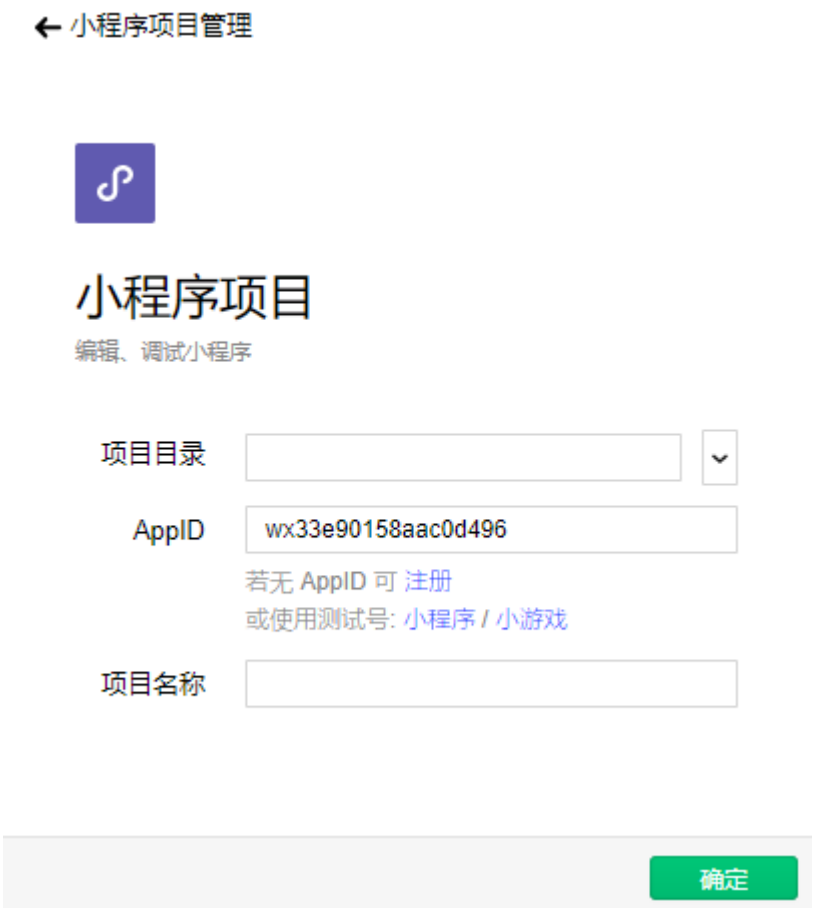
1、注册账号（进入以下页面），下面是已经填写信息完毕的了，如果刚注册的账号，那么需要完善信息之后才可以进行开发小程序

2、进行小程序基本的信息配置，比如小程序名称等等

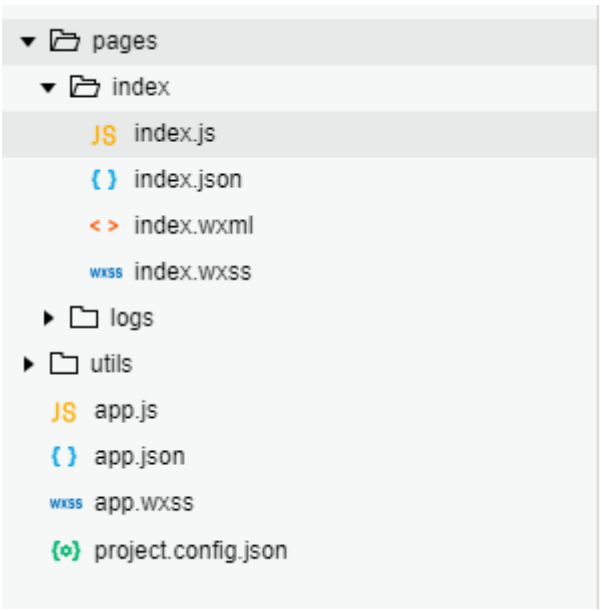
3、点击开发获取appid后进行开发小程序

4、appid的使用开发配置


小程序项目管理
小程序项目
编辑、调试小程序
项目目录
AppID
wx33e90158aac0d496
若无 AppID 可 注册
或使用测试号: 小程序 / 小游戏
项目名称
确定

