基础组件一般都要被复用，所以里面的数据都不是在自身组件内请求的，而是通过props接收来自父组件的数据。**
2. **组件的显隐控制**

点击时候通过父组件调用**子组件的内部方法**来控制显示 this.$refs.mv.show();

子组件的隐藏由**组件自己**控制关闭。this.hide();

注意：

如果this.$refs.mv里面有scroll组件，那么在show方法内要加定时器

setTimeout(() => {

this.$refs.scroll.refresh();

},20)

**因为dom 由 none =>block 重新渲染，scroll要等到浏览器渲染完毕再次刷新，以便重新计算可滚动高度。**

1. **一般计算属性都不要参数，方法才需要参数。**

比如：

排行榜列表组件，前几名是左边有奖杯的图片，后面几名都是文案(数字)，把列表的index作为参数传递给函数，动态计算样式。

<span :class="getRankLevel(index)" v-text="getRankText(index)"></span>

前几名是icon样式，有字体图标，后面排名有文案(数字)

// 前几个返回icon类名，后面几个返回text类名  
getRankLevel(index) {  
 if(index <= 2) {  
 return `icon icon${index}`;  
 }  
 return 'text';  
},  
// 类名为text的才会有内容  
getRankText(index) {  
 if(index > 2) {  
 return index + 1;  
 }  
},

**12. 路由动画要加在<router-view>标签外层，不是在路由组件内，否则transition**

**等动画无效。**

例如：

<!--歌手详情页出口-->

<transition name="slide">

<router-view></router-view>

</transition>

**13. 当动画为animation 不是transition时候, 样式如何定义?**

比如：点击，confirm弹出框 先放大，然后恢复到 scale(1), 只有入场动画

<transition name="confirm-fade">

<div class=”wrapper”>

</div>

</transition>

&.confirm-fade-enter-active {

// 外层动画 透明度增加

animation: confirm-fadein 0.3s;

// 内层动画，先放大在缩小

.confirm-content {

animation: confirm-zoom 0.3s;

}

}

**@keyframes** confirm-fadein {  
 0% {  
 opacity: 0;  
 }  
 100% {  
 opacity: 1;  
 }  
}  
**@keyframes** confirm-zoom {  
 0% {  
 transform: scale(0);  
 }  
 50% {  
 transform: scale(1.1);  
 }  
 100% {  
 transform: scale(1);  
 }  
}

**14. 列表动画transition-group**

<transition-group name="list" tag="ul">  
 <li class="item"

v-for="(item, index) in sequenceList"  
 :key="item.id"  
 >  
 <span class="delete" @click.stop="deleteOne(item)">  
 <i class="icon-delete"></i>  
 </span>  
 </li>  
</transition-group>

// 删除歌曲时候，高度变为0  
&.list-enter-active, &.list-leave-active {  
 transition: all 0.1s;  
}  
&.list-enter, &.list-leave-to {  
 height: 0;  
}

里面的li必须要有**:key**， 并且值是唯一性，不能是:key=”index”, 能是

**:key=”item.id” or :key=”item”** 只要确保唯一性就行。

1. **异步数据请求，有jsonp 和 ajax(axios)两种**

**(1). jsonp请求**

**// jsonp.js**

import originJSONP from 'jsonp';  
// 问号后面查询  
export function queryParams(data) {  
 let url = '';  
 for(const k in data) {  
 if (Object.hasOwnProperty.call(data, k)) {  
 const value = typeof data[k] !== 'undefined' ? data[k] : '';  
 url += '&' + k + '=' + encodeURIComponent(value);  
 }  
 }  
  
 return url ? url.slice(1) : '';  
}

export default function jsonp(url, datas, option) {  
 url += url.indexOf('?') < 0 ? '?' + queryParams(datas) : queryParams(datas);  
 return new Promise((resolve, reject) => {  
 originJSONP(url, option, (err, data) => {  
 if(!err) {  
 resolve(data);  
 } else {  
 reject(err);  
 }  
 });  
 });  
}

**使用：**

function getTopList(topid) {

// 真实数据地址

const url = 'https://c.y.qq.com/v8/fcg-bin/fcg\_v8\_toplist\_cp.fcg';

// 真实接口的参数

const data = {

topid,

platform: 'h5',

...

