目录

[1、 基础篇 2](#_Toc47512444)

[1.1、 组件 2](#_Toc47512445)

[1.1.1、 组件的引入 2](#_Toc47512446)

[1.1.2、 组件间的数据传递 7](#_Toc47512447)

[1.1.3、 微信小程序-周期 14](#_Toc47512448)

[2、 实战篇 18](#_Toc47512449)

[2.1、 项目技术选型 18](#_Toc47512450)

[2.2、 创建并清理项目 18](#_Toc47512451)

[2.2.1、 创建新项目 18](#_Toc47512452)

[2.2.2、 清理无用的代码 19](#_Toc47512453)

[2.2.3、 去掉首页提示 20](#_Toc47512454)

[2.2.4、 创建各个包 21](#_Toc47512455)

[2.2.5、 创建各个page页 21](#_Toc47512456)

[2.2.6、 引⼊字体图标iconfont 22](#_Toc47512457)

[2.3、 创建tabBar 30](#_Toc47512458)

[2.2.1、 导入图标 30](#_Toc47512459)

[2.2.2、 入口文件APP.JSON中代码 30](#_Toc47512460)

[2.4、 初始化页面样式 32](#_Toc47512461)

[2.4.1、 通配的标签样式 32](#_Toc47512462)

[2.4.2、 页面风格全局初始化 33](#_Toc47512463)

[2.5、 搜索框 36](#_Toc47512464)

[2.5.1、 创建组件模块并引入到首页中 36](#_Toc47512465)

[2.5.2、 完善组件样式 37](#_Toc47512466)

[2.6、 轮播图 38](#_Toc47512467)

[3、 后端 39](#_Toc47512468)

[3.1、 数据库搭建 39](#_Toc47512469)

[3.2、 后端SpringBoot开发 41](#_Toc47512470)

[3.2.1、 开发工具 41](#_Toc47512471)

[3.2.2、 SpringBoot框架搭建 41](#_Toc47512472)

[3.2.3、 导入依赖(POM.XML) 45](#_Toc47512473)

[3.2.4、 项目启动准备 48](#_Toc47512474)

[3.2.5、 自动生成生成Entry与Mapper 53](#_Toc47512475)

[3.2.6、 自定义响应数据结构 60](#_Toc47512476)

[3.2.7、 业务开发 64](#_Toc47512477)

# 基础篇

## 组件

组件的作用类似于抽象类一样的存在（Java开发中的某一理论），可一将可共用的功能抽取出来，公共使用；

### 组件的引入

#### 初步关联

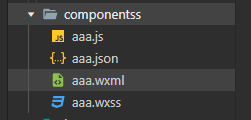
引用很简单，一行代码的事，不过要从创建组件开始；

1. **第一步：创建组件**

在“pages”文件夹平级位置创建一个“components”文件夹，这个文件夹就是子组件文件夹，然后右击从文件夹，选择弹出的选项栏里的“新建Component”选项，出现如下图所示：



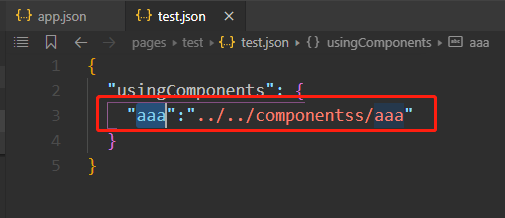
输入所要创建的子组件的名称，填写完子组件名称后回车，再次打开子组件文件夹，会看到文件夹下出现四个文件（此操作尽量在微信开发者工具中使用，其他的开发工具不一定能一键生成这四个文件），如下所示：



这些文件和“pages”文件夹下的很像，实际使用也是基本一致；

1. **第二步：在父组件（页面）中引入子组件**

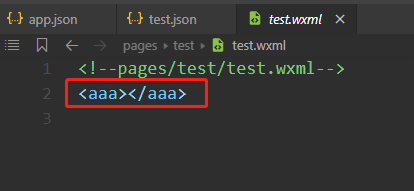
创建一个新的“pages”模块，起名为“test”并将此文件路径放在“app.json”中的“pages”标签中的第一个，然后找到“test”模块下的“json”文件，然后在“usingComponents”代码块中创建如下所示的代码：



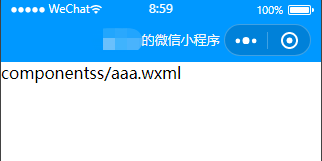
以键值对的形式存在，“aaa”是子组件在父组件中使用时的名称，算是个别名，“../../componentss/aaa”是子组件的相对路径；

1. **第三步：在父组件中使用子组件**

此时已经关联好了，最简单的是方法就是将子组件的别名写到父组件（test）的wxnl文件中，如下所示：



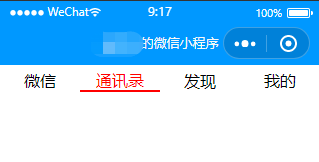
其他的什么都不用写，然后查看模拟器中的界面；



可以看到，界面中显示的地址是“aaa”组件的“wxml”文件中的数据，这就是引入依赖成功的标志；

#### 实战关联

需求：通过组件来完成一个导航栏，点击导航文字时存在样式变化，大概样式如下图所示：



代码：（主要是将导航栏功能代码）

1. 在组建代码集中分别在其wxss文件中写入：

.tabs{

}

.tabs\_title{

  display: flex;

  padding: 10rpx 0;

}

.title\_item{

  flex: 1;

  display: flex;

  justify-content: center;

  align-items: center;

}

.active{

  color: red;

  border-bottom: 5rpx solid red;

}

1. 在其json文件中设置

{

  "component": true,

  "usingComponents": {}

}

（将 component 字段设为 true 可将这一组文件标记为自定义组件；）

1. 在其js文件中设定数据显示数据以及操作所对应的变化逻辑，代码如下：

// componentss/aaa.js

Component({

  /\*\*

   \* 组件的属性列表

   \*/

  properties: {

  },

  /\*\*

   \* 组件的初始数据

   \*/

  data: {

    tabs: [

      {

        id: "0",

        name: "微信",

        isActive: true

      }, {

        id: "1",

        name: "通讯录",

        isActive: false

      }, {

        id: "2",

        name: "发现",

        isActive: false

      }, {

        id: "3",

        name: "我的",

        isActive: false

      }

    ]

  },

  /\*\*

   \* 组件的方法列表

   \*/

  methods: {

    handleChange(e) {

      // 获取被选中的索引

      const { index } = e.currentTarget.dataset;

      // 获取列表数据

      // 复制获取数据 - 最严谨安全的做法 - 深拷贝

      // 深拷贝 - 通过json的序列化反序列化：json.stringify-将json对象转化为字符串对象，json.parse-将字符串对象转化为json对象

      //let {tabs} = JSON.parse(JSON.stringify(thsi.data.tabs));

      // 直接操作数据

      let { tabs } = this.data;

      // 循环修改列表中的isactive属性

      tabs.forEach((v, i) => i === index ? v.isActive = true : v.isActive = false);

      // 数据放回

      this.setData({

        tabs

      })

    }

  }

})

1. 在子组件的wxml文件中编写如下代码：

<view class="tabs">

    <view class="tabs\_title">

        <view wx:for="{{tabs}}" wx:key="id" class="title\_item {{item.isActive?'active':''}}" bindtap="handleChange" data-index="{{index}}">

            {{item.name}}

        </view>

    </view>

</view>

1. 完成以上代码后，在我们的父组件文件中的json文件中关联引用子组件，代码如下：

"usingComponents": {

    "Tabs":"../../componentss/aaa"

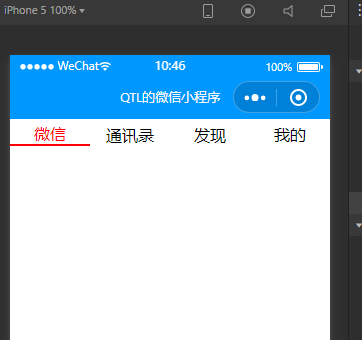
  }

1. 再在父组件的wxml文件中引入子组件代码：

<Tabs></Tabs>

完成！

保存后模拟器自动出现如下所示的界面：



### 组件间的数据传递

通过以上的叙述，我们已经明白了如何从一个子组件中获取其对应的现成的内容，但是这个内容有一个硬伤，那就是它们都是被写死的，如果这么搞下去，一个功能稍微丰富的点的微信小程序，在组建方面的灵活性就太差了，因此，我们接下来要学的是组件间的数据传递，分别是父组件（页面）向子组件传递数据、子组件向父组件传递数据，传递数据的作用是将子组件的功能作为一个封装好的对象，通过调用该对象并给其传值来获得。

#### 父组件（页面）向子组件传递数据

用之前的代码，只不过需要将某些代码块的存放位置换一下地方，很明显，在父组件向子组件传递数据的练习中肯定是数据的传递为重心，那么只能是目录数组数据需要被更换存放地：

tabs: [

      {

        id: "0",

        name: "微信",

        isActive: true

      }, {

        id: "1",

        name: "通讯录",

        isActive: false

      }, {

        id: "2",

        name: "发现",

        isActive: false

      }, {

        id: "3",

        name: "我的",

        isActive: false

      }

    ]

