微信小程序基础

开发工具的安装

直接到官网进行下载即可，网址如下

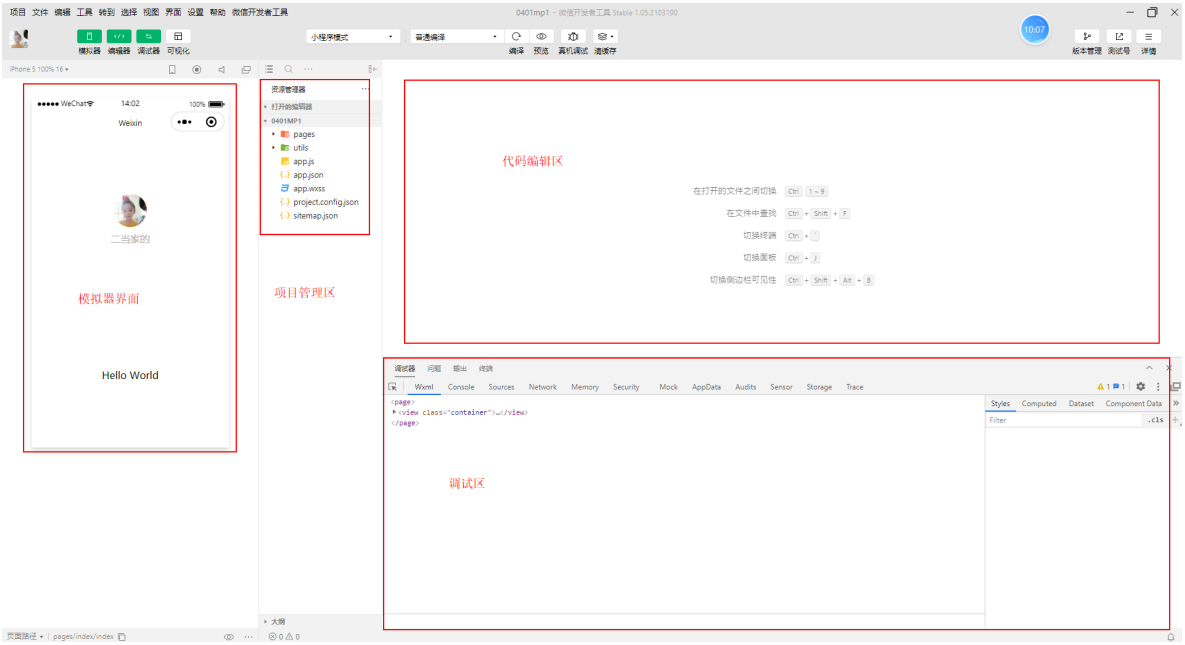
https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html

微信小程序项目创建

小程序的开发工具需要与开发者的微信进行扫码绑定，完成以后将会进入下面的界面

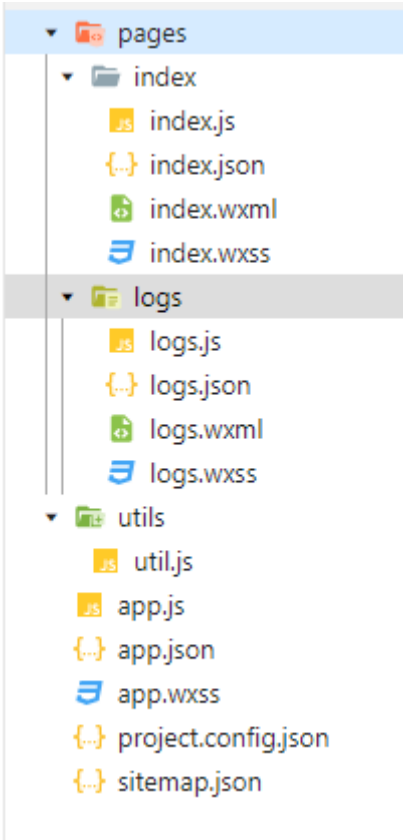
我们目前开发的是小程序，所以我们需要对这个小程序项目进行新建

我们现在是使用测试号来创建了一个微信小程序，这个时候会打开小程序的开发工具，界面如下



小程序的项目结构

要弄清楚小程序，必先弄清楚小程序开发工具创建项目以后的结构是什么样式



1. `pages`目录用于存放所有的页面，相当于在vue里面的`views`目录
2. `index`目录相当于一个具体的页面，它内部有4个文件
3. `index.wxml`相当于vue文件中的`<template>`这个部分，用于存放标签代码，相当于`html`
4. `index.wxss`相于vue文件中`<style scoped>`这个部分，用于存放样式代码，相当于`css`
5. `index.js`相当于vue文件中的`<script>`部分，用于存放脚本代码
6. `index.json`用于存放相关的配置信息，如这个页面使用了某些组件或某些插件
7. `utils`文件夹就当我们之前的文件夹一样，存放一些第三方的工具类
8. `app.js`整个小程序的**入口**文件都在这里，相当于vue当中的`App.vue`这个文件里面的脚本代码
9. `app.wxss`整个小程序的全局样式都在这里，如果在这里写了代码，所有的页面都可以使用这里的样式