};

**return jsonp(url, data, { param: 'jsonpCallback' });**

}

**// 外部调用(在vue文件内)**

getTopList(this.topid)

.then(res => console.log(res));

1. **. axios请求**

**前端部分：**

// api.js

// 获取轮播图数据（移动端抓取）

function getPicList() {

const url = ‘/api/getPicList’;

// 里面参数都是复制浏览器请求接口的参数

const params = {

g\_tk: 5381,

format: 'json',

...

};

return axios.get(url, {

params,

});

}

**// 外部调用(在vue文件内调用)**

getPicList().then(res => console.log(res))

**后端处理：**

const express = require(‘express’);

const routes = express.Router();

routes.get('/getPicList', (req, res) => {

// 真实api接口

const url= 'https://c.y.qq.com/mu

sichall/fcgi-bin/fcg\_yqqhomepagerecommend.fcg';

axios.get(url, {

headers: {

'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8;',

},

params: req.query,

}).then((response) => {

// 注意，这里返回的是response.data,而不是response

res.json(response.data);

}).catch(e => console.log(e));

});

**app.use('/api', routes);**

**app.listen(8900, (error) => ...)**

1. **当对ajax请求到的数据进行格式化的时候，要不要直接赋值给data中数据？**

格式化数据\_normalize时候，如果\_normalize里面**没有ajax请求**，可以利用中间数组**ret**来赋值，

// 因为是**同步**，可以这样

this.songs = \_normalize(list);

\_normalize(list){

const ret = [];

...

return ret;

}

如果\_normalize里有**ajax请求**，就不能直接用中间数组ret来赋值了，

// 因为是**异步请求**，所以**不能**ret数组来赋值

\_normalize(list);

\_normalize(list){

...

this.songs.push(...);

}

**17. 如果ajax请求成功，返回状态码200,但是数据没有返回出来,状态码是这样**

{code: 0, subcode: 1, msg: "invalid referer"} 又或者返回的信息是

{"code":0,"message":"禁止跨域访问","subcode":-10001,"tips":"Refer Error"}

表示对**referer作出限制**，所以需要伪造一下referer。

axios.get(url, {

headers: {

'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8;',

// 在请求头部可以找到

referer: 'https://y.qq.com/',

**// 不需要加host**

},

params: req.query,

}).then((response) => {

**// 注意，这里返回的是response.data,而不是response**

res.json(response.data);

}).catch(e => console.log(e));

**注意：**

最后要加个/, 因为不加/的话，是'https://y.qq.com', 表示的是协议+域名+端口号(是**origin**)

如果加个 / , 即https://y.qq.com/，表示的是 协议+域名+端口号

+路径(是**referer**), 最后的/ 代表的就是路径，不加/，就是origin而不是referer

**18.比如qq音乐的获取歌词接口是个jsonp请求(script标签js请求)，但如果你想用axios请求(ajax)的话，去掉callback请求参数，加个 format: 'json'. 可能会遇到以下问题：**

**(1). 如果请求返回200状态码，但是reponse没有数据，返回的是 code: -1310,**那么url肯定被限制了。

观察jsonp请求，发现**referer**:

https://i.y.qq.com/v8/playsong.html?songmid=003OUlho2HcRHC&ADTAG

其中对songmid做了限制，所以在后端axios请求的时候，要加上referer这个字段。

routes.get('/getLyric', (req, res) => {

const url = 'https://c.y.qq.com/lyric/fcgi-bin/fcg\_query\_lyric.fcg';

axios.get(url, {

headers: {

'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8;',

**referer:`https://i.y.qq.com/v8/playsong.html?songmid=${req.musicmid}&ADTAG=myqq&from=myqq&channel=10007100`,**

},

params: req.query,

}).then((response) => {

**// 注意，这里返回的是response.data,而不是response**

res.json(response.data);

}).catch(e => console.log(e));

});