5、请求开发配置

6、开始文件说明



1、index.js全部是数据

2、index.wxml就是h5里边的 html文件，显示index.js 里边的数据

3、index.json全局配置文件

4、全局配置样式

7、开发流程及其小程序发布

1、点击前往发布后，发布成功后就可以在小程序搜索栏进行搜索

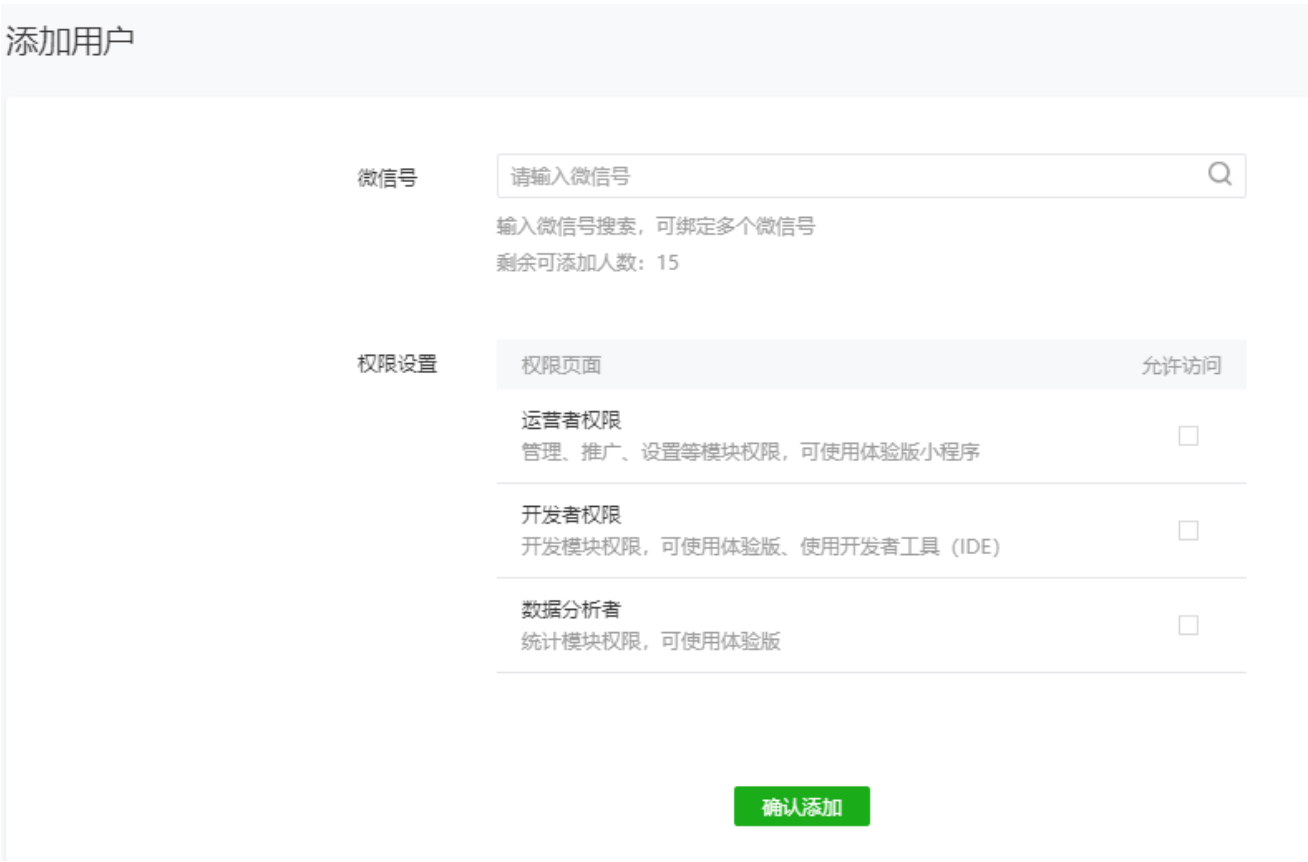
8、项目上线流程步骤

9、小程序配置团队开发

1、点击成员管理后，在蓝色编辑处进行添加成员，之后就可以进行团队开发。

2、此处可以进行相关的权限设置，可以点击查看。

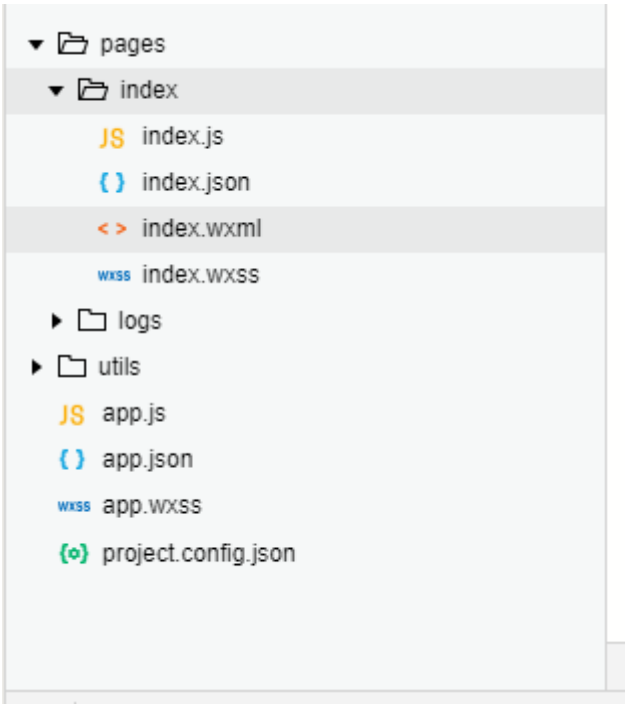
10、添加开发人员



1、点击添加就可以把开发人员或者体验人员给添加进来

2、一般在公司开发项目会有专门的开发账号

11、代码构成



1、.js逻辑文件，书写功能的文件。

2、.json配置文件，比如路由等的文件。

3、.wxml布局文件，相当于h5的html文件。

4、.wxss样式文件，相当于h5的css文件。

5、.wxs针对小程序的一套脚本语言，可以直接写在.wxml里边。

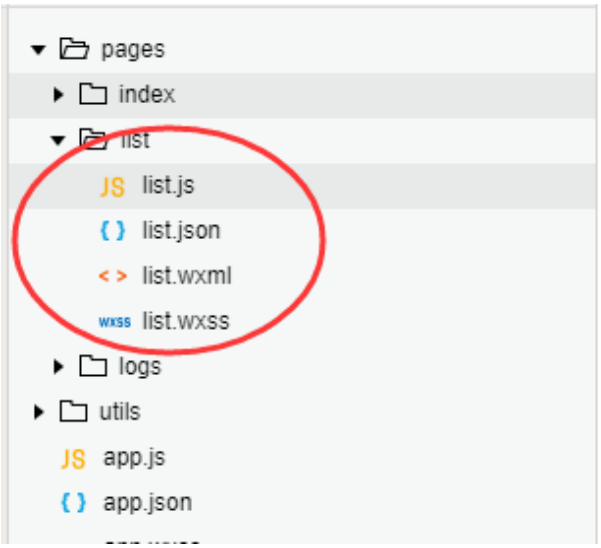
6、`app.json` 是当前小程序的全局配置，包括了小程序的所有页面路径、界面表现、网络超时时间、底部 tab 等。

12、详细梳理目录文件

1、utils里边的文件都是全局配置文件，只要在里边进行设置，其他页面都会跟着改变(在没有其他页面进行相应文件配置的时候，比如页面颜色等)

2、app.json在这个文件里边的“pages”文件里边书写页面，那么书写的页面会自动在文件里边显示出来，比如添加一个" pages/list/list",那么，list文件会自动配置出来。如图所示。

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "WeChat",
    "navigationBarTextStyle":"black"
  }
}
```



3、在pages文件里边，配置的第一个文件是入口文件，谁放在第一，谁就会打开页面就会进入。

13、配置项的含义

1. `pages` 字段 —— 用于描述当前小程序所有页面路径，这是为了让微信客户端知道当前你的小程序页面定义在哪个目录。

2. `window`字段 —— 定义小程序所有页面的顶部背景颜色，文字颜色定义，上下拉弹出，以及上下拉的背景色。
3. tabBar设置全局导航，可以对导航进行样式，字体图标等的设置。
4. networkTimeout设置网络请求，可以设置ajax请求超过某些限度是自动取消等等。
5. debug跟console.log()差不多，都是进行信息打印，在控制台方便查看相应的信息。

14、基本代码的简单说明

```
{
//配置页面
  "pages":[
    "pages/index/index",
    "pages/list/list",
    "pages/details/details",
    "pages/car/car"
  ],
  //配置全局页面属性
  "window":{
    "backgroundTextStyle": "light",//下拉刷新的小圆点样式
    "navigationBarBackgroundColor": "#58bc58",//头部样式
    "navigationBarTitleText": "WeChat",//不要在这里设置这个，否则全局页面头部标题都会是WeChat
    "navigationBarTextStyle":"white",//导航栏标题颜色，仅支持 black / white
    "backgroundColor":"#FF0100",//下拉刷新背面的背景色
    "enablePullDownRefresh":true，//开启下拉刷新
    "onReachBottomDistance": "50px"//设置上拉刷新，（此处不确定是否正确）
  },
  "tabBar":{
    "selectedColor":"#58bc58",//设置导航选中时的颜色
    "backgroundColor":"#F0F0F0",//tab 的背景色
    "borderStyle":"#58BC58",//tabbar上边框的颜色，
    "list":[
      {
        "pagePath": "pages/index/index",//页面配置路径
        "text":"首页",//导航的名字（标题）
        "iconPath": "./img/index.png"，//未选中时的字体图标
        "selectedIconPath":"./selectImg/index.png"//选中时的字体图标
      },
      {
        "pagePath": "pages/list/list",
        "text": "列表",
        "iconPath": "./img/list.png",
        "selectedIconPath": "./selectImg/list.png"
      },
      {
        "pagePath": "pages/details/details",
        "text": "商品详情",
        "iconPath": "./img/details.png",
        "selectedIconPath": "./selectImg/details.png"
      },
      {
        "pagePath": "pages/car/car",

        "text": "购物车",
```

```
      "iconPath": "./img/car.png",
      "selectedIconPath": "./selectImg/car.png"
    }
   ]
  }
}
```

15、小程序的功能基本操作

1、.wxml文件

```
1、在.wxml文件里边把基本的文件目录写好，
2、{{username}}数据是写在.js的data里边的，
<text>数据显示：{{username}}</text>
<view>
<button bindtap='change'>改变数据</button>
</view>
```

2、.js文件

```
Page({
 data: {
  username:"商品详情"
 },
 //以下是自定义事件
 change(){
  this.setData({
   username:"老谢"
  })
 }
})
```

3、通过调用调用自定义事件，改变视图层的数据。

4、应该注意的是 改变数据 ，点击事件需要bind开头，一看到bind就说明是一个事件。

5、直接在事件里边引号里边写自定义事件名字就可以使用该事件了

6、特别注意自定义事件的数据的写法

```
change(){
  this.setData({
   username:"老谢"
})
 }
```

16、通过打印相关的数据，结果显示，在onShow这个生命周期里，只要切换页面都会触发，所以可以在此生命周期发起ajax请求；

```
/**
 * 生命周期函数--监听页面显示
 */
onShow: function () {
  console.log("onShow");
},
```

17、生命周期

1、【onLoad】页面加载时触发，一个页面只会调用一次，可以在onLoad的参数中获取，打开当前页面路径中的参数，所以可以在此发起ajax请求。

```
/**
 * 生命周期函数--监听页面加载
 */
onLoad: function (options) {
  console.log("onLoad");
},
```

2、【onReady】页面初次渲染完成时触发，一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。

```
/**
 * 生命周期函数--监听页面初次渲染完成
 */
onReady: function () {
  console.log("onLoad");
},
```

3、【onShow】页面显示，切换页面时触发。

```
/**
 * 生命周期函数--监听页面显示
 */
onShow: function () {
  console.log("onShow");
},
```

4、【onHide】页面隐藏/切入后台时触发。 如 `navigateTo` 或底部 `tab` 切换到其他页面，小程序切入后台等。

```
/**
 * 生命周期函数--监听页面隐藏
 */
onHide: function () {
  console.log("onHide");
},
```

5、【onUnload】页面卸载时触发。如 `redirectTo` 或 `navigateBack` 到其他页面时。

```
/**
 * 生命周期函数--监听页面卸载
 */
onUnload: function () {
  console.log("onUnload");
},
```

18、页面事件处理函数

1、【onPullDownRefresh】监听用户下拉刷新事件。

- 需要在 `app.json` 的 `window` 选项中或页面配置中开启 `enablePullDownRefresh` 。
- 可以通过 `wx.startPullDownRefresh` 触发下拉刷新，调用后触发下拉刷新动画，效果与用户手动下拉刷新一致。
- 当处理完数据刷新后， `wx.stopPullDownRefresh` 可以停止当前页面的下拉刷新。