10. `app.json`整个小程序的配置信息都在这里，这里面配置的`titleBar`，`tabBar`以及页面等相关信息
11. `project.config.json`这个是小程序项目的配置信息，这里面记录的小程序的开发人员id,小程序的id
12. `sitemap.json`是小程序后期做分包加载的时候使用到了，因为小程序代码总体不能超过2M，但是如果项目过大，这个时候可以考虑分包开发

小程序的配置

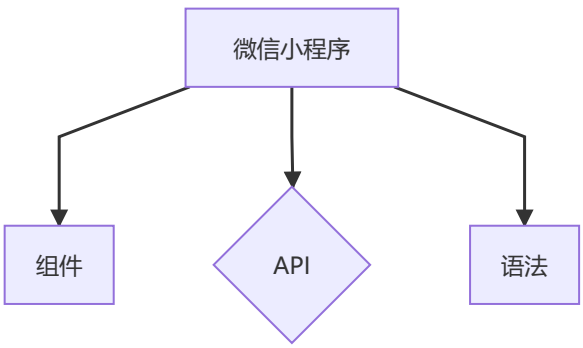
小程序的整个配置信息都在`app.json`这个文件里面，我们可以打开这个文件研究一下

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "Weixin",
    "navigationBarTextStyle":"black"
  },
  "style": "v2",
  "sitemapLocation": "sitemap.json"
}
```

- `pages`代表当前小程序有多少个页面，如果后期新增了页面，也要在这里进行添加；而删除了某些页面以后，也需要在这里进行删除

  同时要注意，配置在第一个的就是默认启动页
- `window`整个小程序的页面的配置信息
  - `navigationBarBackgroundColor`用于设置导航栏的背景颜色
  - `navigationBarTitleText`用于设置小程序的默认标题
  - `navigationBarTextStyle`导航栏文件的颜色，只有`white`与`black`
  - `backgroundTextStyle`小程序背景的文字样式设置，只有`light`与`dark`

小程序的技术点分类



小程序的组件

小程序是基于组件化开发的，所以它没有DOM，更没有BOM。也就是没有`document`，更没有`window`，它只有ES，并且它没有`HTML`标签，所以的一个都是组件（也就是虚拟DOM）virtual DOM

其中有几个重要的组件与大家说一下

小程序组件名	HTML标签名	备注
`<view></view>`	`<div></div>`	块级元素div
`<text></text>`	`<span></span>`	行内元素span
`<image />`	`<img />`	图片标签
`<button></button>`	`<button></button>`	按钮
`<navigator></navigator>`	`<a href="#"></a>`	链接标签

我们上面列举的只是一些简单的常用的组件，还有一些其它的组件，请参考官网开发者文档

https://developers.weixin.qq.com/miniprogram/dev/component/

小程序的语法

小程序的框架语法主要点在js上面，所以我们打开一个页面的js文件，我们将会看到下面的内容

```
// pages/bg2/bg2.js
Page({

  /**
   * 页面的初始数据
   */
  data: {

  },

  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {

  },

  /**
   * 生命周期函数--监听页面初次渲染完成
   */
  onReady: function () {

  },

  /**
   * 生命周期函数--监听页面显示
   */
  onShow: function () {

  },

  /**
   * 生命周期函数--监听页面隐藏
   */
  onHide: function () {
```

```
  },

  /**
   * 生命周期函数--监听页面卸载
   */
  onUnload: function () {

  },

  /**
   * 页面相关事件处理函数--监听用户下拉动作
   */
  onPullDownRefresh: function () {

  },

  /**
   * 页面上拉触底事件的处理函数
   */
  onReachBottom: function () {

  },

  /**
   * 用户点击右上角分享
   */
  onShareAppMessage: function () {

  }
})
```

小程序框架采用的是`MVVM`的开发模式，所以这一套与之前的`vue`是差不多的，但是有部分区别，我们列表一下

1. `data`这个与之前所学习的vue里面的`data`是一样的，这是页面上面所需要使用数据都会在这里
2. `onLoad`是小程序页面的生命周期的钩子函数，相当于vue组件生命周期中的`created()`
3. `onReady`也是小程序页面生命周期的钩子函数，相当于vue组件生命周期中的`beforeMount()`
4. `onShow`小程序生命周期的钩子函数，相当于vue组件生命周期中的`mounted()`
5. `onHide`小程序的生命周期钩子函数。这对比较特殊，相当于在vue组件的生命周期上面加了`<keep-alive>`以后的`deactivated`
6. `onUnload`小程序的生命周期钩子函数，相当于vue中的钩子函数`destoryd()`
7. `onPullDownRefresh`小程序的页面下拉刷新函数，如果页面配置了**可下拉刷新**，则页面在下拉刷新以后会调用这个函数
8. `onReachBottom`如果小程序的页面配置了**可上拉加载**，则页面在触底以后会自动的调用这个函数
9. `onShareAppMessage`小程序右上解的分享按钮点击以后会调用这个方法

小程序的数据渲染

小程序的渲染分为多种多样，所以它也可以参考之前所学习的vue里面来进行

大体上来说，小程序的页面的数据渲染分为普通渲染，条件渲染，列表渲染

普通渲染

这种写法与之前vue的写法是一样的