**(2). 成功请求到数据，但还是JSONP格式的数据字符串格式，这时候用正则过滤，比如：**

// **“MusicCallback({data:...})”,** 只要获取里面的{}内容

const reg = /^\w+\((\{[^()]+\})\)$/; // 注意**不能加g**，加g就是数组里面的元素全是完整匹配，没子项

let ret = response.data;

// callback回调函数的字符串形式

if(typeof ret === 'string') {

const matches = ret.match(reg);

if(matches) {

// 把json字符串转为JSON对象

ret = JSON.parse(matches[1]);

}

}

**res.json(ret);**

**(3). 如果数据内容编码了,比如看到 &#组合，就是'HTML实体编码' 即 unicode编码，对其解码**

// lyric是带有unicode编码的字符串

**第一种：**

this.lyric = lyric.replace(/&#(\d+);/g, (str, match) => String.fromCharCode(match));

// 注意正则不能带 ^和$，因为歌词是多行(没带\n)，带上m也不行，

// 因为m针对的是带有 \n的字符串。

**第二种：**

const div = document.createElement('div');

div.innerHTML = lyric;

return div.textContent || div.innerText;

1. **qq音乐，先获取歌单 歌曲列表，然后获取每个歌曲对应的vkey，最后把vkey+原歌曲数据组成新的歌曲数据。**

**因为有时ajax请求获取vkey返回200成功状态码，但数据却类似这样**

0: {common\_downfromtag: 0, errtype: "", filename: "", flowfromtag: "", flowurl: "", …}

1: {common\_downfromtag: 0, errtype: "", filename: "", flowfromtag: "", flowurl: "", …}

所以需要反复的发送请求获取vkey，这里利用到了**递归**的思想。

**解决：**

利用递归的思想，如果请求返回的数据里面filename为空，则重新请求。

例如：

this.$api.music.getSongVkey(songMidArr)  
 .then(({ data: { req\_0: { data: { midurlinfo } } } }) => {  
 // 正确的数据是每个item里面都有filename  
 if(midurlinfo.every(item => !item.filename)) {  
 // 因为有可能一直无限获取vkey，所以作出请求次数限制  
 this.vkNum++;  
 if(this.vkNum >= MAXNUM) {  
 return;  
 }  
 // return不能省，之后的list.forEach不会执行，性能好一点  
 return this.\_normalizeSongs(list);  
 }

});

这样就保证能获取到正确的**filename**和**vkey**了。

1. **如果scroll组件中有不确定尺寸的图片时候，也会导致better-scroll组件滚动无效，所以应该在img的load事件中 bs.refresh() 重新刷新，以便重新计算可滚动的高度。但是load事件只需要执行一次。**

// 完成，重新刷新BScroll高度  
loadImg() {  
 // 只执行一次  
 if(!this.checkImg) {  
 this.checkImg = true;  
 // 通知父级滚动条刷新  
 this.$emit('refresh');  
 }  
},

1. **当轮播图开启 自动循环时候，当切换tab时候，组件失活，会触发deactivated方法，当再次进入轮播图所在组件时候，组件激活，触发activated方法，**

组件失活，关闭定时器，组件激活，重新开启定时器

// 组件激活时执行，开始定时器  
activated() {  
 if (this.autoPlay) {  
 this.\_play();  
 }  
},  
// 组件失活时候执行,关闭定时器  
deactivated() {  
 clearTimeout(this.timer);  
},

1. **watch的一些应用**

(1). 根据轮播图的图片的个数来计算 圆点 dot的个数

watch: {

// 图片数组的个数  
 recommends(newVal) {  
 this.dots = new Array(newVal.length);  
 },  
},

(2). 滚动组件scroll根据props传入的数据的变化来 bs.refresh();

watch: {  
 // 当数据变化时候,重新计算滚动高度， 浏览器渲染需要时间，所以价格定时器，  
 data() {  
 setTimeout(() => {  
 this.refresh();  
 }, this.refreshDelay);  
 },  
},

1. . 实时监测滚动的位置，来计算currentIndex

// 实时监测滚动，然后计算出currentIndex  
scrollY(newVal) {  
 const { listHeight } = this;  
 // 当滚动到顶部时候，向上滚动是正值，newVal > 0,  
 if(newVal > 0) {  
 this.currentIndex = 0;  
 return;  
 }

