1. **播放列表首页:**
2. **轮播图**

用observers数据监听器监听playlist.playCount的变化（这里面就是当其发生变化时要做的事情）,在这里就是给count赋值(但一定不能给监听数据的本身赋值，不然就会再次触发监听器造成死循环，要在data中再设个值并赋值)

* 要做的事情(methods中的方法)： 把收听数转换成万或亿单位

1. 传入要转换的数和小数点后几位
2. 先忽略普通数后面的小数为并转换成string
3. 如果这个string长度小于6(小于6位数，10万以下)

直接不做改变

1. 如果是10万以上，1亿以下：

获取小数位， 从倒数第四位开始，到倒数第四位加上传进来小数点后几位的数(万和千的单位)

用传来的值/10000获取整数位，在拼接加上小数位

1. 如果是1亿以上，和上面执行差不多，小数位为千万和百万的单位

**3. wx:key,** 它的意义主要是给每个item绑定索引，这个索引(如id,….)都不能一样， 这样在排序等动态渲染下，其对应的item也会动态显示在不同地方

**4. 从接口中往云数据库传入动态数据**

- 建立getPlayList云函数，连接好云数据库，并用axios从相应接口获取数据

- 从云函数入口函数中拿到数据，如果data.code>=1000证明接口有问题return 0

- 如果接口没问题拿到data.result

- 当playList不为空时:

把playList数组一次性传入云数据库的playlist集合中

**5.插入的数据去重: 目的在于防止插入已经在云数据库中存在的值(值不能重复)**  
 - 设个值，把它赋予从云数据库playlist集合中get到的所有信息

- 设个空数组，用于存放最后要的值

- 分别遍历接口中拿到的值和playlist集合中的值

- 把每个的id进行对比，如果两个id相等，则不要这个一样的，不相等才要

- 把传入云数据库的值改为最后要的值的数组

1. **数据多次获取(因为云服务器每次最多获取100条，如果想获取更多，就要自己写多次获取的功能)**

* .count获取到playlist集合中数据的总条数(拿到的是个对象，要用.total转换成数量)
* 拿到需要获取的总次数
* 建立一个空数组，用来存放每一次获取到的promise
* 遍历每一次获取，用.skip来定位每一次从多少条数据开始获取，每次limit限制为max\_limit , 再把每一次获取的promise放到刚刚的空数组中
* 设个空对象(这个对象就是原先5里面赋予从云数据库playlist集合中get到的所有信息)
* Promise all 之前存promise的数组, 用reduce把所有的叠加到一起，赋值到刚刚的空对象中，返回值为两个参数concat合并

1. 定时触发云函数

* 建立一个json文件, 里面有name, type, config属性， config为 “0 0 10, 14,16,23 \* \* \* \*”， 代表了每天上午10点，下午2点，4点，晚上11点出发此函数对数据更新
* 配置超时时间为20秒

**2.Music 云函数(管播放列表首页的后端)**

1. 从playlist集合中拿到数据显示到前端，分别用skip start，和limit count来定义每页显示的数据以及从第几条数据显示
2. 根据创建时间排序，逆序
3. 拿到后返回结果
4. 去到前端playlist组件:

* 拿到music云函数，设置参数中start和count的值
* 载入时显示wx.loading的弹窗
* 触底时要再调用整个过程
* 把playlist的值赋予当前playlist的值再合并上这次拿到的结果的data

1. 下拉刷新: 在当前组件的json文件中，enablePullDownRefresh为true. 再去到js文件中相应钩子函数中清空playlist数组，再重新获取整个页面数据
2. **云函数优化**(tcb-router, 实现一个云函数管多个路由，使用洋葱模型):

安装tcb-router, 拿到它的实例化，把1的逻辑放进app.router里，之后再serve

去到pages前端，获取这个云函数的data里写$url:router名

**3. 每一个播放列表(musicList)**

**1.**

建立musicList页面，接收music云函数，把options里的playlistId和musiclist路由名作为data

后端:

* 在music云函数中建立musicList路由，在里面拿到当前playlistid的url， 把它的data赋予ctx.body(注意url的拼接)

**2.播放列表头部:**

* 建立一个数组musiclist(用于存放当前歌单的所有歌), 再建立一个对象listInfo(用于存放歌单头部显示的那几个属性，\*如图片，歌单名等等)
* 执行完后端拿到的结果(res)，在里面逐层往里找到所有歌单的属性，把它赋予musiclist，再把每个歌单的cover图片和名字赋予listInfo
* 在wxml页面中渲染

**3.musicList组件(用来显示歌单下面所有的歌)**

* 把musicList page中的musiclist数组传给这个组件
* 再在其wxml中循环渲染每一首歌

**4.选择当前播放音乐并显示为红色:**

* 给每一首歌的view中添加绑定事件, 再传一个data-musicid为当前这首歌的id
* Data中设一个数值playId(代表当前播放id)初始为-1
* 在事件方法中传入event(事件源),
* 获取其中currentTarget.dataset.musicid, 再把它赋予playId(把点击的这首歌的id和设的playId绑定)
* 回到wxml页面，判定如果当前歌曲的id（item.id）与playId相等，就证明这首歌被选中，就给它添加一个class, 让此标签里的字都变成红色
* 再回到事件中，点击这首歌，就跳转到player页面(播放页面), 注意给他的url后面要加上currentTarget.dataset.musicid
* **\*这里说明为何要用currentTarget, 因为js事件默认是冒泡的，所以不管你点到这个绑定此事件view里的任何标签，他都会出发绑定了view的事件，所以不用target(你点到的位置，属于view的某标签), 而用currentTarget(最外层绑定此事件的标签)**