将“tabs”数组从子组件的js文件中的date块中转移到父组件的js文件的date块中；

然后再将数据通过标签传递给子组件，代码如下：

<!-- 父组件向子组件传递数据 -->

<!-- 父组件（页面）向子组件传递数据 使用的是标签属性进行传递

        |--第一步：在父组件页面中定义三个数据：接收数据的子组件名称、传递的数据对象名称、数据值

        |   |-- <Tabs aaa="{{a}}"></Tabs>：Tabs：子组件名称； aaa：传递的数据对象名称； “a”：数据值

        |

        |--第二步：在子组件的js文件的“properties”代码块中编写接收数据信息

        |   |-- aaa: { type: Array, value: "" }

        |

        |--第三步：在子组件的wxml文件中对数据进行展示或操作

        |-- -- -- -- -- 注意：在各个名称命名时要注意其之间的关系；

<Tabs tabs="{{tabs}}"></Tabs>

这样就完成了一个父组件向子组件传递数据的。

#### 子组件（页面）向父组件传递数据

通过子组件向父组件传递数据达到修改数据的状态，这里主要通过自定义方法实现子组件向父组件传递数据的目的；

思路：

首先，创建一个子组件模块集和一个父组件的模块集，然后对这两个模块分别进行编码，将子组件关联到父组件中，然后分析：用到的导航条数据组的话仍然放到父模块的js文件中，而对应的样式也依然放到子组件的wxss文件中，子组件的组件的属性列表：“properties”中依然声明传递的数据类型、数据结构，当点击子组件的导航信息时，仍然调用子组件的样式以及点击绑定方法（只不过方法会有所变化），因此，组件的wxml、wxjs中methods中的handleChange(e)方法都要存在，有变化的地方应该就是父组件的js和子组件的js，通过子组件来调用父组件的自定义方法，然后在父组件中处理相应的数据；

代码执行逻辑：

当点击导航信息时（导航信息在子组件中），会触发其绑定的handleChange(e)方法，在此方法中获取被选中的导航元素的码值，然后在通过调用父组件中的自定义方法然后通过此方法把码值传给父组件，然后父组件通过这个码值来进行选中样式切换；

代码：

子组件的js文件代码——组件数据属性列表、组件方法列表handleChange：

**// components/tabss/tabss.js**

**Component({**

**/\*\***

**\* 组件的属性列表**

**\*/**

**properties: {**

**tabs: {**

**type: Array,**

**value: []**

**}**

**},**

**/\*\***

**\* 组件的初始数据**

**\*/**

**data: {**

**},**

**/\*\***

**\* 组件的方法列表**

**\*/**

**methods: {**

**handleChange(e) {**

**// 获取被选中的索引**

**const { index } = e.currentTarget.dataset;**

**// 调用自定义方法**

**this.triggerEvent("itemChange",{index});**

**}**

**}**

**})**

子组件的wxml文件代码——导航条数据格式：

**<view class="tabs">**

**<view class="tabs\_title">**

**<view wx:for="{{tabs}}" wx:key="id" class="title\_item {{item.isActive?'active':''}}" bindtap="handleChange" data-index="{{index}}">**

**{{item.name}}**

**</view>**

**</view>**

**</view>**

父组件的json文件代码——关联子组件：

**{**

**"usingComponents": {**

**"Tabs":"../../components/tabss/tabss"**

**}**

**}**

父组件的js文件代码——导航数据组、自定义数据处理方法handleItemChange：

**// pages/components/componets.js**

**Page({**

**/\*\***

**\* 页面的初始数据**

**\*/**

**data: {**

**tabs: [**

**{**

**id: "0",**

**name: "微信",**

**isActive: true**

**}, {**

**id: "1",**

**name: "通讯录",**

**isActive: false**

**}, {**

**id: "2",**

**name: "发现",**

**isActive: false**

**}, {**

**id: "3",**

**name: "我的",**

**isActive: false**

**}**

**]**

**},**

**/\*\***

**\* 自定义方法：handleItemChange**

**\*/**

**handleItemChange(e) {**

**const { index } = e.detail;**

**let { tabs } = this.data;**

**// 循环修改列表中的isactive属性**

**tabs.forEach((v, i) => i === index ? v.isActive = true : v.isActive = false);**

**// 数据放回**

**this.setData({**

**tabs**

**})**

**}**

**:**

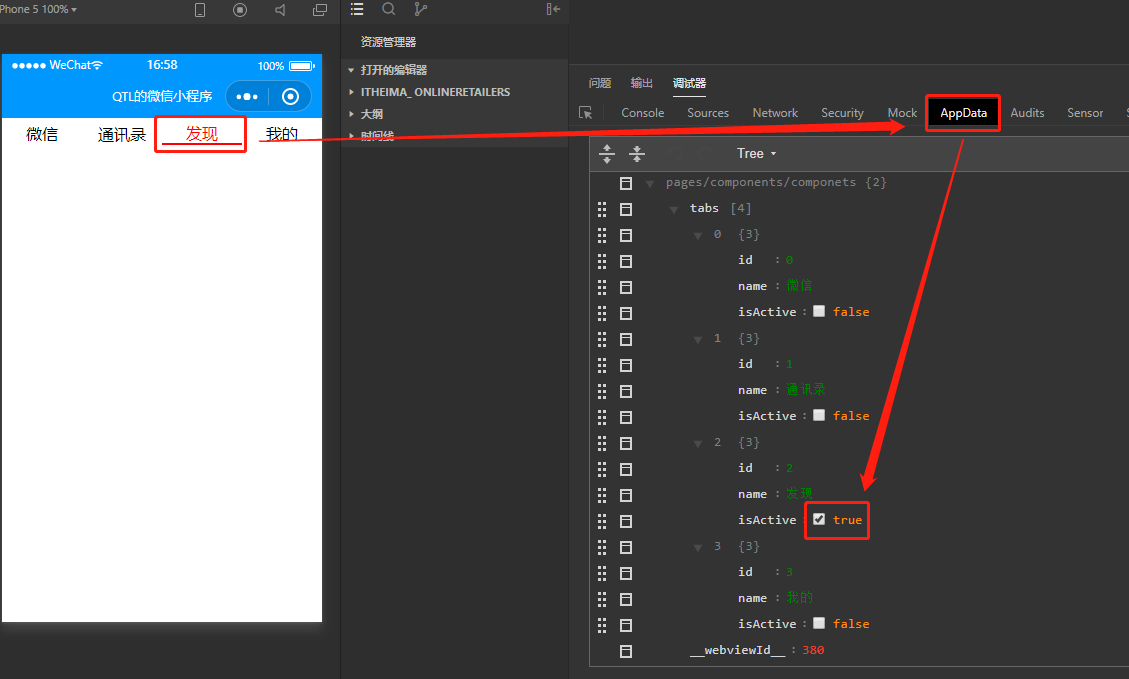
**:**

**}**

父组件的wxml文件代码——导航数据组传递{{tabs}}、自定义方法handleItemChange：

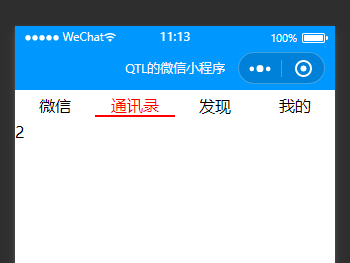
**<Tabs tabs="{{tabs}}" binditemChange="handleItemChange"></Tabs>**

查看“微信小程序开发工具”中的日志台中的AppData块，然后点击模拟器中的导航元素，发现其中的选中属性会跟着改变；



#### Slot标签

新需求：实现点击不同的导航条，下方显示不同的内容；



这样的话就需要一个占位符，声明内容显示的地方被占用了，使用的就是slot标签。

代码实现（在原来的代码中进行补充）：

在子组件的wxml文件中：

**<view class="tabs">**

**<view class="tabs\_title">**

**<view wx:for="{{tabs}}" wx:key="id" class="title\_item {{item.isActive?'active':''}}" bindtap="handleChange" data-index="{{index}}">**