1. . 歌曲播放时间所占总时间的百分比，然后监听这个百分比来计算内层进度条的宽度。

watch: {  
 percent(newVal) {  
 // 当没拖动时候在监测  
 if(this.percent > 0 && !this.touch.inited) {  
 // 可移动的bar宽度  
 const barWidth = this.$refs.progressBar.clientWidth - this.$refs.btnWidth.clientWidth;  
 // 当前的宽度  
 const offsetWidth = newVal \* barWidth;  
 this.\_offset(offsetWidth);  
 }  
 },  
},

1. . 监听当前歌曲的变化, 切换歌曲时候触发, 监听当前歌曲的状态，是暂停还是播放
2. . 监听input输入框的变化，实时派发事件给父组件来处理 业务逻辑，

因为要用防抖事件，所以在created中以方法方式(this.$watcher)调用

// 防抖  
this.$watch('query', debounce((newQuery) => {  
 this.$emit('query', newQuery);  
}, 200));

父组件实时监听子组件派发的事件，来发送ajax请求

watch: {  
 // 当搜索关键词改变就触发 ajax请求  
 query(newQuery) {  
 this.\_getSearchList(newQuery, this.page, perpage);  
 },  
},

1. **vuex中的一些技巧**

(1). 把store分成单独文件，state.js, getter.js, mutation.js, actions

(2). commit中定义的方法最好用变量声明，这样不容易写错，可维护性增强

// mutation-type.js

export const SET\_FULL\_SCREEN = 'SET\_FULL\_SCREEN';

// mutation.js

[types.SET\_FULL\_SCREEN](state, flag) {  
 state.fullScreen = flag;  
 },

(3). 当state中的某个数据为数组时候，commit时候，不要直接赋值，否则

会造成错误。因为**引用类型**，容易造成改变，要对数组**slice** 拷贝一下

[types.SET\_SEQUENCE\_LIST](state, list) {

// list是数组类型，  
 state.sequenceList = list.slice();  
},

获取的时候也要拷贝一下.

const playList = state.playList.slice();  
const sequenceList = state.sequenceList.slice();

(4). actions不仅可以调用异步方法，也可以在actions方法内调用一堆commit

把actions方法当做一个集合的操作

// actions.js

// 清空播放列表  
export const clearSongList = function ({ commit }) {  
 commit(types.SET\_PLAYLIST, []);  
 commit(types.SET\_SEQUENCE\_LIST, []);  
 commit(types.SET\_CURRENT\_INDEX, -1);  
 commit(types.SET\_PLAYING\_STATE, false);  
};

1. **vue-router的一些技巧**

(0). <router-view> 一般放在一级路由页面里面

(1). 使用动态路由(路由懒加载)，性能优化

const Rank = () => import(/\* webpackChunkName: "rank" \*/'@components/rank/rank');

(2). 路由重定向

{  
 path: '/',  
 redirect: '/recommend',  
},

{  
 path: '\*',  
 redirect: '/',  
},

1. . 二级路由，使用js编程方式, 针对的是二级路由，不是一级路由

{  
 path: '/rank',  
 name: 'rank',  
 component: Rank,  
 children: [  
 {  
 path: ':id',  
 name: 'top-list',  
 component: TopList,  
 },  
 ],  
},

可以用path 或者 name + params方式

// 选择排行榜详情  
selectItem(item) {  
 this.$router.push({  
 path: `/rank/${item.topId}`,  
 });  
 this.setTopList(item);

// name + params

this.$router.push({

name: ‘top-list’; // name是二级路由，不是一级路由

params: {

id: item.topId

}

})  
},

1. . 返回上一页

this.$router.back();

(5). 路由之间的数据通讯，通过vuex来实现，点击歌手，把当前歌手信息存入到vuex中，然后进入歌手详情页时候，通过vuex获取到当前歌手的信息，然后根据其id来发送ajax请求，然后渲染页面。