```
data: {
   userName:"徐大江"
 },
```

在页面上面则可以通过下面的方式进行渲染

```
<view>
  {{userName}}
</view>
<view>
  {{userName}}
</view>
<view>
  {{"你好啊"+userName}}
</view>
<view>
  {{1+1}}
</view>
<view>
  {{11>2?"大江":"小江"}}
</view>
```

上面的几种写法都是可行的，与vue保持一致

条件渲染

这个东西在vue里面使用的是`v-if`来进行的，现在我们小程序也有它特殊的语法`wx:if`

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

我们现在通过自己的数据来测试一下

```
data: {
    flag:false,
        userName:"标哥哥"
},
```

现在去渲染

```
<view>
  <text wx:if="{{flag}}">{{userName}}</text>
  <text wx:else>帅气哥</text>
</view>
```

注意事项：在`wx:if`的后面是要加`{{}}`，这里我就提前说明一点，小程序的视图界面（`wxml`里面）如果出现了`{{}}`则这个一定是`JS`代码

列表渲染

这里的列表渲染使用的是`wx:for`，使用过程与`vue`当中的`v-for`一模一样

```
  data: {
    flag:false,
    userName:"标哥哥",
    stuList:[
      "张珊",
      "李四",
      "王五",
      "赵六"
    ]
  },
```

渲染的页界面如下

```
<view>
  <text wx:for="{{stuList}}" wx:for-item="item" wx:for-index="index"
wx:key="index">
    {{item}}-----{{index}}
  </text>
</view>
<view>
  <text wx:for="{{stuList}}" wx:key="index">
    {{item}}-----{{index}}
  </text>
</view>
```

1. `wx:for-item` 代表遍历的每一次，默认叫 `item`
2. `wx:for-index` 代表遍历的这一项的索引，默认叫 `index`
3. `wx:key` 就是以前在vue当中学习的 `key`，作用也是一样的

小程序的事件

小程序里面的事件绑写与之前VUE中的DOM事件绑定是区别

```
<!--小程序绑定方法的时候不能加括号，只能是方法名 -->
<button bindtap="sayHello">这是一个按钮</button>
```

则事件方法应该写在下面这个地方

```
// pages/bg4/bg4.js
Page({

  /**
   * 页面的初始数据
   */
  data: {
    userName:"张珊同学"
  },

  // 小程序会默认向事件方法里面注入一个参数event
  sayHello(event){
    console.log("hello world");
     // 这个事件对象并不是DOM的事件对象
    console.log(event);
  }
```

```
})
```

小程序的默认事件绑定是通过`bind`来进行的，事件名称与之前的DOM中的事件名称了不一样，在事件调用方法的时候也要注意，它没有括号，只有方法 名

1. 绑事事件使用`bind`
2. 事件名称与DOM不一样
3. 事件写在js文件的Page这个对象下面
4. 事件在调用方法的时候不要加括号，只有方法名
5. 小程序在调用事件方法的时候默认会向这个方法注入一个`event`做为参数

小程序的事件传参

正是因为小程序的事件在绑定的时候是方法名，不能加括号，所以怎么去传递参数呢?

```
<button bindtap="aaa" data-stu-name="张珊">这是方法aaa</button>
```

我们可以看到在组件button上面我们添加了一个自定义的属性`data-stu-name`后面跟了一个我们需要传递的值，而我们又知道小程序在调用事件方法的时候会向当前的方法里面传递一个参数`event`，所以最终我们可以通过`event`去得到这个参数

```
aaa(event){
    console.log(event);
    console.log(event.currentTarget.dataset.stuName);
}
```

小程序的事件传递

在普通的DOM里面，我们的事件是可以进行传递的，有冒泡行为与捕获行为，小程序使用的是冒泡行为，所以当内部的元素触发事件以后会冒泡到外边来，这个时候怎么要阻止事件的传播呢（取消事件冒泡）

```
<view class="box" bindtap="a2">
  <button type="default" catchtap="a1">按钮事件</button>
</view>
```

```
a1(event){
    console.log("我是a1的按钮的方法");
},
a2(event){
   console.log("我是盒子a2的方法");
}
```

这个时候的界面如下

当我们点击里面的按钮的时候，事件并不会冒泡到外边去，因为我们使用的是`catchtap`来进行的事件绑定，`catch`进行的小程序事件绑定是没有传播行为的

小程序的数据状态

小程序执行的是单向数据绑定,但是可以重新主动渲染页面，但是并不会自动的渲染页面【这一点类似于react】

我们现在可以通过一个案例去实现掉这个点

```
<view>
  姓名：{{userName}}
</view>
<view>
  年龄：{{age}}
</view>
<button type="default" bindtap="changeUserName">改变userName</button>
```

JS代码