**{{item.name}}**

**</view>**

**</view>**

**</view>**

**<view>**

**<!--**

**slot标签是一个占位符、插槽，当父组件调用子组件是传递标签过来，**

**最终传递的标签就会替换了slot插槽的位置**

**-->**

**<slot></slot>**

**</view>**

在父组件的wxml文件中：

**<Tabs tabs="{{tabs}}" binditemChange="handleItemChange">**

**<block wx:if="{{tabs[0].isActive}}">1</block>**

**<block wx:elif="{{tabs[1].isActive}}">2</block>**

**<block wx:elif="{{tabs[2].isActive}}">3</block>**

**<block wx:else>4</block>**

**</Tabs>**

没错，只需要这两段。在子组件中加上<slot>标签占位，然后再在父组件中在引入插件标签名内写入要传入的逻辑、数据，就能实现需求；

### 微信小程序-周期

#### 应用生命周期

应用生命周期的事件定义方法都写在微信小程序的入口文件中——app.js；一般应用常用的事件定义有onLaunch()、onShow()、onHide()等…

**// 加载启动小程序时调用此事件，一般用于用户信息获取等**

**onLaunch(){**

**console.log("onLaunch");**

**//abc**

**},**

**// 切换进小程序时，更新、重置页面数据**

**onShow(){**

**console.log("onShow");**

**},**

**// 切换到小程序外时，暂停、清除定时器**

**onHide(){**

**console.log("onHid");**

**},**

**// 当出现错误时，调用此方法收集错误信息并可发送给后台**

**onError(err){**

**console.log("onError");**

**console.log(err);**

**},**

**// 当每次启动小程序时找不到入口文件时会调用此方法，注意，是入口文件找不到时，一旦进入onload方法就是入口文件已经找到，通过修改文件入口的配置来达到查看此方法的效果**

**// 应用场景：当无法找到首页时通过此方法定义第二个页面让用户进入，相当于二皮脸**

**onPageNotFound(){**

**console.log("onPageNotFound");**

**wx.navigateTo({**

**// 二皮脸要求和导航页面设置要求相同，即必须不可是tabBar中的那几个页面**

**url: '/pages/button/button'**

**});**

**},**

#### 页面生命周期

**/\*\***

**\* 页面的初始数据**

**\*/**

**data: {**

**},**

**/\*\***

**\* 生命周期函数--监听页面加载**

**\*/**

**onLoad: function (options) {**

**console.log("onLoad");**

**},**

**/\*\***

**\* 生命周期函数--监听页面显示**

**\*/**

**onShow: function () {**

**console.log("onShow");**

**},**

**/\*\***

**\* 生命周期函数--监听页面初次渲染完成**

**\*/**

**onReady: function () {**

**console.log("onReady");**

**},**

**/\*\***

**\* 生命周期函数--监听页面隐藏**

**\*/**

**onHide: function () {**

**// 跳转到其他页面（不关闭此页面的跳转形式）时会触发此监听**

**console.log("onHide");**

**},**

**/\*\***

**\* 生命周期函数--监听页面卸载**

**\*/**

**onUnload: function () {**

**// 其实就是当前页面被关闭时会触发此监听，例如：当我关闭此页面跳转到其他页面时会触发**

**console.log("onUnload");**

**},**

**/\*\***

**\* 页面相关事件处理函数--监听用户下拉动作**

**\*/**

**onPullDownRefresh: function () {**

**// 首先你要开启允许下拉操作才能由此监控效果出现**

**// 目的：页面数据、效果的下拉刷新、重载**

**console.log("onPullDownRefresh");  },**

**/\*\***

**\* 页面上拉触底事件的处理函数**

**\*/**

**onReachBottom: function () {**

**// 上拉加载下一页数据，首先你的页面数据高度要达到足够页面滚动**

**console.log("onReachBottom");**

**},**

**/\*\***

**\* 用户点击右上角分享**

**\*/**

**onShareAppMessage: function () {**

**console.log("onShareAppMessage");**

**},**

**/\*\***

**\* 页面滚动时间处理函数--监听页面滚动动作**

**\*/**

**onPageScroll: function () {**

**// 只要滚动就会触发此监听**

**console.log("onPageScroll");**

**},**

**/\*\***

**\* 屏幕尺寸变化监听--横屏、竖屏变化监听**

**\*/**

**onResize: function(){**

**// 首先开启允许横屏竖屏切换**

**console.log("onResize");**

**}**

# 实战篇

## 项目技术选型

做一个真正的项目，肯定还是需要使用框架的，现在的框架常营的有这么几家：

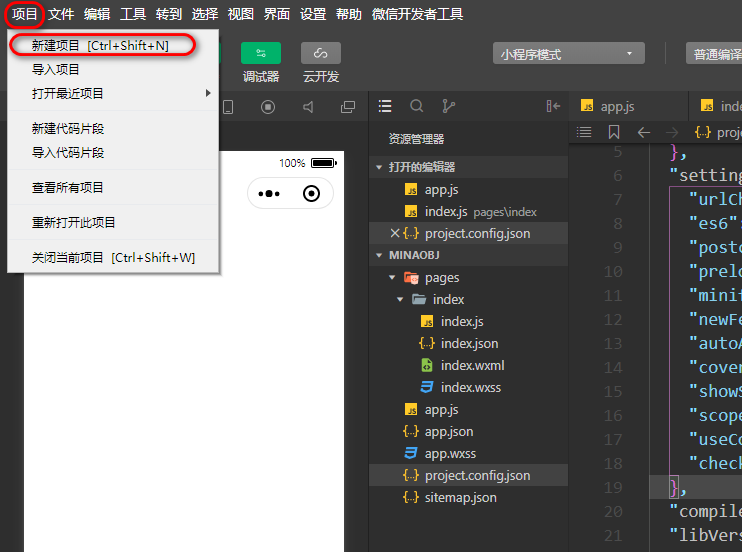
* 腾讯wepy
* 美团 mpvue
* 京东taro
* 滴滴 chameleon
* uni-app

现在的技术更新迭代比较快，暂不选择框架进行开发，仍然选择原生的微信MINA开发；

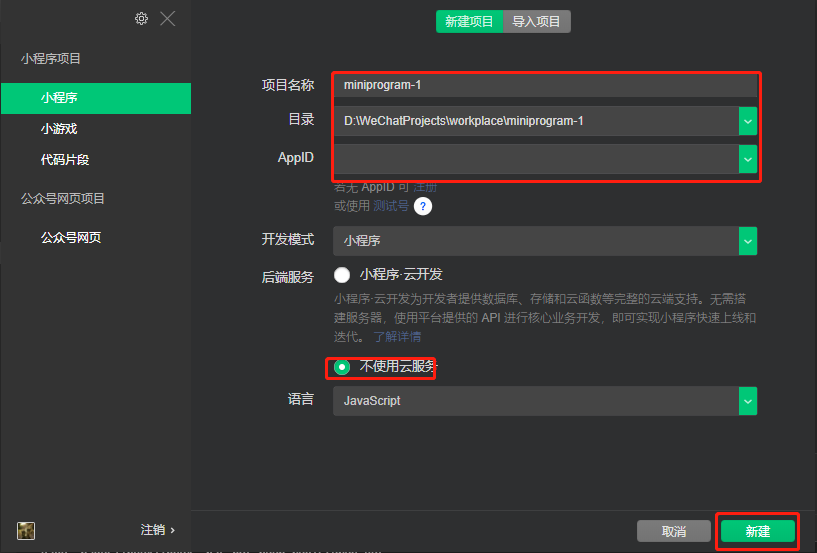
## 创建并清理项目

### 创建新项目

按照如下图所示的操作，直接能获取到一个新的项目；



进入创建页面：

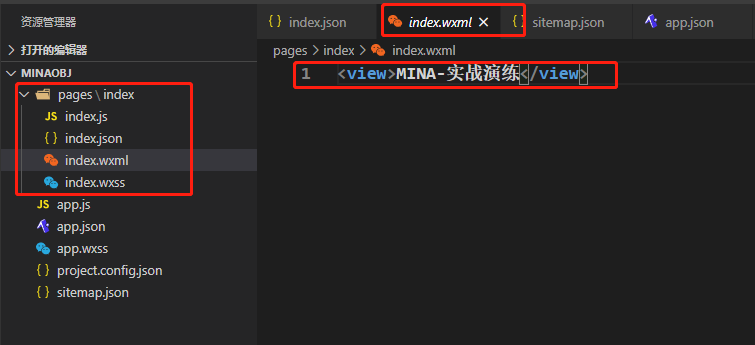


然后就创建出来了一个新的项目；

### 清理无用的代码

#### Pages文件中代码清理

删除日志文件夹及其下的文件，删除index文件中的wxml内容和wxss中的样式；操作完后如下所示：



#### 入口相关文件代码清理

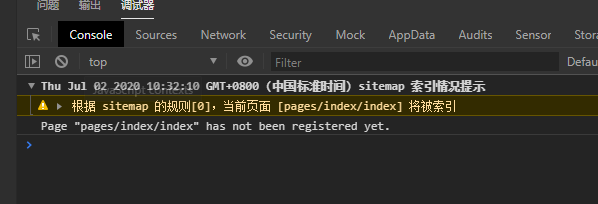
APP.JS文件中的预置代码清除掉，只留下相应的方法；

APP.JSON文件中的pages块中只留下index的，日志页的删除掉；

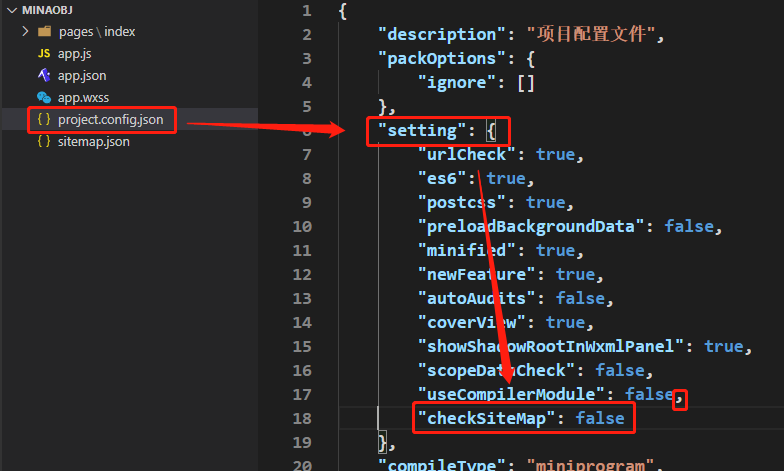
APP.WXSS文件清空；

### 去掉首页提示

去掉图中黄色的提示（**根据 sitemap 的规则[0]，当前页面 [pages/index/index] 将被索引**），太扎眼了；

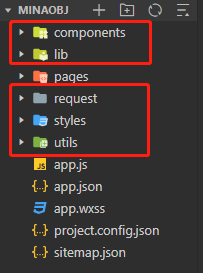


去除方式-添加”checkSiteMap”:false 到如下位置：



### 创建各个包

* components：组件包
* lib：引入外部公共代码包
* utils：自己的公共代码包
* request：请求接口包
* styles：公共样式包

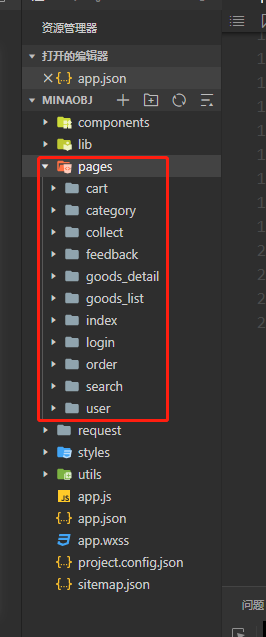


### 创建各个page页

各个页面名称：

|  |  |
| --- | --- |
| ⻚⾯名称 | 名称 |
| ⾸⻚ | index |
| 分类⻚⾯ | category |
| 商品列表⻚⾯ | goods\_list |
| 商品详情⻚⾯ | goods\_detail |
| 购物⻋⻚⾯ | cart |
| 收藏⻚⾯ | collect |
| 订单⻚⾯ | order |
| 搜索⻚⾯ | search |
| 个⼈中⼼⻚⾯ | user |
| 意⻅反馈⻚⾯ | feedback |
| 登录⻚⾯ | login |
| 授权⻚⾯ | auth |
| 结算⻚⾯ | pay |