**25. 滑动歌词页面**

一共有两页，当前歌曲页面，歌词页面(隐藏在右侧，运动的也是歌词页面)。

根据当前是什么页面，来判断歌词页面的初始left为多少。

const left = this.currentShow === 'cd' ? 0 : -window.innerWidth;

从歌词页面向左滑动的时候，如果歌词页面向左移动的距离大于当前手机视口的10%的时候，那就直接覆盖当前歌曲页面，否则回到原来的位置。

但要对滑动的偏移作出限制，向左滑动最大距离为-window.innerWidth,向右滑动最大距离为0，

// 滑动的偏移  
const offsetWidth = Math.min(0, Math.max(-window.innerWidth, left + deltaX));  
// 滑动的距离所占视口宽度的百分比  
this.touch.percent = Math.abs(offsetWidth / window.innerWidth);

**26. transition动画用到js钩子时候，需要判断 两个点的x,y差值。**

从哪(目标位置) 到哪(当前位置)，以**当前位置**为**原点**(0,0),

x向右和y向下为正，不一定非要目标值-当前值，只要按照原点的位置，来计算正负就行(从原点位置向目标位置走过去，先走横坐标，再走纵坐标)。

目标x/y - 当前x/y

1. **左侧歌手 + 右侧首字母联动**

左侧歌手 <li class=”item”>

<h1>C</h1>

<ul>

<li>陈奕迅</li>

<li>蔡依林</li>

</ul>

</li>

把左侧大li的累加高度增加到数组里面。右侧也有一一对应的字母组成的数组

\_calculateHeight() {  
 // 多次重新计算的话，需要初始化为空数组  
 this.listHeight = [];  
 const list = this.$refs.listGroup;  
 let height = 0;  
 this.listHeight.push(height);  
 for(let i = 0, len = list.length; i < len; i++) {  
 const item = list[i];  
 height += item.clientHeight;  
 this.listHeight.push(height);  
 }  
},

**注意：**

第一个要把0加进去，因为第一个大li距离顶部为0，第二个大li距离顶部为第一个大li的高度，依次类推，所以需要**累加**。

当数据存在时候，页面渲染完毕后，再调用方法来累加高度。

watch: {  
 // 动态计算一级li高度  
 data() {  
 // 为什么不用this.$nextTick，因为不够准确，  
 // 会导致计算高度错误，所以用20ms一定能正确计算到  
 setTimeout(() => {  
 this.\_calculateHeight();  
 }, 20);  
 },

实时监测滚动来计算索引(currentIndex)时候时候要判断三种情况：

(1). 下拉，即滚动scrollY >0 (一般向下滚动scrollY < 0)

返回 0；

(2). 在中间滚动时候，

>=数组的第一个高度，<数组的第二个高度时候, 返回数组第一个高度的索引。以此类推。。。

(3). 滚动到底部时候，返回的索引应该是左侧高度数组的长度-2，为什么要

减2，因为左侧数组本身比右侧数组多一个元素(0)

例如：

[0,1,2,3,4] [1,2,3,4]

5-2 右侧最后一个元素索引3

watch: {  
 // 实时监测滚动，然后计算出currentIndex  
 scrollY(newVal) {  
 const { listHeight } = this;  
 // 当滚动到顶部时候，向上滚动是正值，newVal > 0,  
 if(newVal > 0) {  
 this.currentIndex = 0;  
 return;  
 }  
  