```
// pages/bg5/bg5.js
Page({

  /**
   * 页面的初始数据
   */
  data: {
    userName:"张珊",
    age:18
  },
  changeUserName(event){
    // 首先看一下，能不能获取到数据
    /**
     * this.data.userName = "李四小坏蛋";
     * console.log(this.data.userName);
     */
    //上面的赋值方式并不会改变数据，如果想改变数据以后主动的渲染页面
    this.setData({
      userName:"李四小坏蛋"
    });
  }
})
```

代码分析:

1. 当我们去点击按钮以后，我们发现如果采用常规的`this.data.userName='李四小坏蛋'`，这个时候是会赋值成功，但是页面并不会把这个值重新渲染，所以充分的可以说明一点，这个值是执行单向绑定的
2. 当我们调用`this.setData()`这个方法去赋值的时候，这个时候值会改变，并且页面会重新渲染，这是一种主动的渲染模式
3. 当我们在调用`this.setData({userName:'李四小坏蛋'})`的时候，它只会对`userName`的值改变，而`data`内部还有一个`age`不会有任何影响

所以通过上面的一个点我们可以得到，如果你只是想赋值而不想渲染页面，则使用this.data的方式来完成，如果你想赋值完成以后重新渲染页面，则使用this.setData()的方式

复杂情况下的数据改变

对于复杂的数据情况，我们在进行`setData`操作的时候要注意，不能直接去赋值，如下数据结构

```
data: {
    userInfo: {
        userName: "张珊",
            age: 18,
    },
    teacherName: "标哥哥"
}
```

界面代码

```
<view>
  <text>
    {{userInfo.userName}}
  </text>
  <text>
    {{userInfo.age}}
  </text>
</view>

<view>
  {{teacherName}}
</view>

<button type="default" bindtap="changeUserName">改变userName</button>
```

事件方法

```
changeUserName(event) {
    //请问，怎么把userName改变成"李四"并重新渲染页面
    /*
    this.setData({
      userInfo:{
        userName:"李四"
      }
    });
    */
    //通过上面的方式赋值是有问题的，我们的age这个属性会被覆盖掉

    //第一种方式：通过Object.assign()去完成
    /*
```

```
    this.setData({
      userInfo: Object.assign(this.data.userInfo, {
        userName: "李四"
      })
    })
    */

    //第二种方式：借用于ES6语法
    this.setData({
      userInfo:{...this.data.userInfo,userName:"李四"}
    });
  }
```

小程序的样式处理

微信小程序所使用的也是`css`,只是有一些特殊情况要注意

1. 关于选择器
2. 关于单位
3. 关于背景图片

小程序采用的是wxss的标准，WXSS 具有 CSS 大部分的特性，小程序在 WXSS 也做了一些扩充和修改。所以大概点可以列举如下

1. 新增了尺寸单位。在写 CSS 样式时，开发者需要考虑到手机设备的屏幕会有不同的宽度和设备像素比，采用一些技巧来换算一些像素单位。WXSS 在底层支持新的尺寸单位 rpx ，开发者可以免去换算的烦恼，只要交给小程序底层来换算即可，由于换算采用的浮点数运算，所以运算结果会和预期结果有一点点偏差。

   rpx（responsive pixel）：可以根据屏幕宽度进行自适应。规定屏幕宽为750rpx。如在 iPhone6 上，屏幕宽度为375px，共有750个物理像素，则750rpx = 375px = 750物理像素，1rpx = 0.5px = 1物理像素。

设备	rpx换算px (屏幕宽度/750)	px换算rpx (750/屏幕宽度)
iPhone5	1rpx = 0.42px	1px = 2.34rpx
iPhone6	1rpx = 0.5px	1px = 2rpx
iPhone6 Plus	1rpx = 0.552px	1px = 1.81rpx

如果我们当前的设计稿是750px，同时我们现在是以iphone6为模拟器再开发，这个时候得到的结果如下

```
rpx = 设计稿 / 模拟器的像素点
rpx = 750 / 750
```

所以在发前的开发文件上面，我们的1px对应的就是1rpx

2. 小程序不允许使用本地图片做为背景片，在使用过程当它会报错