创建好后的pages文件夹下截图：

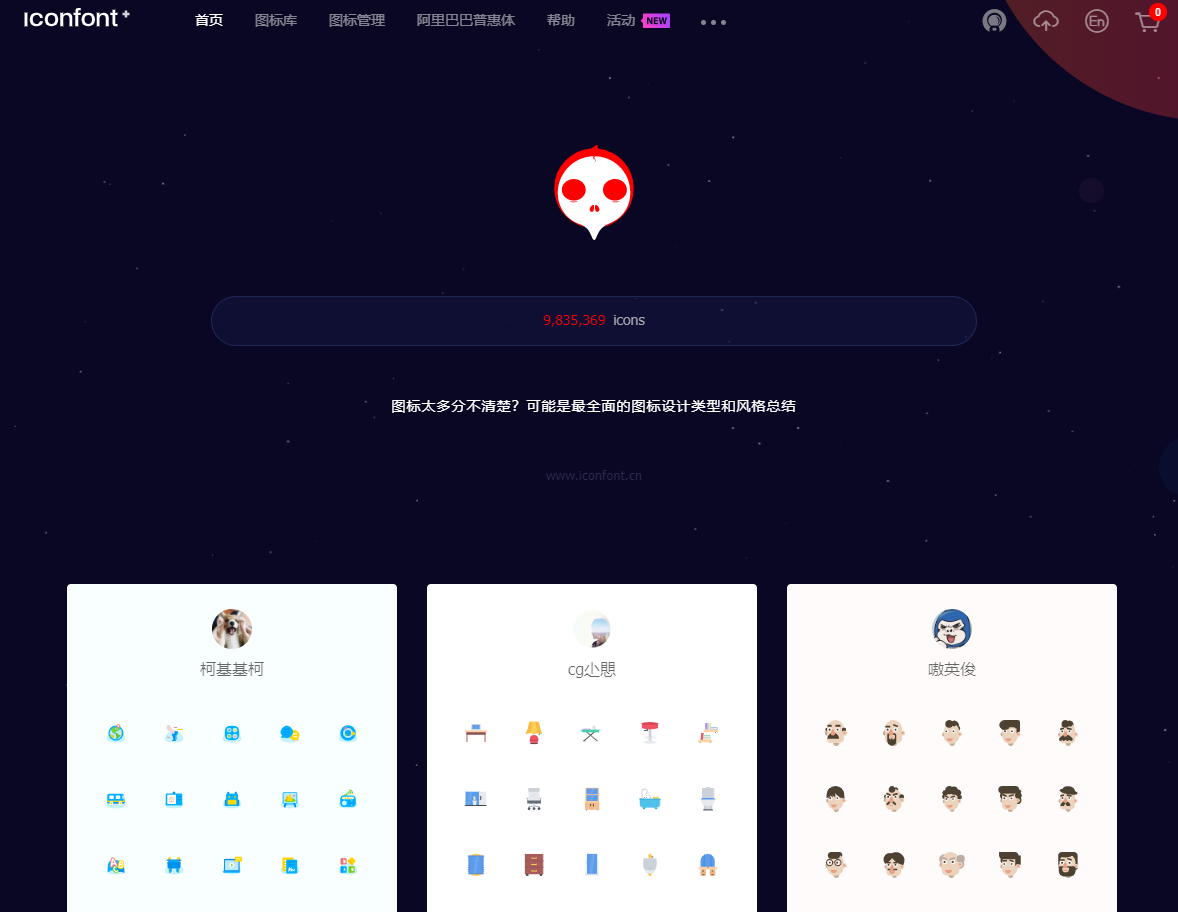


### 引⼊字体图标iconfont

#### 在线生成样式代码

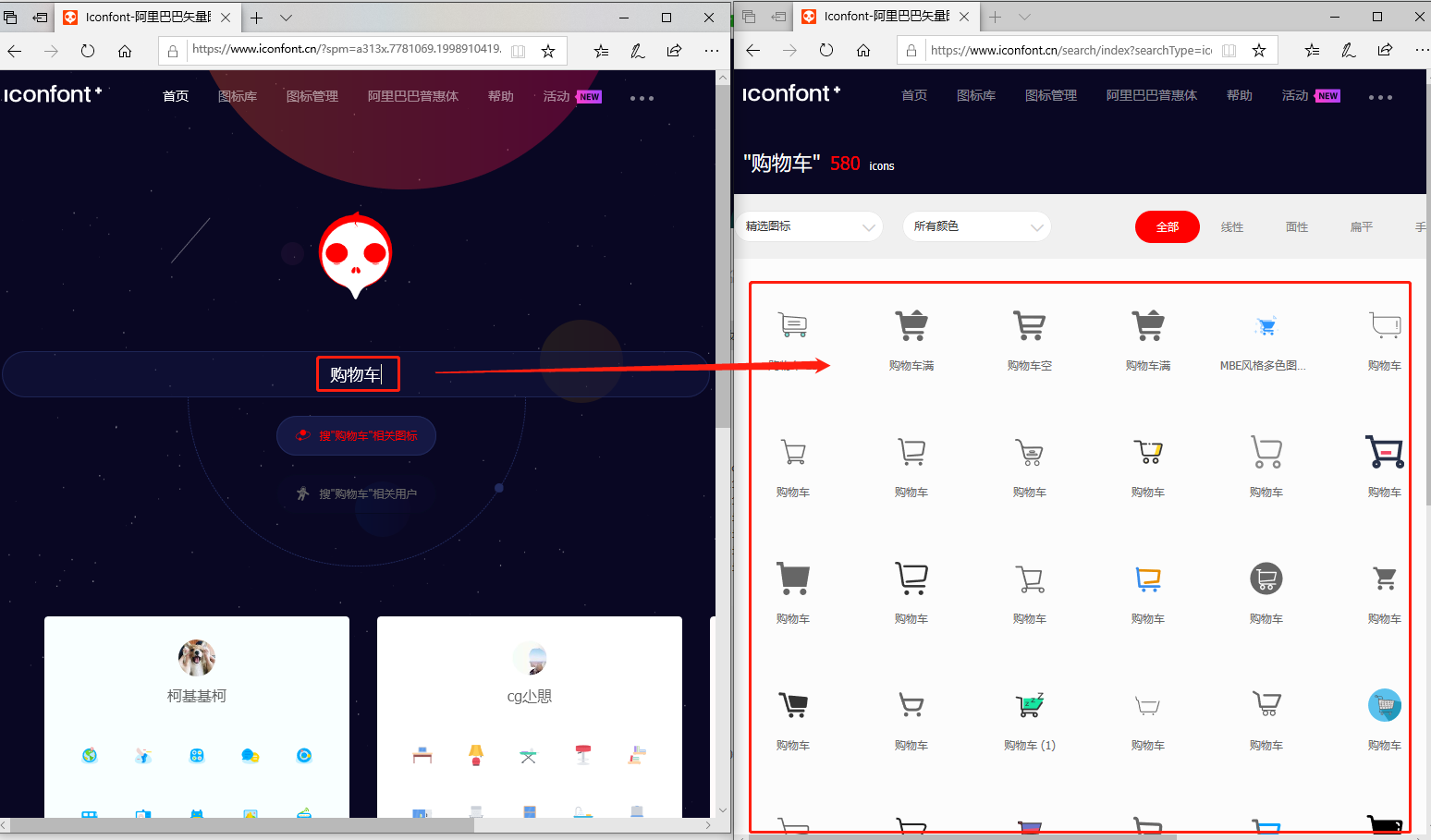
* 使用阿里巴巴图标库

阿里巴巴图标库网址：https://www.iconfont.cn/?spm=a313x.7781069.1998910419.d4d0a486a



* 选择的图标

在图标库首页搜索框直接搜索需要的图标，中文英文名称均可，然后选择所需要的图标添加到购物车中；

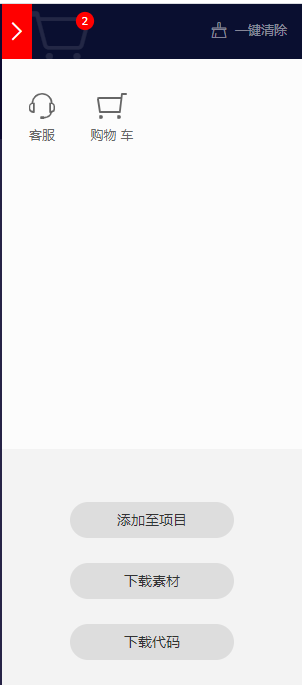


* 添加到项目中

选择那个小车车



选择完后会看到右上角的购物车里会有自己添加了几个图标的数量，然后点击那个购物车，进入到自己添加到购物车里的图标；



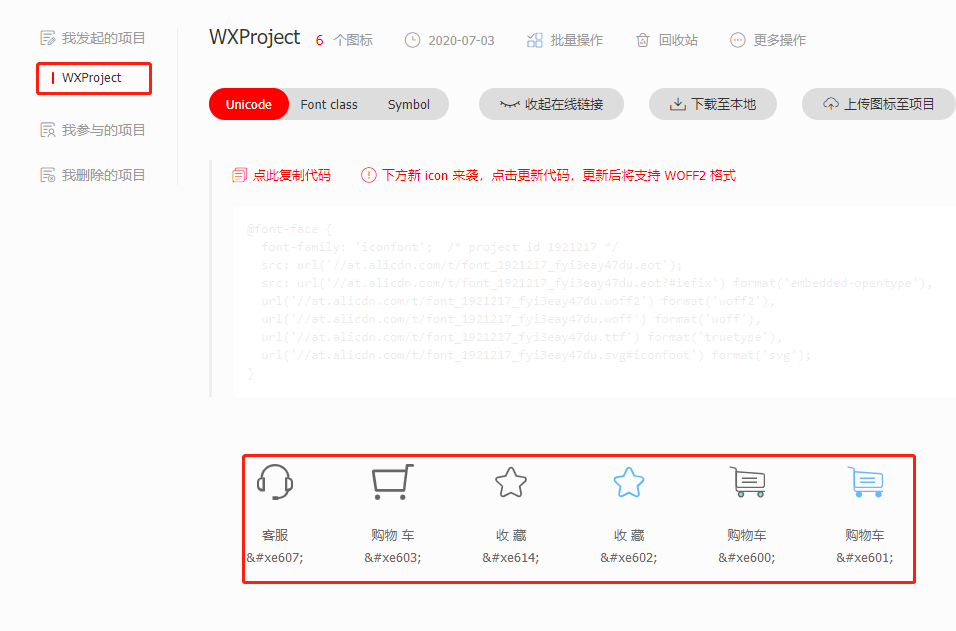
选择添加至项目，然后创建一个项目：



然后将自己之前选择的图标放进去；

* 生成在线链接

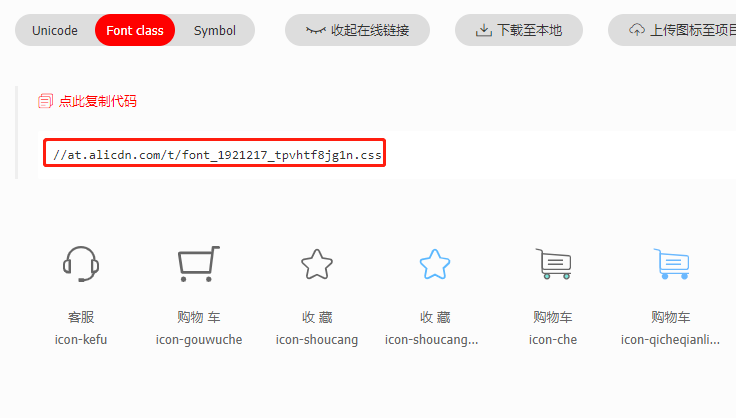
之前选择的图标放到项目中后：



然后生成在线连接（当第一次在自己创建的项目中生成在线连接时是没有“点此复制代码”的，需要先生成）：



生成的在线链接如下：



* 打开链接并复制代码

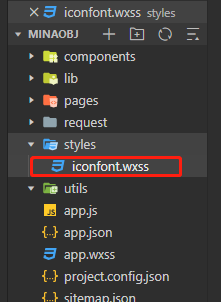
复制生成的在线链接放到浏览器网址栏中并打开，如下所示：



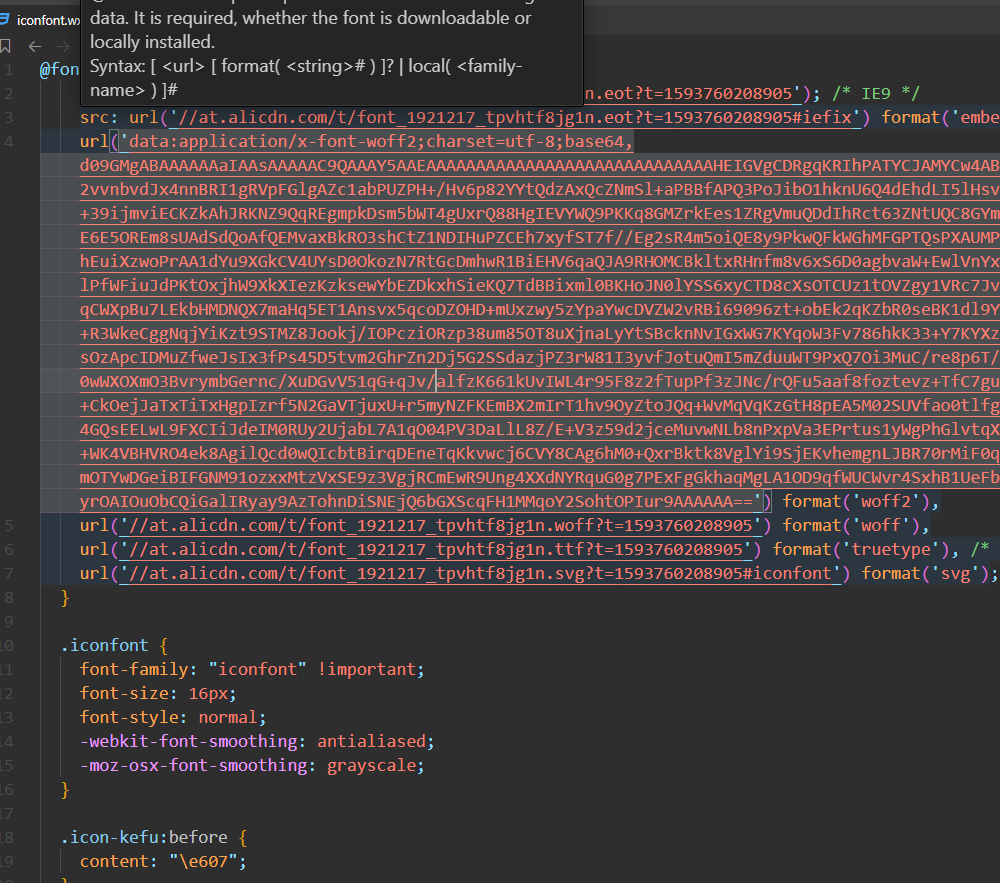
复制上图中的代码；

* 粘贴到小程序中的styles文件夹下（需创建一个图标wxss文件）

微信小程序的Styles文件夹下创建一个图标css文件；



然后将之前在网页中复制的代码粘贴到新建的iconfont.wxss文件中，如下所示：



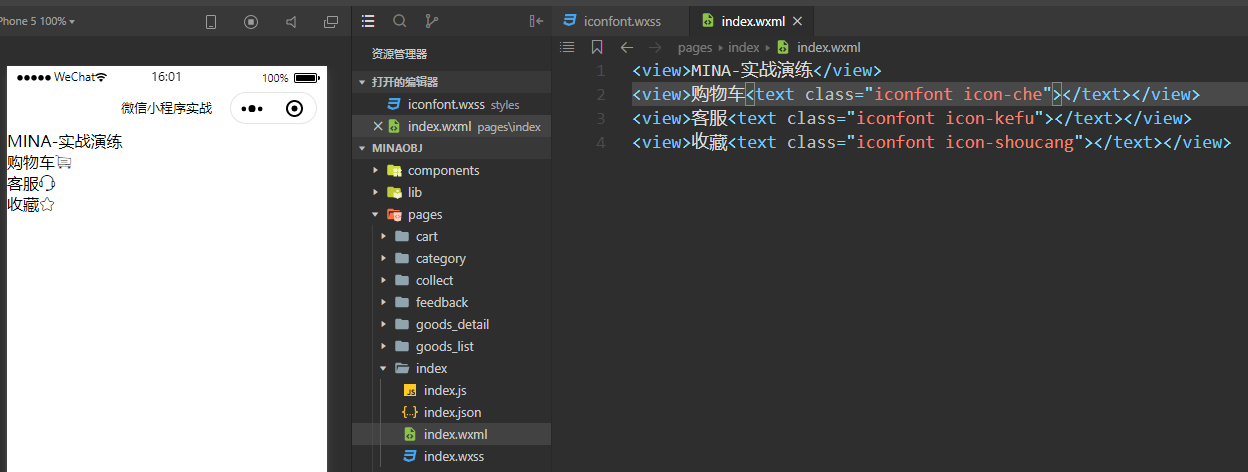
#### 生成代码在项目中使用

通过之前的操作，我们将在线生成的图标代码复制到iconfont.wxss文件中后就可以通过引用后使用了，一般系统的图标要具有一致性的风格和相同含义图标一致化的要求，既，需要我们使用的图标尽量全局话，因此我们引用需要进行全局引用，引用代码如下：

在app.wxss中：

@import"./styles/iconfont.wxss"

在某个页面中尝试一下，代码和显示样式如下：



注意:

Class的值，例如图中：