 // 在中间部分滚动  
 for(let i = 0; i < listHeight.length - 1; i++) {  
 const height1 = listHeight[i];  
 const height2 = listHeight[i + 1];  
 // 向下滚动是负值，向上是正值，要取绝对值  
 if((Math.abs(newVal) >= height1 && Math.abs(newVal) < height2)) {  
 this.currentIndex = i;  
 // differ是当前标题距离视口顶部的高度  
 // newVal是已经滚动的高度，是负值  
 this.differ = height2 + newVal;  
 // 终止循环  
 return;  
 }  
 }  
 // 当滚动到底部时候，并且 Math.abs(newVal) 大于最后一个元素的上限(累计高度之和)  
 // 为什么减去2，因为左边本身比右边多一个元素(0), 所以要减2，就是右边最后一个  
 // 比如右边 21个，最大索引20， 左边22个，最大索引也应该是20,所以要减去2  
 this.currentIndex = listHeight.length - 2;  
 },  
 differ(newVal) {  
 const TitleHeight = this.$refs.fixed.clientHeight;  
 // 意思就是两者接触的时候，fixedTitle往上走  
 // 当两者相距很远时候，都是为0  
 const fixedTop = (newVal > 0 && newVal < TitleHeight) ? newVal - TitleHeight  
 : 0;  
 // 第一次不存在，不走这里，以后都走这里  
 // 因为differ是动态计算，在多数情况下，this.fixedTop为0，  
 // 也就是相距很远时候，什么都不执行，fixedTitle不会被下面的标题顶上去  
 // 这样是为了性能优化  
 if(this.fixedTop === fixedTop) {  
 return;  
 }  
 // 主要为了上一步做的，多数情况下为0  
 this.fixedTop = fixedTop;  
 // fixedTitle被下面的标题 顶上去了  
 this.$refs.fixed.style.transform = `translate3d(0,${fixedTop}px,0)`;  
 },  
},

this.differ = height2 + newVal;

differ实际上就是下一个大li(标题)距离视口顶部的高度，因为

height2 是下一个大li的高度(正值)，而newVal是已经滚动过的高度(即窗口顶部的位置，为负值)。

当两者接触的时候，fixedTitle往上走，走的距离正好是标题的高度，看起来被下面的标题顶上去了。

当两者相距很远时候，都是为0 。

**左侧的滚动动态计算索引index结束**

右侧点击字母，左侧滚动到对应的大li标题位置。

\_scrollTo(index) {  
 // 点击右侧字母栏顶部空白黑色区域 或者 底部空白黑色区域时候  
 if(!index && index !== 0) {  
 return;  
 }  
 // 右侧字母栏按住滑动到顶部或者底部 时候，超出的处理  
 if(index < 0) {  
 index = 0;  
 } else if (index > this.listHeight.length - 2) {  
 index = this.listHeight.length - 2;  
 }  
 // 点击右侧字母，改变this.scrollY,触发watch中的this.currentIndex变化  
 this.scrollY = -this.listHeight[index];  
 this.$refs.scroll.scrollToElement(this.$refs.listGroup[index], 0);  
},

1. **audio播放器的一些技巧**

(1). 播放audio时候，chrome报错Uncaught (in promise) DOMException，同样的firefox会报错

AbortError: The fetching process for the media resource was aborted by the user agent at the user's request.

当第一次点击进入播放时候，会报错，原因是因为chrome66以上版本不能自动播放video，调用video.play()会报这个错误。firefox同理。

解决办法：

可以利用this.$nextTick来解决这个问题。

当快速切换歌曲时候，也会报这个错误。

解决办法：

在video上面绑个 canplay事件，用一个数据来表示歌曲加载是否成功的状态，songReady,刚开始songReady: false,当canplay事件触发了，才为true。在next, prev方法切换歌曲的时候，如果!songReady, return；然后在next, prev方法最后面重置 songReady: false.

注意：

绑定的是canplay事件，如果绑定play事件也会报错。play是audio.play()调用时候才会触发事件

(2). 播放顺序有3种状态，顺序(0)，单曲循环(1)，随机(2)

点击时候触发toggle方法，一直是这3种状态循环，

// 切换播放模式  
changeMode() {

// 不断是 0-2之间循环  
 const mode = (this.mode + 1) % 3;  
 this.setPlayMode(mode);  
},

1. . 根据当前播放顺序的三种状态，利用计算属性，分别对应不同的图标样式，

// 播放三种模式  
iconMode() {  
 return this.mode === playMode.sequence ? 'icon-sequence'  
 : this.mode === playMode.loop ? 'icon-loop'  
 : 'icon-random';  
},

1. **mixin.js里面可以共享vue组件内js的一些数据和方法。**

// 告诉组件必须重写handlePlaylist方法，重写后，就执行重写后的方法，  
// 如果不重写，就执行mixin中的handlePlaylist方法  
methods: {  
 handlePlaylist() {  
 throw new Error('component must implement handlePlaylist method');  
 },  
},