```
<view class="box2">
</view>
```

wxss代码

```
.box2{
  width: 400rpx;
  height: 400rpx;
  border: 1px solid black;
  /* background-image: url("/images/img1.jpg");
     小程序是自带webpack,但是小程序没有配置url-loader
  */
  background-image:
url("https://www.baidu.com/img/dong_528d34b686d4889666f77c62b9a65857.gif");
}
```

直接使用网络图片做为背景图片，这样会节省小程序的开发空间

当然还有另一种方式，把图片转换成`base64`,然后再去使用这也是可以的

当然还可以使用**子绝父相**来完成，最后再添加一个`z-index:-1`

3. 小程序支持的选择器

   小程序并不是完全支持CSS当中的所有选择器，它支持的选择器列举如下

目前支持的选择器有：

选择器	样例	样例描述
.class	.intro	选择所有拥有 class="intro" 的组件
#id	#firstname	选择拥有 id="firstname" 的组件
element	view	选择所有 view 组件
element, element	view, checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
::after	view::after	在 view 组件后边插入内容
::before	view::before	在 view 组件前边插入内容

通过上面的列表我们可以看到，小程序不支持`:hover`的伪类，但是每个组件都有一个`hover-class`的属性，你可以把它当成是`:hover`的伪类样式

注意：小程序不要使用后代选择器，更不能使用子代选择器，后期很容易出总理 ，因为小程序的界面开发全都是其于组件化开的，它里面并不是标签，所以这个组件在内部的小程序里面最终可能会生成多层结点，这样子后代关系就容易出现混乱

技巧点：

1. 在小程序的样式 里面，默认情况下，`wxss`是封闭的，它只用于当前同名的`wxml`文件 ，如`index.wxss`就只用于谁`index.wxml`
2. 小程序如果想实现全局的样式 ，则把这个样式写在`app.wxss`里在，或通过`@import`导入到这个文件里面去，如我们可以在这个文件里面通过`@import`导入`iconfont`，这样所有的页面就都可以使用字体图标

小程序的路由栈

小程序的路由栈也称之为小程序的页面栈，它对小程序里面的页面跳转提供了一些方法

1. `wx.navigateTo`跳转到某个页面，这个页面**在小程序的页面栈里面直接入栈**，之前的页面并没有出栈，同时这个页面会自动的在标题栏添加一个返回按钮，执行返回操作【保留当前页面，跳转到应用内的某个页面】
2. `wx.navigateBack`，这个页面在页面栈中直接中出，相当于返回到了上级页面

3. `wx.redirectTo`，这个是重定向到小程序的某一个页面，相当于Vue的路由管理对象下面的`replace`这个方法，从栈的角度去理角就相当于当前页面出栈，新页面入栈

注意事项:

1. `wx.navigateTo`最多只能套10层，因为它的页面栈只有10层
2. 上面的三个跳转页面方式都只是在普通页面栈里面跳转

小程序的tabBar配置

小程序的tabBar在以前的vue里面，我们可以通过二级路由去进行控制，但是在小程序里面，则要进行特殊的配置以后能可以使用

小程序的tabBar在`app.json`里面进行配置，具体可以参考官方文档https://developers.weixin.qq.com/miniprogram/dev/reference/configuration/app.html#tabBar

app.json的部分代码

```
{
    "tabBar": {
        "selectedColor": "#f4ea2a",
        "color": "#808080",
        "list": [
          {
            "text": "主页",
            "pagePath": "pages/index/index",
            "iconPath": "/images/index.png",
            "selectedIconPath": "/images/index_selected.png"
          },{
            "text": "直播",
            "pagePath": "pages/live/live",
            "iconPath": "/images/live.png",
            "selectedIconPath": "/images/live_selected.png"
          },
          {
            "text": "我的",
            "pagePath": "pages/my/my",
            "iconPath": "/images/user.png",
            "selectedIconPath": "/images/user_selected.png"
          }
        ]
      }
}
```

当上面的东西配置完以后，它会自动出现tabBar,并且已实现了页面跳转的响应

注意事项:

1. 上面的`tabBar`如果从vue的原理上面来讲，其实就是vue里面的二级路由，所以它的页面跳转方式与上面的普通路由栈的跳转方式是不一样的
    - `tabBar`里面配置的页面都是默认添加了`<keep-alive>`的
    - 它是一个特殊的页面栈，所以`wx.redirectTo`,`wx.navigateTo`都不能跳转`tabBar`里面的页面，如果要跳tabBar的页面，只能通过`wx.switchTab`这个方法

      同旦，`wx.switchTab`也不能跳普通页面栈

2. `tabBar`在配置的时候里面的图标不能使用`base64`图片，也不能使用`iconfont`图标，必须是一个实实在在的图片
3. `tabBar`在配置list的页面的时候，最少2个，最多5个
4. 还有一种页面跳转方式叫`wx.reLaunch`，小程序会重启到某一个页面，这个方法对任何页面都有效

小程序的界面交互

小程序提供了很多与用户及界面交互的方式方法

1. `wx.showLoading`在小程序的页面上面显示一个加载框
2. `wx.hideLoading`隐藏刚刚的加载框
3. `wx.showToast`显示轻消息提示
4. `wx.hideToast`隐藏轻消息提示
5. `wx.showModal`显示操作模拟框

```
wx.showModal({
    title:"警告",
    content:"你确定要删除吗？",
    cancelText:"爷不取消",
    confirmText:"老子确定",
    success:res=>{
        if(res.confirm){
            //说明用户点击是确定
            console.log("确定");
        }
        else{
            //说明用户点击的是取消
            console.log("取消");
        }
    }
})
```

6. `wx.showActionSheet`显示底部弹出的操作选项菜单