**class="iconfont icon-kefu"**

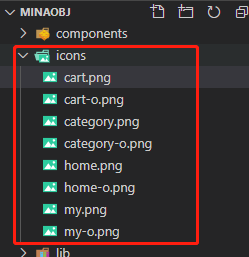
此段的值是由两部分组成，要对照着iconfont.wsxx文件来看，第一个“iconfont”是总样式名称，“icon- kefu”是对应的具体哪个小样式；



## 创建tabBar

### 导入图标

导入tabBar需要用到的图标（这些图标可以从阿里巴巴图标矢量库中进行下载，和上一步引入图标类似，只不过是把图标作为图片下载下来即可），首先创建一个“icons”文件夹在项目中，专门用于本地的图标，如下所示：



### 入口文件APP.JSON中代码

下方这段代码是在最外层的，与pages、window同一级别，代码如下：

**"tabBar":{**

**"color": "#999",**

**"selectedColor":"#D81E06",**

**"backgroundColor":"#fafafa",**

**"position":"bottom",**

**"borderStyle":"black",**

**"list":[**

**{**

**"pagePath":"pages/index/index",**

**"text":"首页",**

**"iconPath":"icons/home.png",**

**"selectedIconPath":"icons/home-o.png"**

**},{**

**"pagePath":"pages/category/category",**

**"text":"分类",**

**"iconPath":"icons/category.png",**

**"selectedIconPath":"icons/category-o.png"**

**},{**

**"pagePath":"pages/cart/cart",**

**"text":"购物车",**

**"iconPath":"icons/cart.png",**

**"selectedIconPath":"icons/cart-o.png"**

**},{**

**"pagePath":"pages/user/user",**

**"text":"我的",**

**"iconPath":"icons/my.png",**

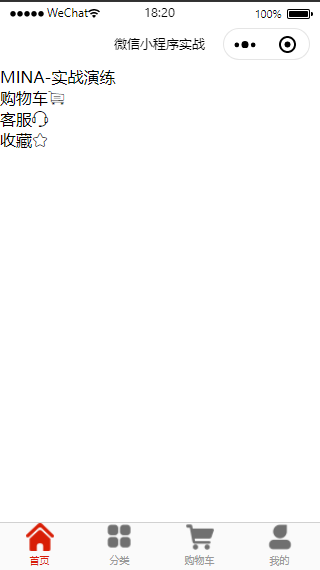
**"selectedIconPath":"icons/my-o.png"**

**}**

**]**

**},**

最终的样式如下:



## 初始化页面样式

这里的初始化是指全局的样式初始化设置，在入口文件APP.WXSS中进行操作。

### 通配的标签样式

通配的标签样式，既是适用于各种标签的样式，但是在微信小程序中并没有通配符这个骚操作，因此只能一个一个的定义了，代码如下所示：

**@import"./styles/iconfont.wxss";**

**/\* 微信小程序中不支持通配符 '\*'，所以使用一些大标签来统一设定 \*/**

**page,view,text,swiper,swiper-item,image,navigator{**

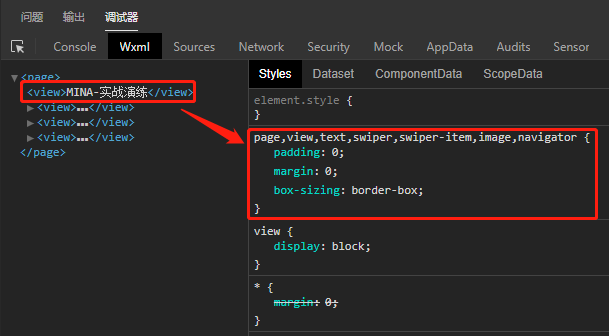
**padding: 0;**

**margin: 0;**

**box-sizing: border-box;**

**}**

上面的代码编译后查看对应的标签，可以看到如下所示的样式：



### 页面风格全局初始化

#### 颜色风格

主题风格初始化也是在APP.WXSS中进行操作，在文件中加入如下的代码：

**/\***

**主题颜色通过变量来实现**

**less中存在变量的知识，**

**原生的css、wxss中也支持变量**

**\*/**

**page{**

**/\*主题颜色变量\*/**

**--themeColor:#eb4450;**

**}**

上面的代码注释中写的很清晰，这个实用变量来进行设定的，就是说找个字符串提前给它设定好某种意义，然后这个字符串就可以称之为是个变量（Java中我更喜欢称它为形参、对象名）；然后如上代码设定好了后到页面的wxss文件中进行使用，代码如下所示：

**view{**

**/\* 引用主題顏色 \*/**

**color:var(--themeColor)**

**}**

上面的代码的含义就是，凡view标签中的字体颜色均为之前设定的主题颜色，再写一段page的wxml中的代码，如下看效果：

**<view>MINA-实战演练</view>**

**<view>购物车<text class="iconfont icon-che view"></text></view>**

**<view>客服<text class="iconfont icon-kefu"></text></view>**

**<view>收藏<text class="iconfont icon-shoucang"></text></view>**

查看效果如下图所示，颜色均发生了变化（正常情况下肯定不能如此大面积的使用，仅仅会作为一个类似于约定俗成的样式，当涉及到特殊的地方时统一要求用这种样式即可）；



#### 字体大小

直接在注意颜色变量的代码下加上如下代码：

**/\***

**统一字体大小，假设屏幕宽375px，**

**1px=2rpx  14px=28rpx 可根据屏幕大小而发生自适应大小变化**

**\*/**

**font-size: 28rpx;**

很明显能看出来，这个是直接定义的全局，没有想颜色风格那样使用变量来处理，而且这个不像衍射风格那样需要在被使用的地方的wxss文件中声明，字体大小的设定是直接生效的，只需要在app.wxss文件中设定好就可以了，上面的代码效果如下所示（这个字体大小设定使用的是rpx，是自适应式的，会根据屏幕变化而变化）：



#### 小程序window风格

一个微信小程序的风格必须保持一致，特别是window头部的颜色（字体颜色、背景颜色），如下图所示，根据视频设计的风格如下，window部分为红色（#eb4450），字体颜色为白色；



window部分为红色（#eb4450），字体颜色为白色，在APP.JSON中的window代码块中进行修改，window的代码如下所示：

**"window":{**

**"backgroundTextStyle":"light",**

**"navigationBarBackgroundColor": "#eb4450",**

**"navigationBarTitleText": "微信小程序实战",**

**"navigationBarTextStyle":"white"**

**},**

## 搜索框

最终要达到的样式目标如下图所示：

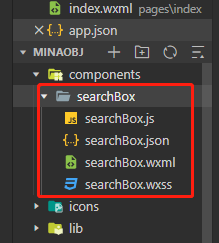


一般的购物应用中（例如淘宝、京东的首页最高处上就是一个搜索框），而且这个搜索框是重复出现的，也就是说是可以被抽离出来作为一个组件的，因此，做一个搜索框组件并关联到相应使用位置，实际操作如下；

### 创建组件模块并引入到首页中

* 创建组件

创建模块的位置就在之前创建的components文件夹下，命名为searchBox，创建时使用微信小程序开发工具来创建，这样可以使用其中内嵌的创建组件的功能，比较省心，创建完如下所示：



* 引入到首页

创建的组件通过相对路径引入到首页的中，主要通过两步，分别是将组件通过相对路径与首页的json文件中、在首页的wxml文件中使用组件名作为标签，其代码分别如下所示：

相对路径的引入（在index.json文件中编辑）：

**{**

**"usingComponents": {**

**"searchBox":"../../components/searchBox/searchBox"**

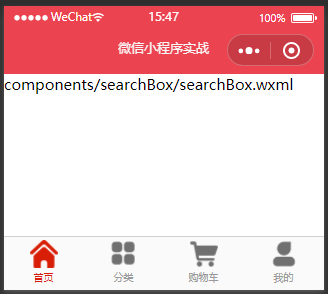
**}**

**}**

组件名作为标签使用（在index.wxml文件中编辑）

**<searchBox></searchBox>**

最终显示如下，能看到下图中的路径信息是搜索框的，引入成功；



### 完善组件样式

对组件进行内容填充，是指达到正常的样式；

#### searchBox.wxml文件中代码

在组件的searchBox.wxml文件中编辑代码，代码如下所示：

**<view class="search\_input">**

**<navigator url="../../pages/search/search" open-type="navigate">**

**<image src="../../icons/sousuo.png"/>搜索**

**</navigator>**

**</view>**

在navigator标签中设定一个url路径，定义点击这个超链接跳转到search页面（之前生成的一堆page文件中的某一个），跳转方式为navigate即可；

【提示：image框中的放大镜图片链接是另外下载的，由于组件中wxss不支持样式（wxss）图标，所以使用图片的形式来达到出现放大镜的效果；】

#### searchBox.js文件中代码

对照searchBox.wxml中

**.search\_input {**

**height: 90rpx;**

**padding: 10rpx;**

**background-color: var(--themeColor);/\*\* 主题颜色 \*\*/**

**}**

**.search\_input navigator {**

**height: 100%;**

**display: flex; /\*\* 将navigator盒子变成一个可伸缩的 \*\*/**

**justify-content: center; /\*\* 文字水平居中 \*\*/**

**align-items: center;  /\*\* 文字垂直居中 \*\*/**

**background-color: #fff;  /\*\* 白色背景 \*\*/**

**border-radius: 15rpx;  /\*\* 盒子圆角 \*\*/**

**color: #666;**

**}**

**.search\_input navigator image {**

**height: 38rpx;**

**width: 38rpx;**

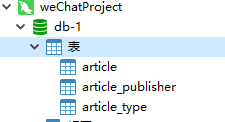
**}**

## 轮播图

# 后端

## 数据库搭建

采用MySQL数据库，Navicat数据库视图工具，主要表如下：



详细表中字段截图如下：

* Article表：文章表

建表Sql:

**DROP TABLE IF EXISTS `article`;**

**CREATE TABLE `article`  (**

**`UUID` varchar(32) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '表主键 - uuid',**

**`TYPE\_UUID` varchar(32) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '关联的分类表主键 - uuid',**

**`IMAGE\_TYPE` varchar(1) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '文章图片显示类型（0:无图，1:一张，3:三张）',**

**`IMAGE\_PATH` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '当image\_type不为\"0\"时，image\_path不为空；\r\n[“”，“”]  json串的形式',**

**`TITLE` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '文章标题',**

**`CONTENT` text CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '文章内容',**

**`PUBLISHER\_UUID` varchar(32) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '关联的发布者表主键 - uuid',**

**`PUBLISH\_TIME` datetime(0) NOT NULL COMMENT '发布时间',**

**`STATUS` char(1) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '文章状态；审核未通过:0;审核已通过:1;  强制下架:2;',**

**PRIMARY KEY (`UUID`) USING BTREE**

**) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;**

**SET FOREIGN\_KEY\_CHECKS = 1;**

* Article\_publisher：文章发布者表

建表Sql:

**DROP TABLE IF EXISTS `article\_publisher`;**

**CREATE TABLE `article\_publisher`  (**

**`PUBLISHER\_UUID` varchar(32) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '表主键 - uuid',**

**`PUBLISHER\_NAME` varchar(16) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '发布者名称',**

**`PUBLISHER\_IMAGE` varchar(255) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL COMMENT '发布者头像路径',**

**`STATUS` varchar(1) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '发布者状态（1:启用，0:停用），停用时不可发布文章且之前文章不给予展示',**

**PRIMARY KEY (`PUBLISHER\_UUID`) USING BTREE**

**) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;**

**SET FOREIGN\_KEY\_CHECKS = 1;**

* Article\_type：文章类型表

建表Sql:

**DROP TABLE IF EXISTS `article\_type`;**

**CREATE TABLE `article\_type`  (**

**`ARTICLETYPE\_UUID` varchar(32) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '表主键 - uuid',**

**`ARTICLETYPE\_NAME` varchar(16) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '文章类型名称',**

**`STATUS` char(1) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL COMMENT '类型状态（1:启用，0:停用），停用时不给予显示相应文章；',**

**PRIMARY KEY (`ARTICLETYPE\_UUID`) USING BTREE**

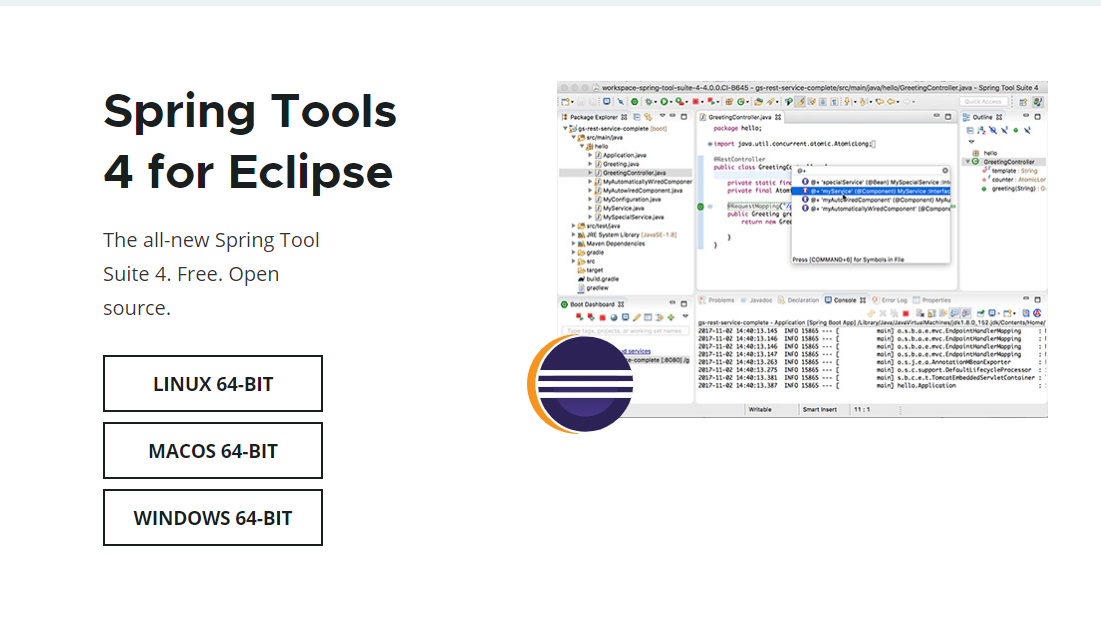
**) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8\_general\_ci ROW\_FORMAT = Dynamic;**

**SET FOREIGN\_KEY\_CHECKS = 1;**

## 后端SpringBoot开发

### 开发工具

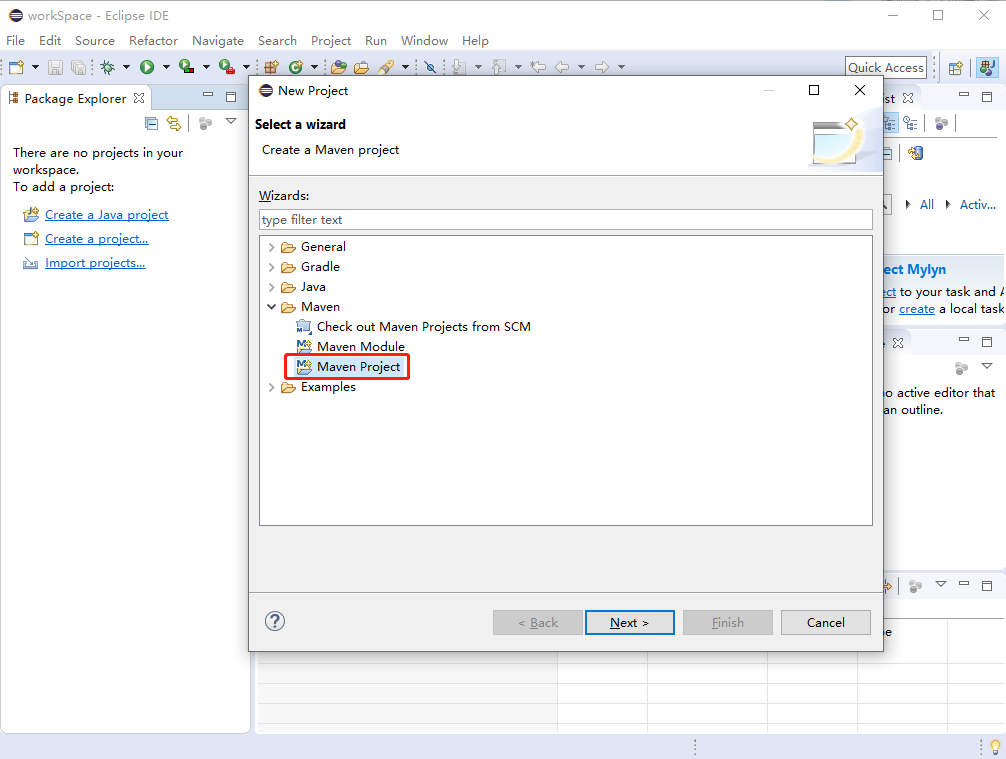
使用spring提供的eclipse开发工具，下载地址：<https://spring.io/tools>，或者使用idea也可以；