如果vue组件里有相同方法，就会覆盖mixins中的方法，如果没有，就用

mixins中的方法。让mixins中某个方法抛错，就使得vue组件内必须要重写这个方法，否则报错。

1. **打开歌曲列表时候，显示歌曲是否被收藏(收藏显示红心，没收藏显示灰心)**

判断是否收藏很简单，看vuex的收藏数组favoriteList是否存在这首歌曲，如果不存在，显示灰心样式，否则显示红心样式

// methods

// 是否收藏过  
isFavorite(song) {  
 const index = this.favoriteList.findIndex(item => item.id === song.id);  
 return index > -1;  
},

// 切换收藏 和 取消收藏的样式  
getFavoriteIcon(song) {  
 if(this.isFavorite(song)) {  
 console.log('getIcon');  
 return 'icon-favorite';  
}  
console.log('xxx');  
return 'icon-not-favorite';

},  
// 切换 收藏 或 取消收藏  
toggleFavorite(song) {

console.log('toggle');  
 if(this.isFavorite(song)) {  
 this.deleteFavoriteList(song);  
 } else {  
 this.saveFavoriteList(song);  
 }  
},

点击事件，切换是否收藏歌曲，然后动态计算歌曲样式。

<i class="icon"  
 @click="toggleFavorite(currentSong)"  
 :class="getFavoriteIcon(currentSong)"  
>

**如果没有收藏，点击一下，收藏了，当前i元素的 getFavoriteIcon会调用一下，**

**然后所有i(22个)又会重新调用一遍getFavoriteIcon方法。**

结果如下：

toggle

getIcon

3 getIcon(执行3遍)

19 xxx （执行19遍）

一共22个歌曲，收藏当前歌曲，会触发一下getIcon，返回红心样式，然后所有i(共22个)都重新计算一遍样式，结果当前歌曲列表一共有3个红心的，19个灰心的。这点跟**Computed计算属性**有点相似。

1. 下拉加载更多数据，数据hasMore默认为true，每次ajax请求后，就调用函数

\_checkMore(data)来判断是否还有数据，如果不存在hasMore为false，

每次滚动停止后看看是否快要滚动到底部了，如果滚动快到底，scroll组件派发个事件scrollEnd给父组件，然后调用searchMore函数，如果hasMore为false，什么都不做，否则继续ajax请求，然后在ajax请求后再次调用 \_checkMore(data)来判断是否还有数据，以此类推。

this.$api.music.getSearchList(query, page, perpages)  
 .then(({ data: { data } }) => {  
 // 因为\_normalizeSongs里有ajax请求，所以不能用赋值方式  
 this.\_normalizeSearchList(data);  
 // 检查是否还有数据  
 this.\_checkMore(data);  
 });

请求到的数据肯定有curnum, curpage,totalnum等参数

// 检测是否还有数据  
\_checkMore(data) {  
 const { song } = data;  
 if(!song.list.length || (song.curnum + (song.curpage - 1) \* perpage) >= song.totalnum) {  
 this.hasMore = false;  
 }  
},

监听滚动事件，是否滚动到底，快到底时候，调用searchMore函数

<scroll  
 class="suggest"  
 :data="result"

:pullup="pullup"  
 @scrollToEnd="searchMore"  
 @beforeScroll="listenScroll"  
>

如果没有更多数据(hasMore为false), 就什么不做，否则继续加载数据

// 下拉加载更多数据  
searchMore() {  
 // 没有更多数据的话  
 if(!this.hasMore) {  
 return;  
 }  
 this.page++;  
 this.$api.music.getSearchList(this.query, this.page, perpage)  
 .then(({ data: { data } }) => {  
 // 因为\_normalizeSongs里有ajax请求，所以不能用赋值方式  
 this.\_normalizeSearchList(data);  
 // 检查是否还有数据  
 this.\_checkMore(data);  
 });  
},