```
wx.showActionSheet({
    itemList: ["添加收藏","查看详细"],
    success:res=>{
        if(res.tapIndex==0){
            console.log("你点的是  添加收藏 ");
        }
        else if(res.tapIndex==1){
            console.log("你点的是  查看详细  ");
        }
    },
    fail:error=>{
        console.log("你点了个取消");
    }
});
```

7. `wx.previewImage`图片预览

小程序当中的iconfont的使用

小程序是可以使用字体图标，只是使用方式上面略有不同

开发环境

1. 在小程序的项目下面，我们新建一个文件夹叫`iconfont`，同时在这个文件夹下面创建一个`iconfont.wxss`文件
2. 在`iconfont`的网站上面，打开`font class`下面的这个连接，然后把这个链接里面的内容复制下来，粘贴到刚刚创建的`iconfont.wxss`的文件里面
3. 因为字体图标是需要全局使用的，所以我们在`app.wxss`里面去进行`@import`导入

```
/* 这里是全局的样式 */
@import "./iconfont/iconfont.wxss";
```

通过上面的步骤以后，我们就可以在开发环境下面正常的去使用字体图标

生产环境

正常情况下的生产环境里面，我们都会把字体图标下载下来，再去使用，但是因为小程序是在大小限制的，所以这一点可能有区别

1. 如果下载下来以后，字体文字比较大，我们还是会把这个下载以后的iconfont文件夹上传到自己的服务器，然后再使用这个在线链接，这样可以减少小程序的体积（因为小程序最多只能是2M）
2. 如果下载以后，字体图标文件比较小，则可以像以前一样，直接去使用

小程序的组件化开发

小程序本身是提供了很多组件的，但是我们也可以自定义一些组件（这一点与Vue是保持一致的）

1. 先在小程序的项目下面新建一个`components`的文件夹
2. 在刚刚`components`的文件夹下面继续创建组件名称的文件夹，如`TextItem`，接下来在这个文件夹下面点击鼠标右键，选择创建`Component`

其实的东西都相差无几，关键看生成的js文件

```
// components/TextItem/TextItem.js
Component({
  /**
   * 组件的属性列表
   */
  properties: {

  },

  /**
   * 组件的初始数据
   */
  data: {

  },

  /**
   * 组件的方法列表
   */
```

```
  methods: {

  }
})
```

上面的目录 结构与vue的目录结构是一模一样的，同时请注意这里与页面的区别

component组件的js与page页面的js的区别

1. 页面使用的`Page`的方法，而组件使用的是`Component`的方法
2. 页面里面的事件方法是直接写在对象下面，而组件里面的事件方法是要写在`methods`里面的
3. 页面里面会多出一个`properties`的属性，用于接收外部传给组件的值
4. 组件不是页面，所以它的生命周期与页面是不一样的，例如页面是有`onLoad`，而组件是没有的

组件生命周期

1. `created`组件生命周期函数-在组件实例刚刚被创建时执行，注意此时不能调用`setData`)，这个相当于vue里面的`beforeCreate`
2. `attached`组件生命周期函数-在组件实例进入页面节点树时执行)，相当于vue里面的`created`及`beforeMount`，在这里已经可以操作数据与方法了
3. `ready`组件生命周期函数-在组件布局完成后执行)，相当于vue里面的`mounted`，这个时候已经可以操作虚拟DOM了
4. `moved`组件生命周期函数-在组件实例被移动到节点树另一个位置时执行)
5. `detached`组件生命周期函数-在组件实例被从页面节点树移除时执行)

组件内部的方法与属性

1. `data`组件的属性列表，组件页面上面使用的数据都来源于这里
2. `properties`组件从外部接收的数据列表，写法与vue的对象语法基础保持一致
3. `observers`这个就是vue里面的`computed`与`watch`

组件的引用

一个页面如果要使用组件，或另一个组件要使用组件都需要提前进行配置，在当前页面或组件下面找对象的`json`文件，如`index.json`

index.json的代码

```
{
  "usingComponents": {
    "text-item":"/components/TextItem/TextItem"
  }
}
```

index.wxml的代码

```
<!-- 现在要使用组件了，怎么办呢 -->
<text-item></text-item>
```

父级组件的传值

这一点与vue保持一致，都是通过自定义属性传值 ，并且自定义属性都不能是驼峰命名

index.wxml页面里面的代码

```
<!-- userName是我们定义的变量，值为"标哥哥" -->
<text-item user-name="{{userName}}"></text-item>
```

TextItem.js组件里面的代码

```
properties: {
    userName:{
      type:String,
      value:"徐小江"
    }
  },
```

在组件里面，我们还是通过`this.data.userName`来正常取值就可以了

组件里面的`observers`

这个东西相当于vue当中的监听器`watch`

```
 observers:{
    age(newValue){
      console.log(newValue)
    }
  },
```

当我们对`age`去执行改变的时候，就会触发`observers`里面的操作

组件里在的插槽

在小程序里面，如果想使用插槽是可以使用的，但是它默认情况下只能使用默认插槽，不能使用具名插槽，如下

index.wxml里面的代码

```
<text-item user-name="{{userName}}">
  <!-- 这里是组件的插槽 -->
  <view>这是我插入进去的东西</view>
</text-item>
```

TextItem.wxml里面的代码