```
/**
 * 页面相关事件处理函数--监听用户下拉动作
 */
onPullDownRefresh: function () {
  console.log("onPullDownRefresh");
},
```

2、【onReachBottom】监听用户上拉触底事件。

- 可以在 `app.json` 的 `window` 选项中或页面配置中设置触发距离 `onReachBottomDistance` 。
- 在触发距离内滑动期间，本事件只会被触发一次。

```
/**
 * 页面上拉触底事件的处理函数
 */
onReachBottom: function () {
  console.log("onReachBottom");
},
```

3、【onShareAppMessage】监听用户点击页面内转发按钮（``` 组件 `open-type="share"` ）或右上角菜单“转发”按钮的行为，并自定义转发内容。

注意：只有定义了此事件处理函数，右上角菜单才会显示“转发”按钮

```
 /**
  * 用户点击右上角分享
  */
 onShareAppMessage: function () {
   console.log("onShareAppMessage");
 },
```

```
Page({
 onShareAppMessage(res) {
   if (res.from === 'button') {
     // 来自页面内转发按钮
     console.log(res.target)
   }
   return {
     title: '自定义转发标题',
     path: '/page/user?id=123'
   }
 }
})
```

4、【onPageScroll(Object)】监听用户滑动页面事件。

scrollTop	Number	页面在垂直方向已滚动的距离（单位px）

注意：请只在需要的时候才在 page 中定义此方法，不要定义空方法。以减少不必要的事件派发对渲染层-逻辑层通信的影响。 注意：请避免在 onPageScroll 中过于频繁的执行 setData 等引起逻辑层-渲染层通信的操作。尤其是每次传输大量数据，会影响通信耗时。

5、【onResize(object)】

小程序屏幕旋转时触发。

6、onTabItemTap(Object)

点击 tab 时触发

```
Page({
 onTabItemTap(item) {
   console.log(item.index)
   console.log(item.pagePath)
   console.log(item.text)
 }
})
```

19、组件【button】的参数使用

1、.wxml

```
<button
  type="default"
  size="{{defaultSize}}"
  loading="{{loading}}"
  plain="{{plain}}"
  disabled="{{disabled}}"
  bindtap="default"
  hover-class="other-button-hover"
>
  default
</button>
<button
  type="primary"
  size="{{primarySize}}"
  loading="{{loading}}"
  plain="{{plain}}"
  disabled="{{disabled}}"
  bindtap="primary"
>
  primary
</button>
<button
  type="warn"
  size="{{warnSize}}"
  loading="{{loading}}"
  plain="{{plain}}"
  disabled="{{disabled}}"
  bindtap="warn"
>
  warn
</button>
<button bindtap="setDisabled">点击设置以上按钮disabled属性</button>
<button bindtap="setPlain">点击设置以上按钮plain属性</button>
<button bindtap="setLoading">点击设置以上按钮loading属性</button>
<button open-type="contact">进入客服会话</button>
<button open-type="getUserInfo" lang="zh_CN" bindgetuserinfo="onGotUserInfo">
  获取用户信息
</button>
<button open-type="openSetting">打开授权设置页</button>
```

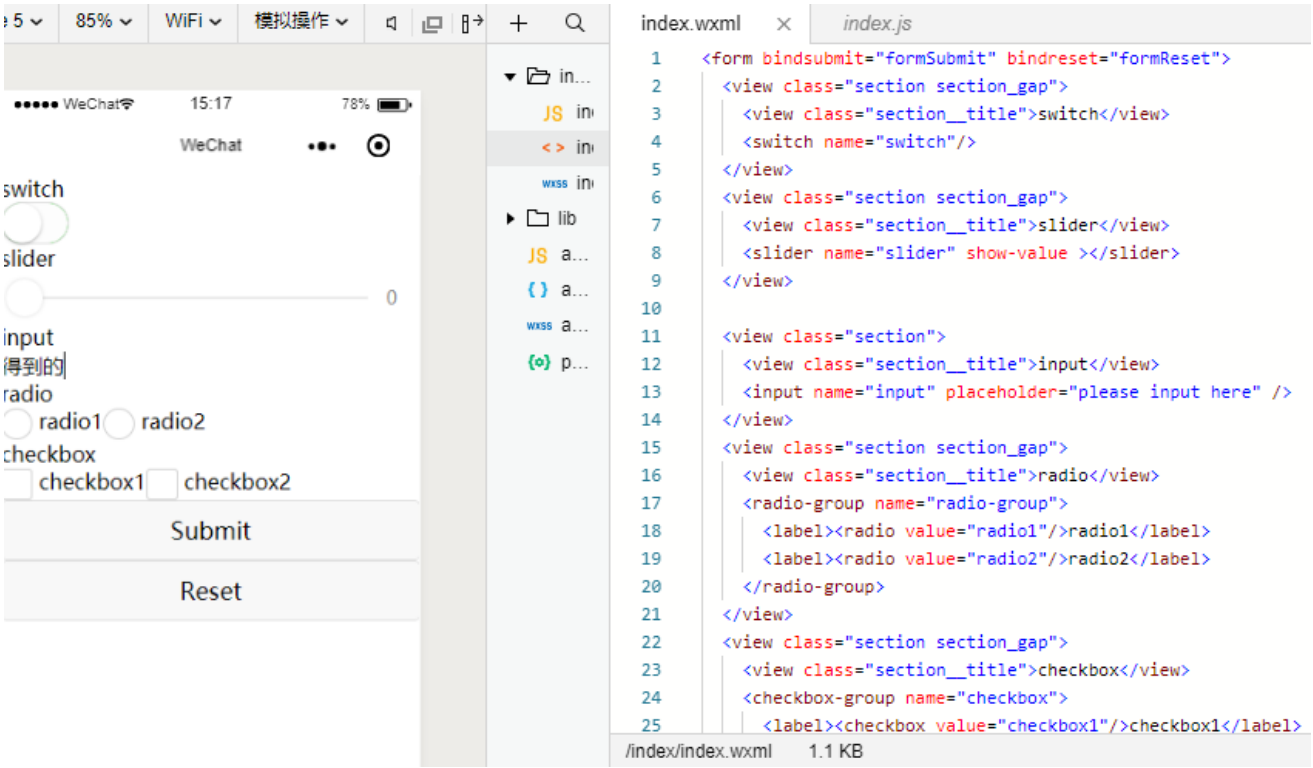
2、.js逻辑层

```
const types = ['default', 'primary', 'warn']
const pageObject = {
  data: {
    defaultSize: 'default',
    primarySize: 'default',
    warnSize: 'default',
    disabled: false,
    plain: false,
```

```
    loading: false
  },
  //以下是自定义事件
  setDisabled(e) {
    this.setData({
      disabled: !this.data.disabled
    })
  },
  setPlain(e) {
    this.setData({
      plain: !this.data.plain
    })
  },
  setLoading(e) {
    this.setData({
      loading: !this.data.loading
    })
  },
  onGotUserInfo(e) {
    console.log(e.detail.errMsg)
    console.log(e.detail.userInfo)
    console.log(e.detail.rawData)
  },
}
//这个不清楚【没有深究】
for (let i = 0; i < types.length; ++i) {
  (function (type) {
    pageObject[type] = function (e) {
      const key = type + 'Size'
      const changedData = {}
      changedData[key] =
        this.data[key] === 'default' ? 'mini' : 'default'
      this.setData(changedData)
    }
  }(types[i]))
}

Page(pageObject)
```

20、表单



需要记住一些标签，

1、switch标签

slide标签

input标签

21、picker重要组件标签，从底部弹起的滚动选择器，现支持五种选择器，通过mode来区分，分别是普通选择器，多列选择器，时间选择器，日期选择器，省市区选择器，默认是普通选择器。

1、.wxml

（1）、value的值表示选择了 range 中的第几个（下标从 0 开始）

（2）、range mode为 selector 或 multiSelector 时，range 有效，（我的理解是，当底部弹起时，有range的话才会有数据在谈起的框中）

【以下是没有range的样子】


```
<view>
 <picker bindchange="change" value="{{index}}" range="{{array}}">
  <view>
   当前的数据显示是：{{array[index]}}
  </view>
 </picker>
</view>
```

2、.js

