1.播放器动画效果

<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  
 }  
}

2.实现播放器播放功能

<!--canplay,error,timeupdate，ended是audio中的事件，canplay表示歌曲已经准备好了，error表示请求不到资源出现错误,timeupdate在歌曲播放时会不停的触发该事件,ended在歌曲播放完后会触发，play是当音频/视频已开始或不再暂停时

-->  
<audio :src="currentSong.**url**" ref="audio" @play="ready" @error="error" @timeupdate="timeupdate" @ended="ended"></audio>

// 添加动画效果，使播放器图片旋转  
cdCls() {  
 return this.**playing** ? 'play' : 'play pause'  
},

toggleplaying(){  
 // 改变vuexplaying的播放状态  
 this.**setPlaying**(!this.**playing**);  
},

currentSong(){  
 // 由于audio发送请求需要时间，所以我们要延迟执行  
 this.$nextTick(()=>{  
 this.$refs.audio.play();  
 });  
},  
playing(newPlay){  
 // 由于audio发送请求需要时间，所以我们要延迟执行  
 this.$nextTick(()=>{  
 newPlay?this.$refs.audio.play():this.$refs.audio.pause();  
 })  
}

&.play  
 animation: rotate 20s linear infinite  
&.pause  
 *//暂停动画效果* animation-play-state: paused

3.圆形进度条

<div class="progress-circle">  
 <!--svg是网上的组件width和height分别设置他的宽和高，viewBox定义他的视图大小-->  
 <svg :width="**radius**" :height="**radius**" viewBox="0 0 100 100" version="1.1" xmlns="http://www.w3.org/2000/svg">  
 <!--迷你播放器暂停播放内层的圆-->  
 <circle class="progress-background" r="50" cx="50" cy="50" fill="transparent"/>  
 <!--迷你播放器暂停播放外层的圆，r是圆半径，cx和cy是他的圆点坐标，stroke-dasharray是描边的距离，这里我们描边的距离是整个圆，stroke-dashoffset是描边的偏移，这里是指我们的圆形进度条，stroke-dashoffset为0是表示整个圆都描满，即歌曲播放完毕，stroke-dashoffset与stroke-dasharray的值相等，描边完全偏离出去，即歌曲刚开始播放-->  
 <circle class="progress-bar" r="50" cx="50" cy="50" fill="transparent" :stroke-dasharray="**dashArray**"  
 :stroke-dashoffset="dashOffset"/>  
 </svg>  
 <slot></slot>  
</div>

**props**:{  
 // 圆的直径  
 **radius**:{  
 **type**:Number,  
 **default**:100  
 },  
 // 圆形进度条  
 **percent**:{  
 **type**:Number,  
 **default**:0  
 }  
},  
data(){  
 return{  
 **dashArray**:Math.PI\*100  
 }  
},  
**computed**:{  
 // 描边偏移量  
 dashOffset(){  
 return (1-this.**percent**)\*this.**dashArray**;  
 }  
}

4.歌词

本地服务器

app.get('/getLyric', function (req, res) {  
 var url = 'https://c.y.qq.com/lyric/fcgi-bin/fcg\_query\_lyric\_new.fcg'  
 // 发送请求  
 axios.get(url, {  
 // 请求头，欺骗qq音乐后台，使它误认为发出请求的是https://c.y.qq.com/  
 **headers**: {  
 **referer**: 'https://c.y.qq.com/',  
 **host**: 'c.y.qq.com'  
 },  
 // 请求所需要的参数,axios会自动拼接成xx=xx&yy=yy  
 **params**: req.**query** }).then((response) => {  
 // 请求成功后返回数据给前端  
 var ret = response.**data** if (typeof ret === 'string') {  
 var reg = /^\w+\(({[^()]+})\)$/  
 var matches = ret.match(reg)  
 if (matches) {  
 ret = JSON.parse(matches[1])  
 }  
 }  
 res.json(ret)  
 }).catch((e) => {  
 console.log(e)  
 })  
})

前端获取数据

export function getLyric(mid) {  
 const url='/getLyric';  
 var data=Object.assign({},commonParams,{  
 **songmid**: mid,  
 **platform**: 'yqq',  
 **hostUin**: 0,  
 **needNewCode**: 0,  
 **categoryId**: 10000000,  
 **pcachetime**: +new Date(),  
 // 因为后台是用res.json()返回数据的,所以我们这里要告诉他我们要的是json

**format**: 'json'  
 });  
 return axios.get(url,{  
 **params**:data  
 }).then((res)=>{  
 return Promise.resolve(res.data);  
 })  
}

getLyric(){  
 if(this.**lyric**){  
 return Promise.resolve(this.**lyric**)  
 }  
 return new Promise((resolve,reject)=>{  
 getLyric(this.**mid**)  
 .then((res)=>{  
 if(res.retcode==ERR\_OK){  
 // 解析字符编码  
 this.**lyric**=**Base64**.decode(res.**lyric**);  
 resolve(this.**lyric**)  
 }else {  
 reject('no lyric');  
 }  
 })  
 })

player组件内

// 获取歌词  
getLyric(){  
 this.currentSong.getLyric().then((res)=>{  
 // 把歌词变成一个类，第一个参数是歌词，第二个参数是回调函数，每次滚动歌词的时候都会触发该函数  
 this.**currentLyric**=new Lyric(res,this.handelLyric);  
 // 当前正在播放才开始滚动歌词  
 if(this.**playing**){  
 this.**currentLyric**.play();  
 }  
 }).catch(()=>{  
 // 获取不到歌词的时候，清空数据  
 this.**currentLyric**=null;  
 this.**playingLyric**='';  
 this.**currentLineNum**=0  
 });  
},