```
<view class="ttt1">
  这是一个组件啊，孩子们
</view>
<slot></slot>
```

上面的代码就是在使用默认插槽

但是在多数情况之下我们可能都会使用具名插槽，这个时候小程序的组件默认是允许的，需要在组件里面去配置开启

```
Component({
  options:{
    //开启了多插槽模式，也就是开启了具名插槽模式
    multipleSlots:true
  },
  //省略代码
)
```

组件数据流的单向性

当我们尝试着去改变外部传递给组件里面的数据的时候，我们发现组件内部的数据可以改变，但是外部的数据没有变，怎么办呢?

这一个过程其实可以去看一下我们的Vue，可以使用自定义事件去完成

TextItem.js里面的代码

```
methods: {
    haha(event) {
      // 我们尝试着改变父级传递过来的值
      //通过父级去改变这个值
      //vue里面使用 this.$emit("")
      // 小程序里面使用triggerEvent来触发一个自定义的事件
      this.triggerEvent("changeusername","天下第一帅哥-标哥");
    }
  }
```

在上面的代码当中，当我们触发`haha`的方法的时候，它会调用`this.triggerEvent`来触发一个自定义的事件，然后向外传值

index.wxml

```
<!-- 在组件这里，我们组定了一个自定义的事件 bindchangeusername-->
<text-item user-name="{{userName}}" bindchangeusername="callFather">
  <!-- 这里是组件的插槽 -->
  <view slot="header">-----这是我在头部插入的内容---</view>
  <view slot="footer">这是我插入进去的东西</view>
</text-item>
```

在使用这个组件的时候，我们绑定了一个自定义的事件 ，因为组件内部可能会触发一个自定义事件来进行传值

index.js

```
callFather(event){
    console.log(event);
    console.log("我儿子在叫爸爸");
    this.setData({
        userName:event.detail
    });
}
```

所传递过来的值在`event.detail`里面

父级使用子级组件的方法

在以前的vue里面是可以的，直接通过`this.$refs`选中某一个组件然后调用方法就可以了，在小程序里面原来差不多

TextITem.js里面的代码

在子级组件里面有一个方法叫callSon，我们看能否调用到这个方法

```
methods:{
    callSon(){
      console.log("父亲在调儿子，看能否调用，怎么调");
    }
}
```

index.wxml代码

我们在当前使用的组件上面添加了一个`id="mySon"`方法我们找到这个组件

```
<!-- 在组件这里，我们组定了一个自定义的事件 bindchangeusername-->
<text-item user-name="{{userName}}" bindchangeusername="callFather" id="mySon">
  <!-- 这里是组件的插槽 -->
  <view slot="header">-----这是我在头部插入的内容---</view>
  <view slot="footer">这是我插入进去的东西</view>
</text-item>
```

index.js的代码