```
// pages/details/details.js
Page({

 /**
  * 页面的初始数据
  */
 data: {
  array:["中国","美国","日本","法国"],
  shuju:[
   {
    idx:0,
    name:'中国'
   },
```

```
      {
        idx: 1,
        name: '美国'
      },
      {
        idx: 2,
        name: '日本'
      },
      {
        idx: 3,
        name: '法国'
      }
    ]
  },
  index:0,//先设置一个下边作为没有选中时的默认
  change(e){
    this.setData({
      index: e.detail.value//选中时改变数据下标
    })
  }
})
```

22、picker-view跟picker不同的就是，不会底部弹出，

23、参数传递

```
<text>数据显示：{{username}}</text>
<view>
<button bindtap="change" data-username="{{username}}">改变数据</button>

</view>
```

1、此时当点击button时，"username"这个值就会被传递，在另一个文件里边，也就是change(){}这个函数里边就可以获取到参数了，(只要是时间处理函数都会有个e，所以在change(e){}函数里边呢传个e，打印出来你会看到很多东西，代用不同的值就会获取到传递的参数值)

2、.js页面

```
 change(e){
 console.log(e);//此时就会获取到很多值，
   this.setData({
     username:"老谢"
   })
 }
```

24、小程序里边的一些指令

1、wx:for

```
(1).js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    username:"商品详情",
    tabs:["小明","小红","小华"]
  },
  change(){
    this.setData({
      username:"老谢"
    })
  }
})
(2)、.wxml
<view wx:for="{{tabs}}">
    {{item}}————{{index}}
</view>
此时以上的数组就会遍历到页面当中

(3) 以上的指令也可以进行自定义
<view wx:for="{{tabs}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName}}
</view>

(4)以上还可以进行参数传递
<view wx:for="{{tabs}}" wx:for-index="idx" wx:for-item="itemName" id="{{itemName}}">
  {{idx}}: {{itemName}}
</view>

只有一个的话，可以直接使用以上方法进行传递，id="{{itemName}}"，但是多个的话，建议使用自定义方法
```

(5)自定义参数传递
<view wx:for="{{tabs}}" wx:for-index="idx" wx:for-item="itemName" data-username="{{itemName}}" data-index="{{idx}}" bindtop="reverName">
 {{idx}}: {{itemName}}
</view>

.js:只要是时间都可以传递一个参数e，表示事件
reverName(e){
  let username=e.currentTarget.dataset.username;//此时只要执行以上这个方法，就能获取到对应的参数，就可以对参数进行相应的处理；
  let index=e.currentTarget.dataset.idx;
  username=username.split("").reverse().json('');//此时就会名字颠倒，然后在改变以上的数据
  let arr=[...this.data.tabs];//获取上边数组的数值
  arr[index]=username;
  this.setData({
    tabs:arr
  })
}

(6)路由（api）wx.navigateBack
.wxml:
<button bindtap="back"></button>

.js:
back(){
  wx:redirectTo({
    url:'/pages/index/index?us=123',
    success(){//成功的回调
      console.log("success"),
    },
    fail(){
      console.log("fail");//失败的回调
    }
  })
}

25、在不同的页面获取到参数

const app=getApp();
不管在哪个页面获取，首先需要加上这一句，表示通过全局函数 getApp() 可以获取全局的应用实例；

1、实例伺候

(1)、提供数据的页面（这是全局配置的app.js）

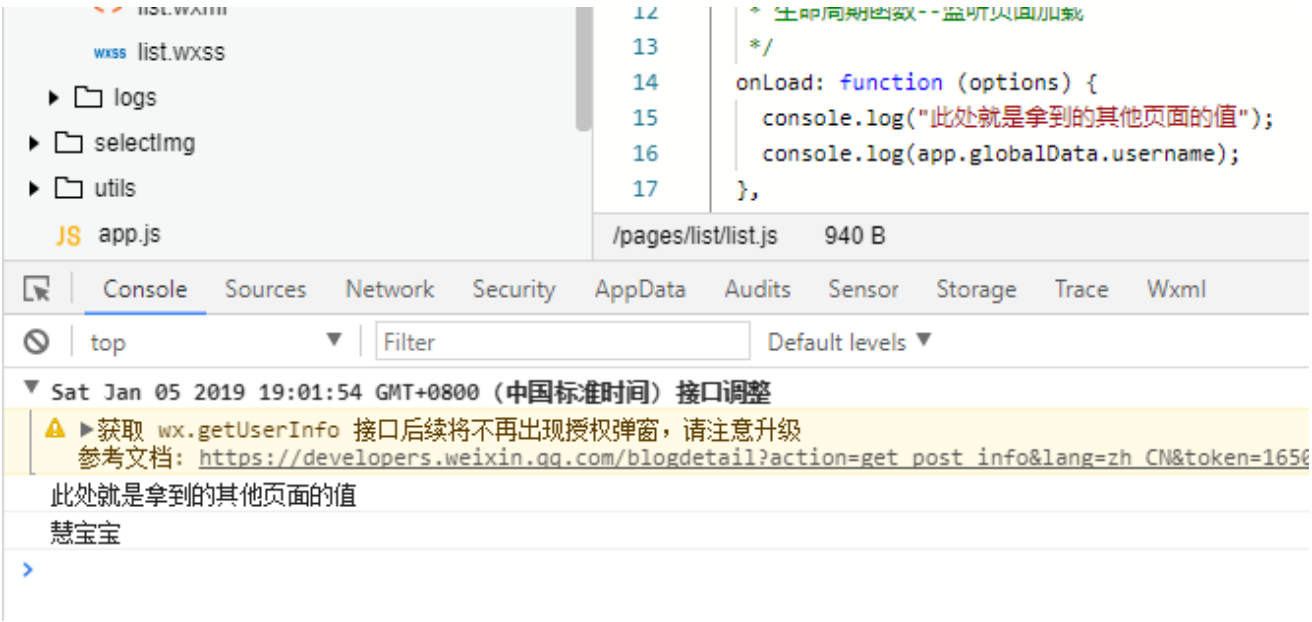
```
App({
 globalData: {
  username:"慧宝宝"；//这就是我设置的参数值
 }
})
```

（2）、获取值得文件

```
const app=getApp();
Page({
 onLoad: function (options) {
  console.log("此处就是拿到的其他页面的值");
  console.log(app.globalData.username);
 },
})
```

注意：（app.globalData.username）globalData是提供数据的方法，username是数据名字

（3）、实例图片



26、模块化

1、可以将一些公共的代码抽离成为一个单独的 js 文件，作为一个模块。模块只有通过 `module.exports` 或者 `exports` 才能对外暴露接口。

（1）、暴露公有文件

```
// common.js公有文件
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}

module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye

module.exports和exports都可以暴露接口
```

（2）、应用公有文件

```
const common = require('common.js')
Page({
  helloMINA() {
    common.sayHello('MINA')
  },
  goodbyeMINA() {
    common.sayGoodbye('MINA')
  }
})
```

（3）、公有文件通常在util.js工具里边暴露接口

```
实例：
module.exports = {
  formatTime: formatTime;//处理时间格式的接口
}
```

27、对象

（1）、也可以用扩展运算符 `...` 来将一个对象展开

```
<template is="objectCombine" data="{{...obj1, ...obj2, e: 5}}"></template>
```

（2）、**注意：** 花括号和引号之间如果有空格，将最终被解析成为字符串

```
<view wx:for="{{[1,2,3]}} ">{{item}}</view>
```

28、列表渲染

一、

（1）、wx:for默认数组的当前项的下标变量名默认为 `index`，数组当前项的变量名默认为 `item`

使用wx:for的时候，记得加上wx:key，wx:key表示唯一的标识符，wx:key的变量可以直接使用变量，就可以不需要使用item.index等，直接可以使用index

```
.wxml
<view wx:for="{{array}}">{{index}}: {{item.message}}</view>

.js
Page({
  data: {
    array: [{ message: 'foo',}, {message: 'bar'}]
  }
})
```

(2)、使用 `wx:for-item` 可以指定数组当前元素的变量名，

使用 `wx:for-index` 可以指定数组当前下标的变量名：

应用场景：循环嵌套，此时就会分不清item是哪里提供的，所以，此时就应该更改指令代码

```
<view wx:for="{{array}}">{{item}}
    <view wx:for="{{array}}">
            {{item}}
    </view>
</view>
```

更改够

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName.message}}
</view>
```

二、

（1）、block wx:for

此种用法可以把block当做一个标签来使用，

```
<block wx:for="{{array}}">
    <view>{{item}}</view>
    <view>{{item}}</view>
</block>
```