**总结：**

hasMore默认为false, 第一次请求完数据后，调用\_checkMore函数，如果没有更多数据，hasMore为false。

监听滚动事件，如果快要滚动到底，派发个事件scrollEnd给父组件，父组件调用searchMore函数,

如果hasMore为false(没有更多数据), 什么都不做，否则继续ajax请求数据，依次类推，重复下去。

1. **播放器进度条，点击进度条某个位置，就快进到某个位置的 实现方法**

滚动条的dom为 bar

// e.offsetX 获取不对，offsetX相对于外层定位的

相对于滚动条左侧的位置 = e.pageX - 滚动条左侧到页面左侧的位置

然而 滚动条左侧到页面左侧的位置 = bar.getBoundingClientRect.left

所以：

点击进度条某个位置(相对于滚动条左侧的位置) =

e.pageX - bar.getBoundingClientRect.left

然后让内部滚动条的宽度 = 点击进度条某个位置；

1. **用户中心的喜欢歌曲列表和最近播放列表过几天突然不能播放歌曲了**

原因：歌曲的vkey是有时间限制的，过了几天之后，之前歌曲的vkey也就过期。

解决：

每次打开用户中心时候，都重新请求两个列表中歌曲的vkey。

// 限制getVkey获取次数，有可能一直递归下去，所以作限制  
vkNum: 0,  
// 克隆一份本地playListHistory，因为vkey有时间限制，所以打开用户中心  
// 都要获取最新的vkey，防止播放错误。  
clonePlaylistHistory: [],  
// 同上  
cloneFavoriteList: [],

原来的favoriteList和playListHistory分别换成 cloneFavoriteList和

clonePlayListHistory，因为localstorage里面的数据不能随便改变，所以通过克隆一份数据，并且把里面能播放的数据添加到克隆数组中。

<scroll  
 class="list-scroll"  
 v-if="currentIndex === 0"  
 :data="cloneFavoriteList"  
 ref="favoriteList"  
>  
 <div class="list-inner">  
 <song-list @select="selectSong" :songs="cloneFavoriteList" ></song-list>  
 </div>  
</scroll>  
<!--最近播放列表-->  
<scroll  
 class="list-scroll"  
 v-if="currentIndex === 1"  
 :data="clonePlaylistHistory"  
 ref="playList"  
>  
 <div class="list-inner">  
 <song-list @select="selectSong" :songs="clonePlaylistHistory" ></song-list>  
 </div>  
</scroll>

mounted() {  
 // 为什么不直接传递playHistory给子组件渲染，因为playHistory里面歌曲的  
 // vkey可能过期了，导致播放错误，应该每次打开用户中心都要获取最新的vkey  
this.\_normalizeSong(this.favoriteList,this.cloneFavoriteList);  
 this.\_normalizeSong(this.playHistory,this.clonePlaylistHistory);  
},

重新发送请求获取vkey，把原先localStorage数据里面能播放的歌曲，存入到新的克隆数组中(cloneFavoriteList等).

// 格式化localStorage数据，主要是获取最新的vkey，不用原来localStorage中的vkey  
\_normalizeSong(list, cloneList) {  
 const songMidArr = list.map(item => item.mid);  
 if(songMidArr.length) {  
 this.$api.music.getSongVkey(songMidArr)  
 .then(({ data: { req\_0: { data: { midurlinfo } } } }) => {  
 // 正确的数据是每个item里面都有filename  
 if(midurlinfo.every(item => !item.filename)) {  
 // 因为有可能一直无限获取vkey，所以作出请求次数限制  
 this.vkNum++;  
 if(this.vkNum >= MAXNUM) {  
 return;  
 }  
 // return不能省，之后的list.forEach不会执行，性能好一点  
 return this.\_normalizeSong(list, cloneList);  
 }  
 list.forEach((item, index) => {  
 if(midurlinfo[index]) {  
 const { vkey, filename } = midurlinfo[index];  
 if(vkey) {  
 const obj = {};  
 for(const k in item) {  
 if(k === 'url') {  
 obj[k] = createUrl(vkey, filename);  
 } else {  
 obj[k] = item[k];  
 }  
 }  
 cloneList.push(obj);  
 }  
 }  
 });  
 });  
 }  
},

song-list组件内渲染的数据都是克隆后的数组clonePlayListHistory和

cloneFavoriteList