```
{
    sayHello(){
        // 我要调子级组件TextItem下面的callSon这个方法 ， 咋办呢？
        // 我儿子【组件】的id叫mySon
        this.selectComponent("#mySon").callSon();
    }
}
```

小程序的数据交互

小程序的开发也是一种前后端分离的开发模式，它也需要后端提供接口才可以开发【云开发模式除外】，小程序的内部也提供了获取数据的方式（可以理解为ajax请求）

1. `wx.request`发起一个`http`或`https`的请求，相当于`ajax`
2. `wx.downloadFile`下载一个文件
3. `wx.uploadFile`上传一个文件
4. `WebSocket`这个是基于双工通信的socket来的

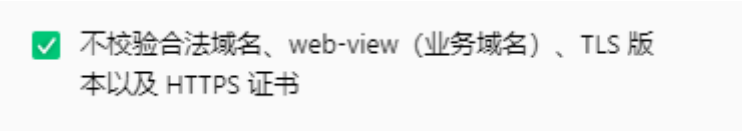
以上的四种全部都是基于TCP下面的通信，而1~3又是基于http模式下面的通讯

小程序在发起请求之前一定要进行配置，它只允许在特性的配置项项里面发起请求，并且这个请求只能是`https`协议的

如果没有经过配置 则会出现上面的情况

针对上面的问题，我们可以使用2种方式解决

第一种方式：在开发环境下面不检测域名的合法性

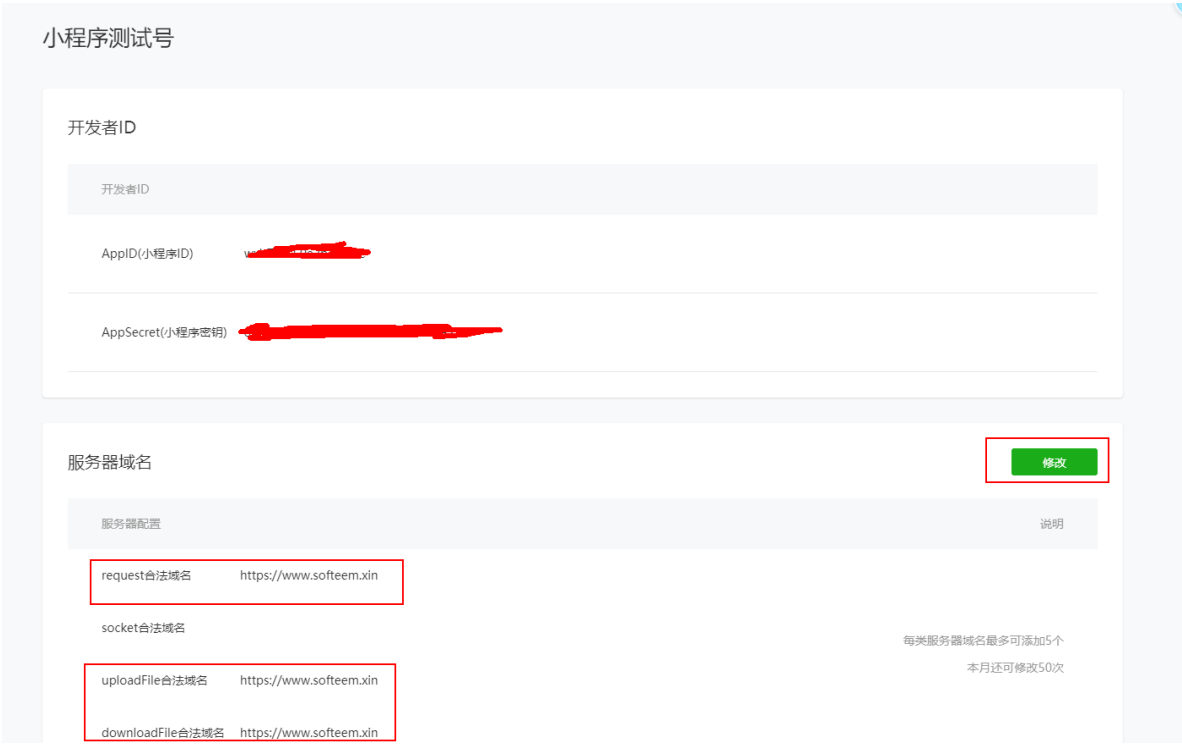


在开发工具上面，把这个选项钩上即可，但是这么做只能在开发环境下面，小程序如果真机预览或发布以后这是不可以的

第二种方式：直接在小程序的账号的后台去配置一下

下面以测试号为例子去配置

在后台对请求域名进行配置



一旦配置完成以后就会在小程序开发工具里面看到这个信息


```
getData() {
    //显示一个加载的动画
    wx.showLoading({
      title: '正在请求数据',
    });
    wx.request({
      method: "GET",
      url: 'https://m2.qiushibaike.com/article/list/text',
      data: {
        page: 1,
        count: 12,
        readarticles: "[123015030]"
      },
      success: res => {
        //请求成功
        //请注意，这个res就相当于resp
        console.log(res);
      },
      fail: error => {
        //请求失败
        console.log("请求失败了");
        console.log(error);
      },
      complete: () => {
        //请求完成
        //关闭加载的动画
        wx.hideLoading();
      }
    })
  }
```

最后再说一点，一般情况下，我们都不会直接去使用`wx.request`，而是要进行二次封装，与我们之前所学习的`axios`一样的

封装过程

1. 先在小程序的项目下面新建一个文件utils，然后在utils的文件夹里面创建API.js文件
2. 根据自己的业务需求去编写接口请求代码，如下

```
/**
 * API对象
 */
export default {
  Qiushi: {
    //获取纯文列表
    getTextList({
      page,
      count,
      readarticles
    }) {
      return new Promise((resolev, reject) => {
        wx.request({
          method: "GET",
          url: 'https://m2.qiushibaike.com/article/list/text',
          data: {
            page,
            count,
            readarticles
          },
          success: res => {
            //请求成功
            resolev(res.data);
          },
          fail: error => {
            //请求失败
            reject(error);
          }
        })
      });
    }
  }
}
```

3. 当编写完接口文档以后就可以正常的去使用了这个接口了，你可以使用`async/await`来完成，当然如果报错可以导入异步的包`regenerator-runtime`

```
async getData() {
    wx.showLoading({
      title: '正在请求数据...'
    });
    try {
      let result = await
API.Qiushi.getTextList({page:1,count:12,readarticles:"[123063612]"})
      console.log(result);
    } catch (error) {
      console.log(error);
    }
    finally{
      wx.hideLoading();
    }
  }
```

小程序的页面传值

小程序的跨页面传值 使用的仍然是`search`的模式，这一点与之前没有区别

index.js里面的代码

```
//去详细信息的页面
  toDetail(){
    wx.navigateTo({
      url: '/pages/detail/detail?id=123&userName=标哥哥'
    });
  }
```

在上面的页面跳转的时候，我们给它添加了一个页面地址的同时还在后面添加了`?id=123&userName=标哥哥`，这就是`search`参数

那么怎么样在目标页面接收到参数呢

detail.js里面的代码

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

在目标页面的生命周期钩子函数`onLoad`里面，它会有一个参数`options`，这个参数就包含了之前的页面通过`search`传递过来的值