三、

（1）、wx:if表示直接去掉某个节点，

在框架中，使用 `wx:if="{{condition}}"` 来判断是否需要渲染该代码块：

```
<view wx:if="{{condition}}">True</view>
```

也可以用 `wx:elif` 和 `wx:else` 来添加一个 else 块：

```
<view wx:if="{{length > 5}}">1</view>
<view wx:elif="{{length > 2}}">2</view>
<view wx:else>3</view>
```

（2）、block wx:if

因为 `wx:if` 是一个控制属性，需要将它添加到一个标签上。如果要一次性判断多个组件标签，可以使用一个 `<block/>` 标签将多个组件包装起来，并在上边使用 `wx:if` 控制属性。

```
<block wx:if="{{true}}">
  <view>view1</view>
  <view>view2</view>
</block>
```

注意： `<block/>` 并不是一个组件，它仅仅是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

（3）、 `wx:if` vs `hidden`

同时 `wx:if` 也是**惰性的**，如果在初始渲染条件为 `false`，框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下， `hidden` 就简单的多，组件始终会被渲染，只是简单的控制显示与隐藏。

一般来说， `wx:if` 有更高的切换消耗而 `hidden` 有更高的初始渲染消耗。因此，如果需要频繁切换的情景下，用 `hidden` 更好，如果在运行时条件不大可能改变则 `wx:if` 较好。

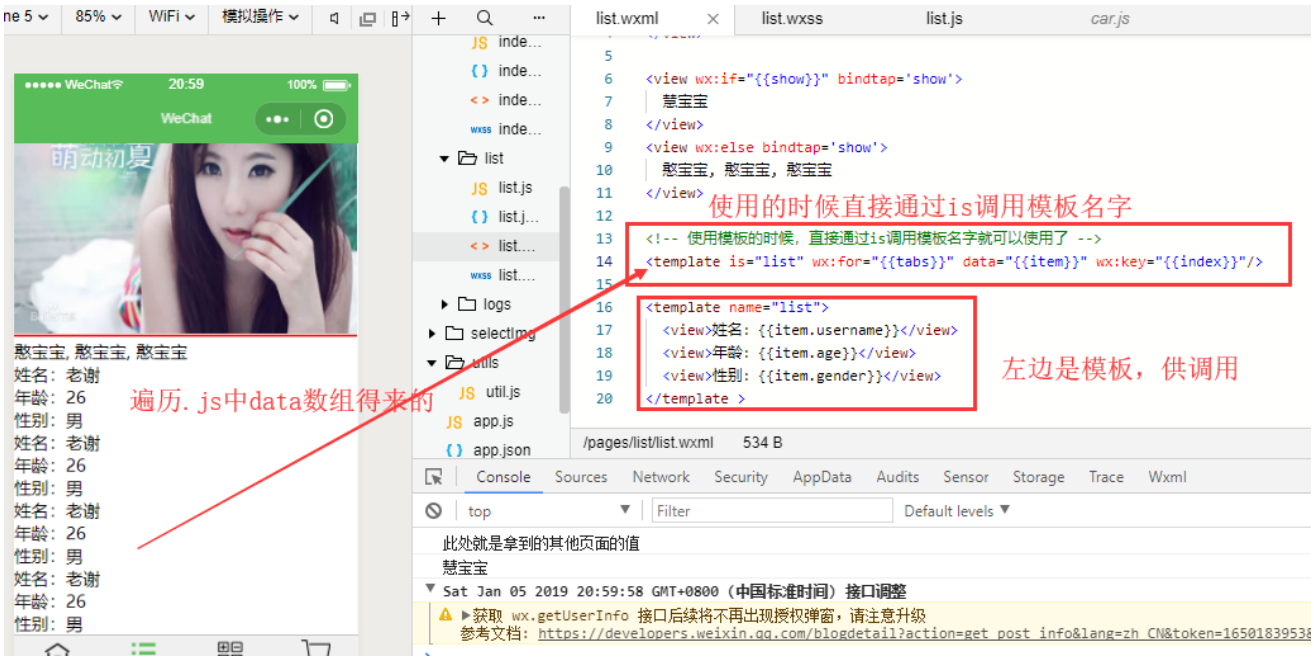
实例：

```
<view wx:if="{{show}}" bindtap='show'>
  慧宝宝
</view>
<view wx:else bindtap='show'>
  憨宝宝, 憨宝宝, 憨宝宝
</view>

<view hidden="show">
  憨宝宝, 憨宝宝, 憨宝宝
</view>

逻辑代码：
const app=getApp();
Page({
  data: {
    show:true
  },
  show(){
    this.setData({
      show:false
    })
  }
})
```

四、使用模板



1、模板使用很简洁，先设置好模板，然后在需要使用模板的地方通过<template is="模板name的名字 " wx:for="{{数组名字}} data="{{传递给模板使用的参数，通常是item}}"/>

```
.wxml:
<!-- 使用模板的时候，直接通过is调用模板名字就可以使用了 -->
<template is="list" wx:for="{{tabs}}" data="{{item}}" wx:key="{{index}}"/>

<template name="list">
  <view>姓名：{{item.username}}</view>
  <view>年龄：{{item.age}}</view>
  <view>性别：{{item.gender}}</view>
</template >

.js:
data: {
 tabs:[
    {
      username:"老谢",
      age:26,
      gender:"男"
    },
    {
      username: "老谢",
      age: 26,
      gender: "男"
    },
    {
      username: "老谢",
```

```
      age: 26,
      gender: "男"
    },
    {
      username: "老谢",
      age: 26,
      gender: "男"
    }
  ]
}
```

2、通过单独设定共有的文件夹，然后通过 引入后使用

```
（1）、供有模板
<template name="list">
  <view>姓名：{{item.username}}</view>
  <view>年龄：{{item.age}}</view>
  <view>性别：{{item.gender}}</view>
</template >

（2）、引入并使用代码
通过绝对路劲映入
<import src="/commonts/moban.wxml" />
<!-- 使用引入的模板 -->
<template is="list" wx:for="{{tabs}}" data="{{item}}" wx:key="{{index}}"/>
```

3、include，它是直接使用了模板里边的内容，通过 这样一个闭合的标签

```
模板代码：
<view>
  这是include
</view>

引用代码：
<include src="/commonts/moban.wxml" />
```

29、事件

1、点击事件：

`bindtap` 点击该组件的时候会在该页面对应的Page中找到相应的事件处理函数。

```
<view id="tapTest" data-hi="WeChat" bindtap="tapName">Click me!</view>
```

冒泡事件：

事件分为冒泡事件和非冒泡事件：

1. 冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。

WXML的冒泡事件列表：

类型	触发条件	最低版本
touchstart	手指触摸动作开始	
touchmove	手指触摸后移动	
touchcancel	手指触摸动作被打断，如来电提醒，弹窗	
touchend	手指触摸动作结束	
tap	手指触摸后马上离开	
longpress	手指触摸后，超过350ms再离开，如果指定了事件回调函数并触发了这个事件，tap事件将不被触发	1.5.0
longtap	手指触摸后，超过350ms再离开（推荐使用longpress事件代替）	
transitionend	会在 WXSS transition 或 wx.createAnimation 动画结束后触发	
animationstart	会在一个 WXSS animation 动画开始时触发	
animationiteration	会在一个 WXSS animation 一次迭代结束时触发	
animationend	会在一个 WXSS animation 动画完成时触发	
touchforcechange	在支持 3D Touch 的 iPhone 设备，重按时会触发	

2、非冒泡事件：

除上表之外的其他组件自定义事件如无特殊声明都是非冒泡事件，如

的submit事件， **的input事件，的scroll事件，(详见各个组件)**

3、事件冒泡：

事件绑定和冒泡

事件绑定的写法同组件的属性，以 key、value 的形式。

- key 以 `bind` 或 `catch` 开头，然后跟上事件的类型，如 `bindtap` 、 `catchtouchstart` 。自基础库版本 1.5.0 起，在非原生组件中， `bind` 和 `catch` 后可以紧跟一个冒号，其含义不变，如 `bind:tap` 、 `catch:touchstart` 。
- value 是一个字符串，需要在对应的 Page 中定义同名的函数。不然当触发事件的时候会报错。