// lineNum当前是第几行，txt该行的是歌词  
handelLyric({lineNum, txt}){  
 this.**currentLineNum**=lineNum;  
 // 修改当前正在播放的歌词  
 this.**playingLyric**=txt  
 // 歌词的行数大于5才开始滚动，这样保证了歌词处于中间  
 if(lineNum>5){  
 var el=this.$refs.lyricLine[lineNum-5]  
 this.$refs.lyricList.scrollToElement(el,1000)  
 }else{  
 this.$refs.lyricList.scrollTo(0,0,1000)  
 }  
},

5.播放器歌词左右滑动效果

middleTouchStart(e){  
 this.**touch**.**initiated**=true;  
 var touch=e.touches[0];  
 this.**touch**.**startX**=touch.pageX;  
 this.**touch**.**startY**=touch.pageY;  
 },  
 middleTouchMove(e){  
 if(!this.**touch**.**initiated**){  
 return;  
 }  
 var touch=e.touches[0];  
 var delatx=touch.pageX-this.**touch**.**startX**;  
 var delaty=touch.pageY-this.**touch**.**startY**;  
 // 因为scroll组件可以向上滚动，当在y轴上的差值大于x轴上的差值是，说明是在滚动scroll组件。不是切换cd页面和歌词页面  
 if(Math.abs(delaty)>Math.abs(delatx)){  
 return;  
 }  
 // 记录歌词页面所在的位置，距离屏幕左边的位置  
 var left=this.**currentPage**=='cd'?0:-window.innerWidth;  
 // 手指移动的距离，也是歌词移动的距离，如果当前页是cd页面，手指从左向右滑，left+delatx>0,offsetWidth=0;手指从右向左滑，left+delatx<0,offsetWidth=手指滑动的距离。如果当前页是歌词页面，手指从左向右滑，left+delatx<0,offsetWidth=手指滑动的距离;手指从右向左滑，left+delatx<0,offsetWidth=-window.innerWidth-100  
 var offsetWidth=Math.min(0,Math.max(-window.innerWidth-100,left+delatx));  
 // 当前移动的距离占可移动最大距离的百分比  
 this.**touch**.**percent**=Math.abs(offsetWidth/window.innerWidth)  
 // 设置过渡  
 this.$refs.lyricList.**$el**.**style**[transitionDuration] = 0  
 // 移动歌词页面  
 this.$refs.lyricList.**$el**.**style**[transform]=`translate3d(${offsetWidth}px,0,0)`  
 this.$refs.middleL.**style**.**opacity** = 1 - this.**touch**.**percent** this.$refs.middleL.**style**[transitionDuration] = 0  
 },  
 middleTouchEnd(e){  
 var offsetWidth;  
 var opacity;  
 if(this.**currentPage**=='cd'){  
 // 从右向左移动的距离大于20%，切换成歌词页面  
 if(this.**touch**.**percent**>0.2){  
 offsetWidth=-window.innerWidth;  
 opacity=0  
 this.**currentPage**='lyric';  
 }else{  
 // 从左向右  
 opacity=1;  
 offsetWidth=0;  
 }  
 }else{  
 // 当前页是歌词页时  
 // 从左向右移动的距离小于80%，切换成cd页面  
 if(this.**touch**.**percent**<0.8){  
 offsetWidth=0;  
 this.**currentPage**='cd';  
 opacity=1;  
 }else {  
 opacity=0;  
 offsetWidth=-window.innerWidth;  
 }  
 }  
 this.$refs.lyricList.**$el**.**style**[transitionDuration] = '400ms'  
 this.$refs.lyricList.**$el**.**style**[transform]=`translate3d(${offsetWidth}px,0,0)`  
 this.$refs.middleL.**style**.**opacity** = opacity  
 this.$refs.middleL.**style**[transitionDuration] = '400ms'  
 },

6.播放器下方的控制器与歌词同步

// 修改歌曲播放的进度  
this.$refs.audio.**currentTime**=currentPercent;  
if(this.**currentLyric**){  
 this.**currentLyric**.seek(currentPercent\*1000)  
}

if(this.**currentLyric**){  
 // 循环播放时让歌词回到时间为0的时间点上  
 this.**currentLyric**.seek(0)  
}

if(this.**currentLyric**){  
 // 点击暂停按钮时让歌词也暂停  
 this.**currentLyric**.togglePlay();  
}

if(this.**currentLyric**){  
 // Lyric可以自动滚动是因为内有定时器，但我们切换歌曲时，由于定时器没有清除，所以歌词会不断地跳动，所以这里我们要清除定时器  
 this.**currentLyric**.stop();  
}