`bind` 事件绑定不会阻止冒泡事件向上冒泡， `catch` 事件绑定可以阻止冒泡事件向上冒泡。

如在下边这个例子中，点击 inner view 会先后调用 `handleTap3` 和 `handleTap2` (因为tap事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，不再向父节点传递)，点击 middle view 会触发 `handleTap2` ，点击 outer view 会触发 `handleTap1` 。

```
<view id="outer" bindtap="handleTap1" data-type="outerView">
//简单的事件传递参数，使用的是id（id="outer"）如果参数是很多，多个，尽量使用data-开头，（data-type="outerView"），然后在逻辑层通过  e.currentTarget.来获取相应的参数。
  outer view
  <view id="middle" catchtap="handleTap2">
    middle view
    <view id="inner" bindtap="handleTap3">inner view</view>
  </view>
</view>
```

4、事件对象：

dataset

在组件中可以定义数据，这些数据将会通过事件传递给 SERVICE。 书写方式： 以 `data-` 开头，多个单词由连字符 `-` 链接，不能有大写(大写会自动转成小写)如 `data-element-type` ，最终在 event.currentTarget.dataset 中会将连字符转成驼峰 `elementType` 。

示例：

```
<view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap">
  DataSet Test
</view>

Page({
  bindViewTap(event) {
    event.currentTarget.dataset.alphaBeta === 1 // - 会转为驼峰写法
    event.currentTarget.dataset.alphabeta === 2 // 大写会转为小写
  }
})
```

4、事件冒泡：

(1)、bind:tap使用这个事件时会发生事件冒泡

(2)、为了阻止事件冒泡，可以改用catch:tap此时就不会发生事件冒泡了

(3)、在使用显示隐藏的时候，尽量使用hidden，可以在里边做一些相应的计算，

```
<view hidden="{{show!=index}}"></view>
像上面这种可以做一些简单的计算然后实现功能的显示隐藏，还可以做三元运算
```

30、事件补充

（1）、当逻辑层需要使用到视图层的数据时，那么必须使用到事件，在使用事件获取参数时，尽量使用

currentTarget获取参数，因为事件会存在事件冒泡，可能户出现错误。

(2)、要想阻止事件冒泡，就用catch:开头，不能使用原生js的阻止事件冒泡的方法，如果使用了，不但不能阻止，反而还是报错

31、事件捕获

(1)、如果使用了bind:开头来的事件，那么会存在事件冒泡，一般是从里到外执行，如果某一次你想让某一层先执行，那么就要使用到事件捕获。事件捕获，就是在在想要先实行的事件前面添加capture-

```
<view
  id="outer"
  bind:touchstart="handleTap1"
  capture-bind:touchstart="handleTap2"//在这里就使用了捕获，此时就会先执行这一层
>
  outer view
  <view
    id="inner"
    bind:touchstart="handleTap3"
    capture-bind:touchstart="handleTap4"
  >
    inner view
  </view>
</view>
```

(2)、在原生js里边使用的事件捕获，利用是是设置监听函数

addEventListener(type,handler,iscapture)

32、样式

(1)、样式导入

使用`@import`语句可以导入外联样式表，`@import`后跟需要导入的外联样式表的相对路径，用`;`表示语句结束。

```
@import "common.wxss";
.middle-p {
  padding:15px;
}
```

33、小程序页面之间的参数传递的方法

```
https://blog.csdn.net/yzi_angel/article/details/80568411
```

34、使用小程序框架的方法

（1）、一般在github上面下载，下载之后放在干净的文件夹里边，之后打开开发工具，在开发发工具里边选择（项目--->新建项目，在项目目录哪点选择刚才下载的东西，之后选择文件里边的dist文件，点击确定，在选择appid，再给该项目取个名字，之后打开就是小程序的ui的界面了）

35、导航

```
<view class="btn-area">
  <navigator url="/navigate/navigate?title=id" hover-class="navigator-hover">跳转到新页面</navigator>
  <navigator url="../redirect/redirect?title=redirect" open-type="redirect" hover-class="other-navigator-hover">在当前页打开</navigator>
</view>
```

在跳转的页面这样获取

```
Page({
  onLoad: function (options) {
    console.log(options);
    console.log("此时就能获取到其他.wxml页面传递过来的参数");
    this.setData({
      title: options.title//其他.wxml页面传递过来的数据
    })
  }
})
```

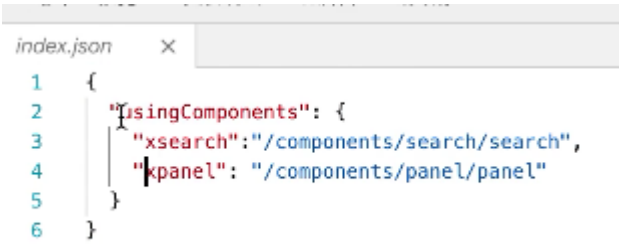
(1)、笔记

open-type：跳转方式

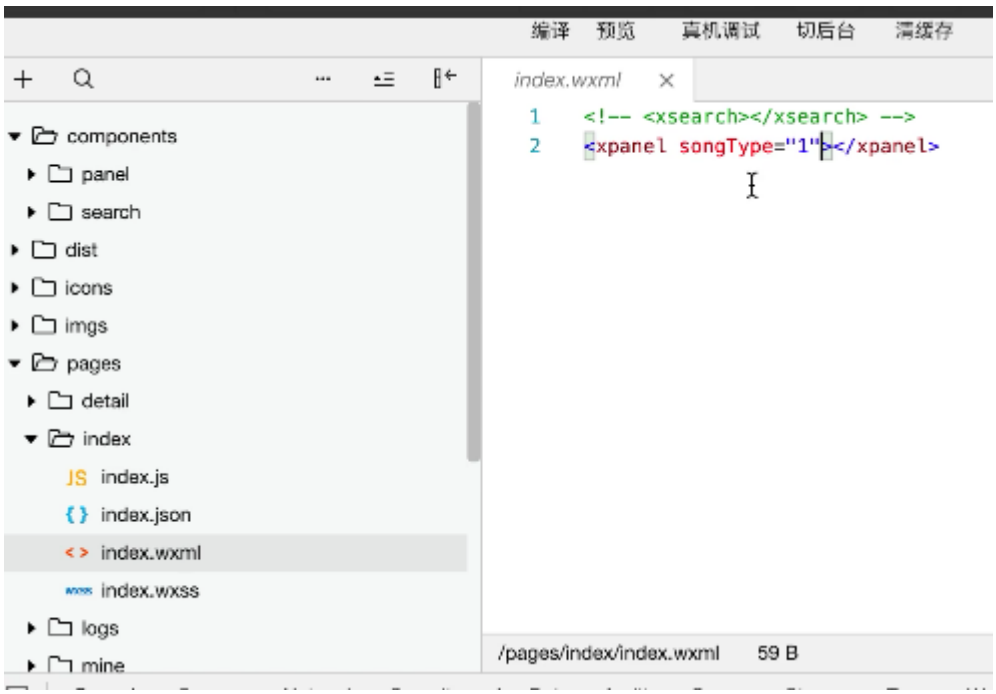
hover-class：点击时的样式类名

36、小程序组件

(1)、

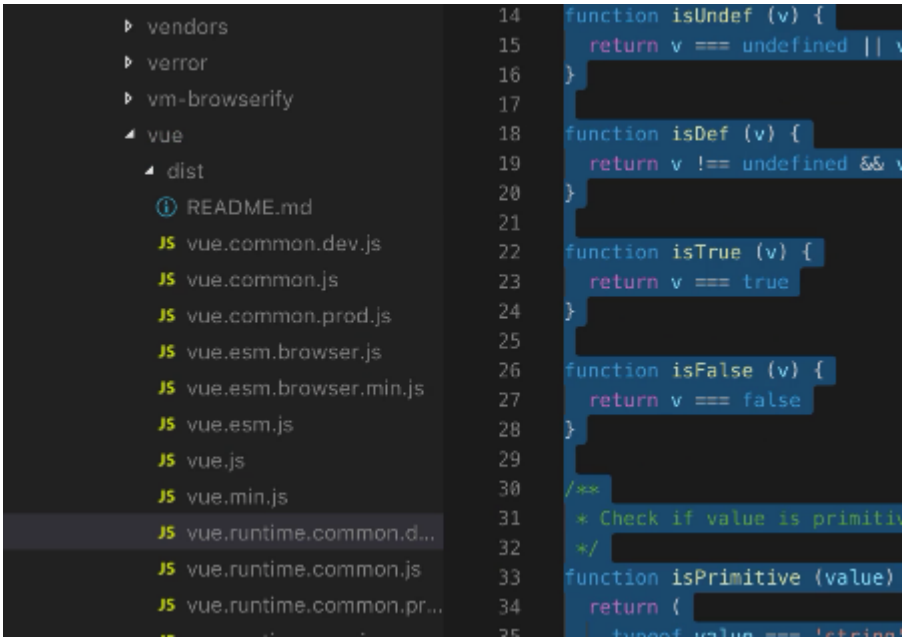


（2）、使用组件

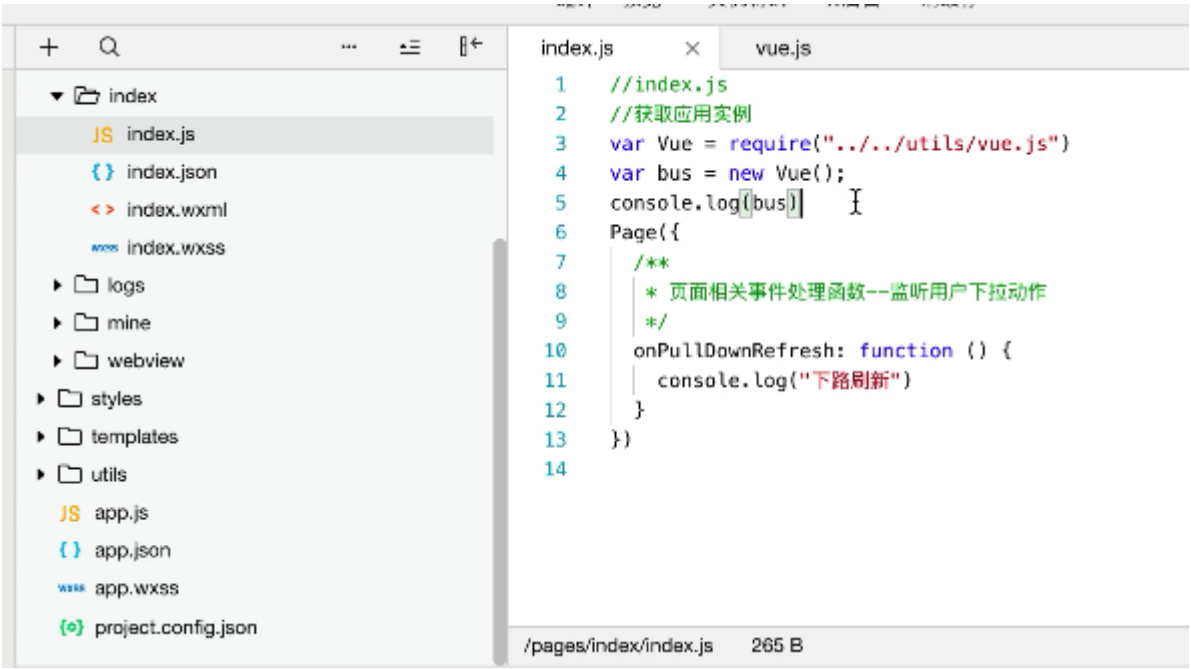


37、组件间传递参数

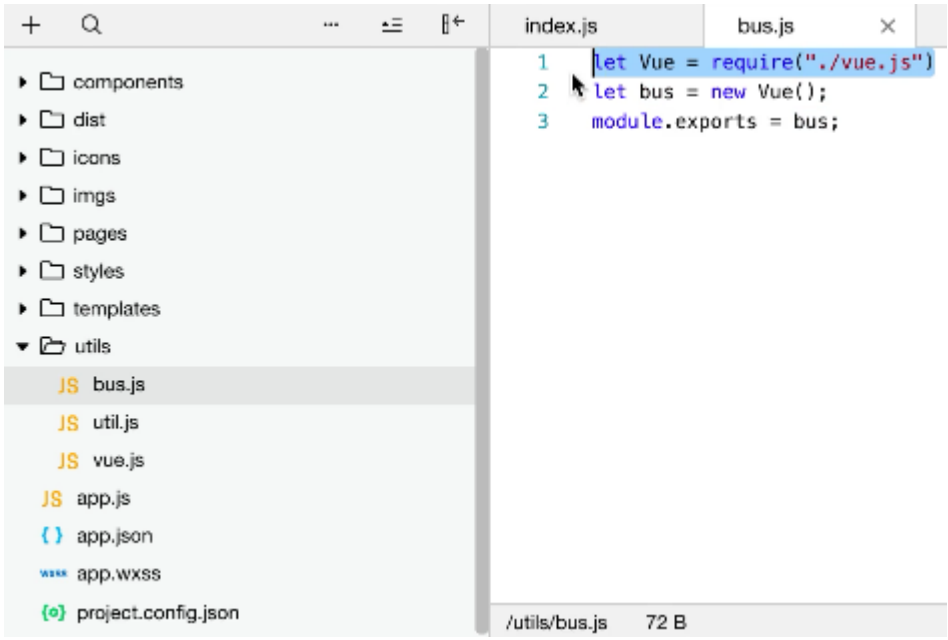
（1）、在utils里边建立vue.js，然后从脚手架里边复制这份代码进来，然后在主要的组件里边引入



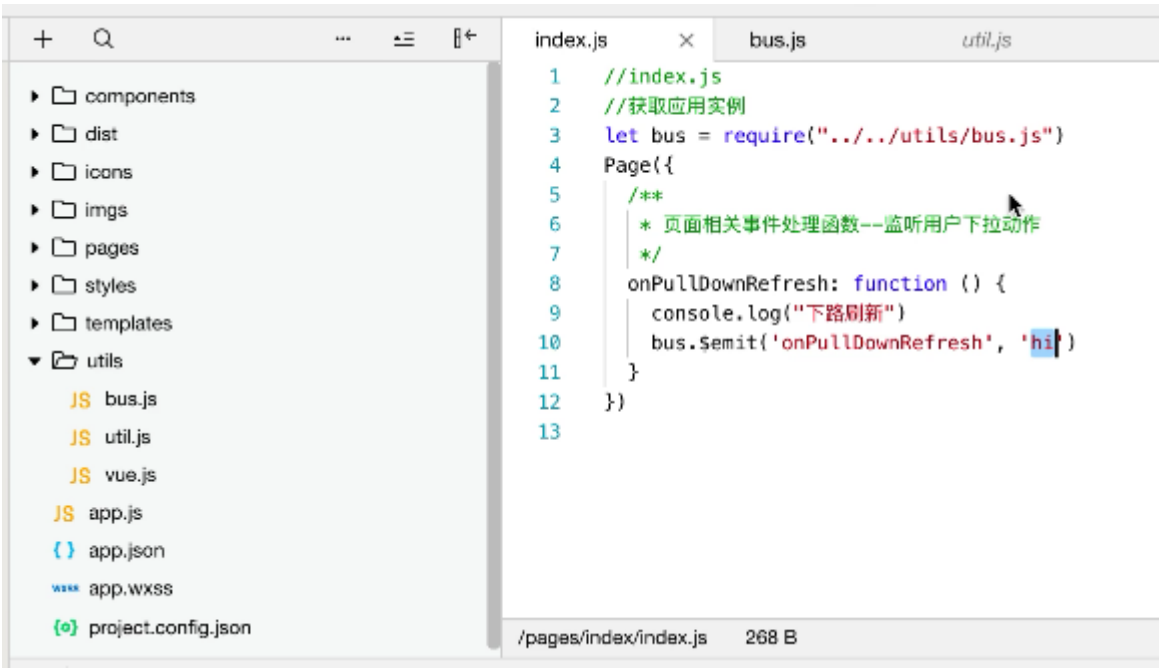
（2）、在需要的组件里边引入



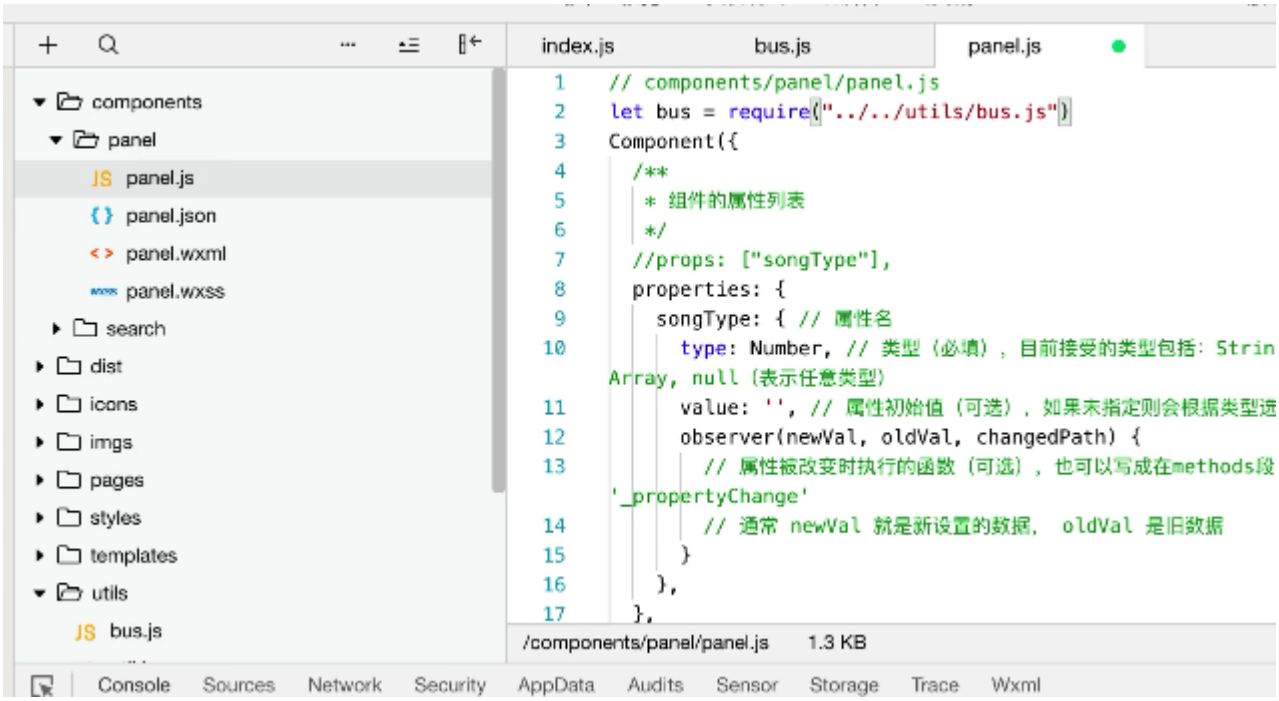
（3）、在以上的方法中删除刚刚的代码，然后在utils里边新建一个bus.js的文件，需要暴露接口



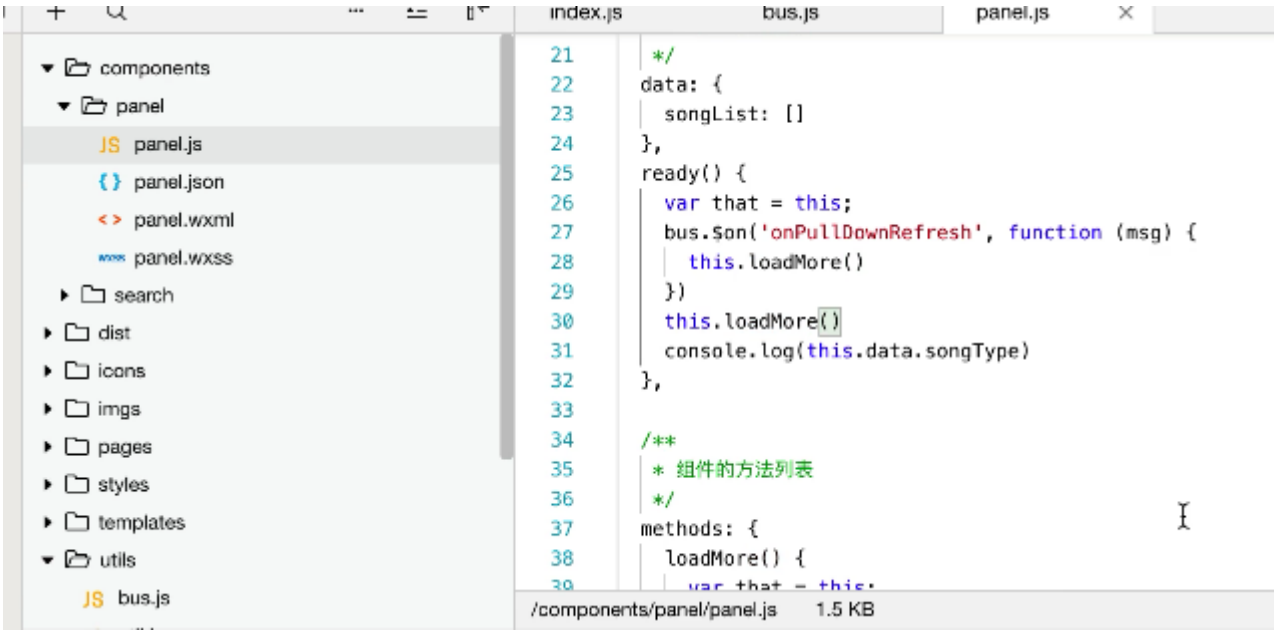
(4)、在刷新页面引入，事件总成



(5)、子组件先 引入



（6）、子组件在封装ajax请求，然后复用，在入场是就使用一次，触发事件时有发起一次请求



（7）、相关网页知识链接

https://blog.csdn.net/hope93/article/details/80803447

38、本地缓存的方法/例子

（1）、Storage

父页:
Page({

/**
* 页面的初始数据

```
  */
  data: {
    str:"小程序初次使用的数据在这"  //要存储的数据
  },
  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {
    let that=this;
    wx.setStorage({
      key: 'str',     //存储时的键值
      data: that.data.str//存储时的数据
    })
  }
})
```

```
组件:
Page({
  /**
   * 页面的初始数据
   */
  data: {
    shuju:''   //定义一个参数，用于成功后的存储值
  },

  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {
    const that = this
    wx.getStorage({
      key: 'str',     //本地存储的那个键值
      success(res) {   //成功后的获取数据
        that.setData({
          shuju: res.data
        })
      }
    })
  }
})
```

39、音乐笔记补充

（1）、组件之间传递参数

1. 在utils里边新建vue.js，然后在脚手架里边拿取vue的数据代码，放入该文件，
2. 在utils里边在新建一个bus.js文件，把vue.js文件在引入，同时暴露该文件，以至于在其他页面都能使用到

```
var Vue = require("./vue.js");
var bus = new Vue();
module.exports = bus;
// 暴露以上的东西
```

3. 在需要使用的页面里边引入bus文件

```
var bus = require("../../utils/bus.js");
Page({
  data: {
  },
  onPullDownRefresh: function () {
    // 已经触发下拉事件，此时通知panel页面，承载着你的页面已经触发，在ready（）里边添加
    bus.$emit("onPullDownRefresh","hi");
  },
  /**
   * 页面上拉触底事件的处理函数
   */
  onReachBottom: function () {
    console.log("触到底啦");
    bus.$emit("onReachBottom", "hi");
  },
})
```

4.在组件里边，（只有监听，没有数据传递）

```
var bus = require("../../utils/bus.js");
Component({
  properties: {
    songType: {
      type: Number,
      value: '',
      observer(newVal, oldVal, changedPath) {}
    }
  },
  /**
   * 页面的初始数据
   */
  data: {
    songList: [],
    offset: -1
  },
  ready() {
    var that = this;
    //一下是重点，父页面发起了通知，这里应当做相应的处理了
    bus.$on("onPullDownRefresh",(msg)=> {
      this.loadMore("onPullDownRefresh");//当下拉刷新的时候，继续触发一次请求
    }),
    bus.$on("onReachBottom", (msg) => {
```

```
      this.loadMore("onReachBottom");//当下拉舒心的时候，继续触发一次请求
    }),
    this.loadMore();//一进场触发一次请求
  },
  // 组件的方法列表
  methods: {
    loadMore(TypeX) {
      console.log(TypeX)
      var that = this;
      wx.request({
        url: 'http://tingapi.ting.baidu.com/v1/restserver/ting', // 仅为示例，并非真实的接口地址
        data: {
          method: 'baidu.ting.billboard.billList',
          type: that.data.songType, //返回歌曲类型
          size: 10, //每页返回的数据
          offset: ++that.data.offset //返回页数
        },
        header: {
          'content-type': 'application/json' // 默认值
        },
        success(res) {
          wx.stopPullDownRefresh();//下拉结束后拉的部分弹回来
            that.setData({
              songList: TypeX === "onPullDownRefresh" ? res.data.song_list.concat(that.data.song_list) :
that.data.songList.concat(res.data.song_list)
            })
        }
      })
    }
  }
})
```

5、

（1）、通过传递的参数，在逻辑层使用以下方法获取

```
onLoad: function (options) {
    console.log(options);
    console.log("这就是<navigator>标签传递过来的参数")
  },
```