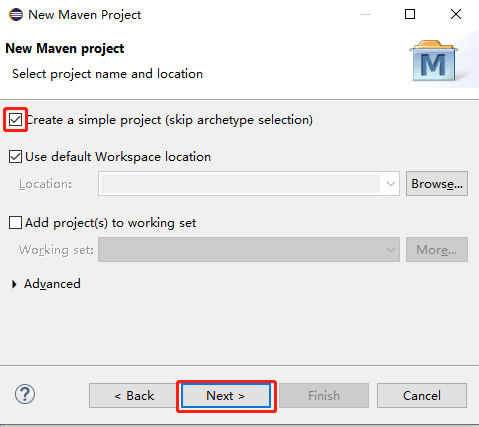
下载下来安装上即可；

### SpringBoot框架搭建

创建工程，依次点击：file > New > Project，进入如下所示的界面：

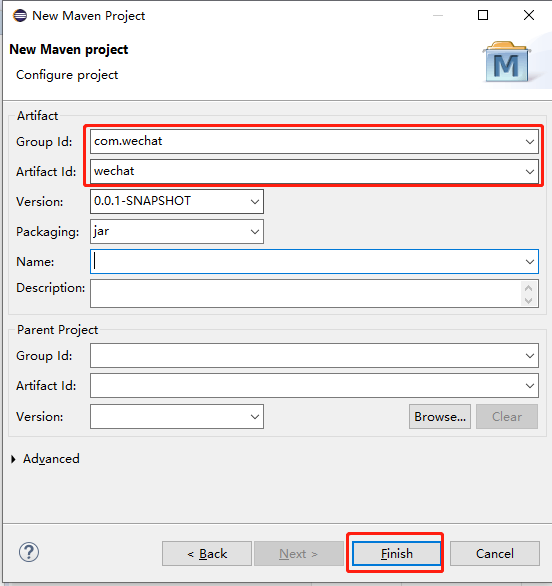


选择如上图的“Maven Project”项目后点击“Next >” 进入如下界面：



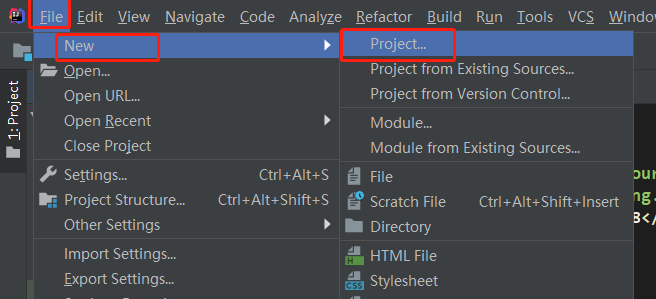
上图中只需要将第一个选择框勾选上即可，其他的不必修改，然后下一步；

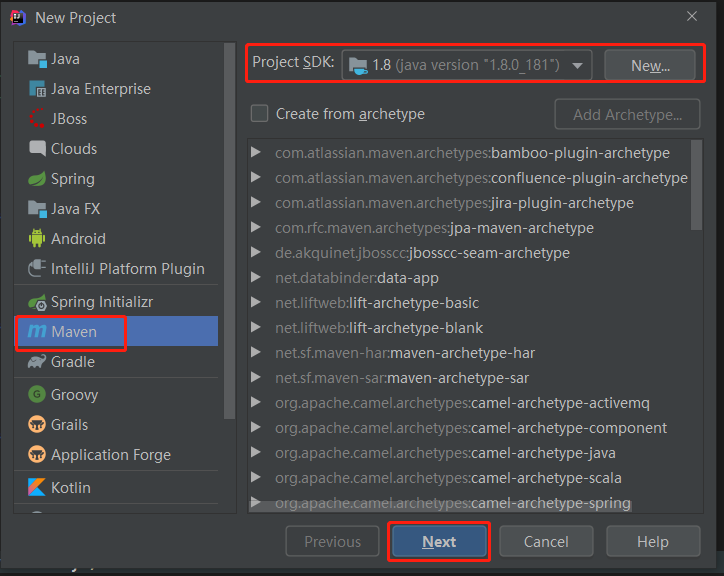
此步骤中填写的时项目模块的信息，如下所示：

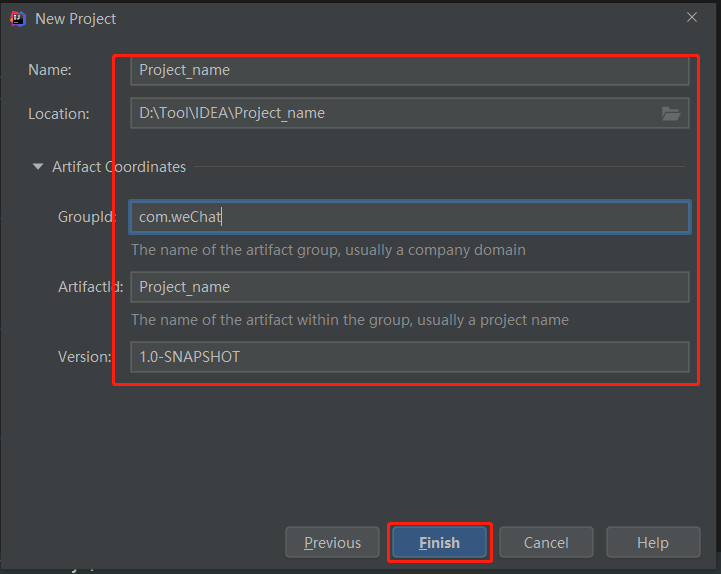


填写完必须的两项后即可点击“Finish”完成创建；

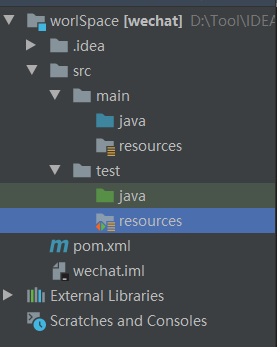
Idea中创建项目的流程也是差不多的，主要流程如下：







如此即可创建相应的项目了，但是需要注意的是，创建的项目在test文件夹中会少一个“resources”文件夹，可以给他创建一个出来即可；创建完的项目如下所示：



### 导入依赖(POM.XML)

导入的一般有这么几个模块的：数据持久层框架MyBatis模块、Apache模块、Log日志模块等等，具体的需要的大概如下：

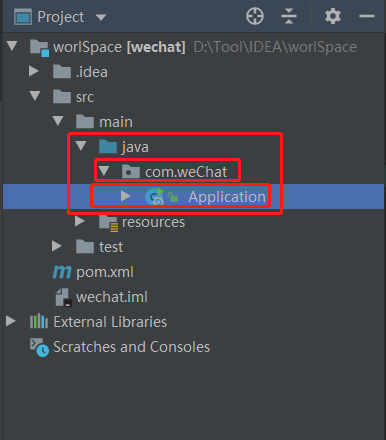
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>**4.0.0**</modelVersion>  
 <groupId>**com.weChat**</groupId>  
 <artifactId>**weChat**</artifactId>  
 <version>**0.0.1-SNAPSHOT**</version>  
 <name>**weChatProject**</name>  
 <description>**weChatProject SpringBoot**</description>  
  
 <parent>  
 <groupId>**org.springframework.boot**</groupId>  
 <artifactId>**spring-boot-starter-parent**</artifactId>  
 <version>**2.1.5.RELEASE**</version>  
 <relativePath/>  
 </parent>  
  
 <dependencies>  
 <dependency>  
 <groupId>**org.springframework.boot**</groupId>  
 <artifactId>**spring-boot-starter**</artifactId>  
 <version>**1.3.1.RELEASE**</version>  
 </dependency>  
  
 <dependency>  
 <groupId>**org.springframework.boot**</groupId>  
 <artifactId>**spring-boot-starter-web**</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>**org.springframework.boot**</groupId>  
 <artifactId>**spring-boot-configuration-processor**</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>**mysql**</groupId>  
 <artifactId>**mysql-connector-java**</artifactId>  
 </dependency>  
  
 <!--mybatis依赖-->  
 <dependency>  
 <groupId>**org.mybatis.spring.boot**</groupId>  
 <artifactId>**mybatis-spring-boot-starter**</artifactId>  
 <version>**2.1.3**</version>  
 </dependency>  
  
 <!-- mybatis 反向自动生成依赖 -->  
 <dependency>  
 <groupId>**org.mybatis.generator**</groupId>  
 <artifactId>**mybatis-generator-core**</artifactId>  
 <version>**1.3.7**</version>  
 </dependency>  
  
 <dependency>  
 <groupId>**tk.mybatis**</groupId>  
 <artifactId>**mapper**</artifactId>  
 <version>**4.1.5**</version>  
 </dependency>  
  
 <!-- 分页插件 -->  
 <dependency>  
 <groupId>**com.github.pagehelper**</groupId>  
 <artifactId>**pagehelper-spring-boot-starter**</artifactId>  
 <version>**1.2.3**</version>  
 </dependency>  
  
 <!-- apache 工具类 -->  
 <dependency>  
 <groupId>**commons-codec**</groupId>  
 <artifactId>**commons-codec**</artifactId>  
 <version>**${commons-codec.version}**</version>  
 </dependency>  
  
 <dependency>  
 <groupId>**org.apache.commons**</groupId>  
 <artifactId>**commons-lang3**</artifactId>  
 <version>**${commons-lang3.version}**</version>  
 </dependency>  
  
 <dependency>  
 <groupId>**org.apache.commons**</groupId>  
 <artifactId>**commons-io**</artifactId>  
 <version>**1.3.2**</version>  
 </dependency>  
  
 <!--logger日志 spring boot 默认日志是较新的logback 日志，不用再引入slf4j 日志了-->  
  
 </dependencies>  
  
 <properties>  
 <project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>**UTF-8**</project.reporting.outputEncoding>  
 <java.version>**1.8**</java.version>  
 </properties>  
  
 <build>  
 <!-- 插件 -->  
 <plugins>  
 <!-- maven插件 -->  
 <plugin>  
 <groupId>**org.springframework.boot**</groupId>  
 <artifactId>**spring-boot-maven-plugin**</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

引入完这些模块后eclipse开发工具中的POM.XML文件会报错，只需要右击报错的POM.XML文件，选择功能面板中的Maven功能中的Update Project…即可，等更新完依赖就不会再报错了；

### 项目启动准备

#### 创建项目启动类: Application.java

在项目的如下图所示的位置创建com.weChat文件夹并在其中创建Application.java文件：

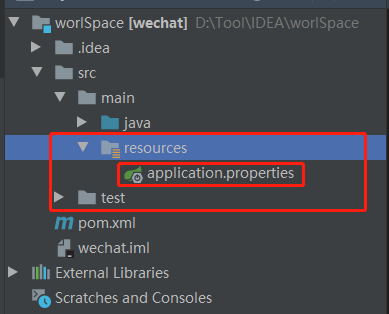


application.java文件中代码如下：

package com.weChat;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.ComponentScan;  
import tk.mybatis.spring.annotation.MapperScan;  
  
// 当前项目启动类  
@SpringBootApplication // spring boot 支持 移动应用的注解  
// 当前启动类所扫描文件位置，当有SpringBootApplication注解时需要注意此注解的使用  
@ComponentScan("com.weChat.\*")  
// 扫描 Mapper 范围定义  
@MapperScan("com.weChat.mapper")  
public class Application {  
  
 // 启动类主方法 - 启动方法就是其本身（及其方法内引用、关联的方法模块）  
 public static void main(String[] args) {  
  
 SpringApplication.run(Application.class, args);  
  
 }  
}

#### 创建并配置文件:application.properties

在如下的路径中选择New 中的 file，将文件名填入创建即可:

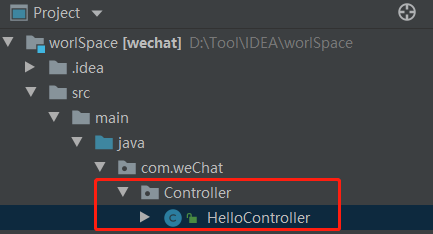


这个文件的左右就是配置一些SQL链接信息、JPA配置信息、服务器配置信息等，具体配置如下：

# server 服务端相关配置  
server.port=8080  
# server - tomcat 常见相关配置  
server.tomcat.uri-encoding=utf-8  
  
# mysql连接配置文件  
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/db-1?useUnicode=true&characterEncoding=UTF8  
spring.datasource.username=root  
spring.datasource.password=root123  
spring.datasource.driverClassName=com.mysql.jdbc.Driver  
  
# 连接池  
# 等待连接池分配连接的最大时长（毫秒），超过这个时间仍没连接则发生SQLException,默认30s  
spring.datasource.hikari.connection-timeout=30000  
# 最小连接数  
spring.datasource.hikari.minimum-idle=5  
# 最大连接数  
spring.datasource.hikari.maximum-pool-size=15  
# 自动提交  
spring.datasource.hikari.auto-commit=true  
# 一个链接 idle状态最大时长设置、超时释放时间（默认10mis）  
spring.datasource.hikari.idle-timeout=600000  
# 连接池名称  
spring.datasource.hikari.pool-name=DatebookHikariCP  
# 一个链接的生命市场（毫秒），超时且未被使用怎会被释放，默认30mis  
spring.datasource.hikari.max-lifetime=28740000  
  
# Mybatis配置  
mybatis.type-aliases-package=com.weChat.Entry  
mybatis.mapper-locations=classpath:mapper/\*.xml  
  
# 通用Mapper配置  
mapper.mappers=com.weChat.Mapper.Mapper  
mapper.not-empty=false  
mapper.identity=MySQL  
  
# 分页插件配置  
pagehelper.helper-dialect=mysql  
# pagehelper.reasonable=true  
pagehelper.support-methods-arguments=true  
pagehelper.params=count=countSql  
  
# JPA配置信息  
spring.jpa.database=MYSQL  
# 是否显示每个SQL的日志  
spring.jpa.show-sql=true  
# Hibernate ddl auto (create, create-drop, update)  
spring.jpa.hibernate.ddl-auto=update  
# Naming strategy  
#[org.hibernate.cfg.ImprovedNamingStrategy #org.hibernate.cfg.DefaultNamingStrategy]  
spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy  
# stripped before adding them to the entity manager)  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

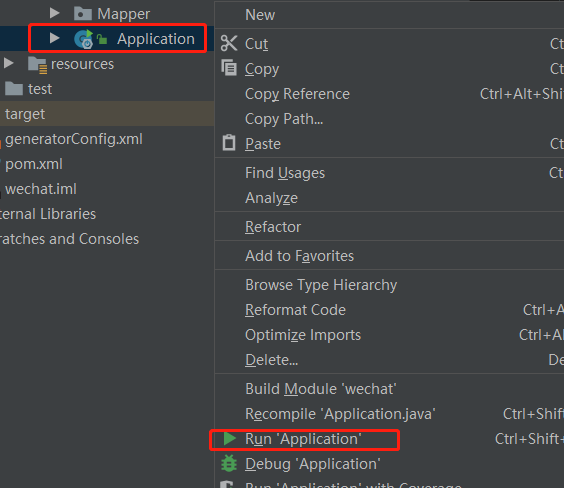
#### 创建请求文件: HelloController.java

HelloController.java文件创建位置如下图所示（创建一个controller文件夹，然后再在这个文件夹中创建这个controller方法文件）：

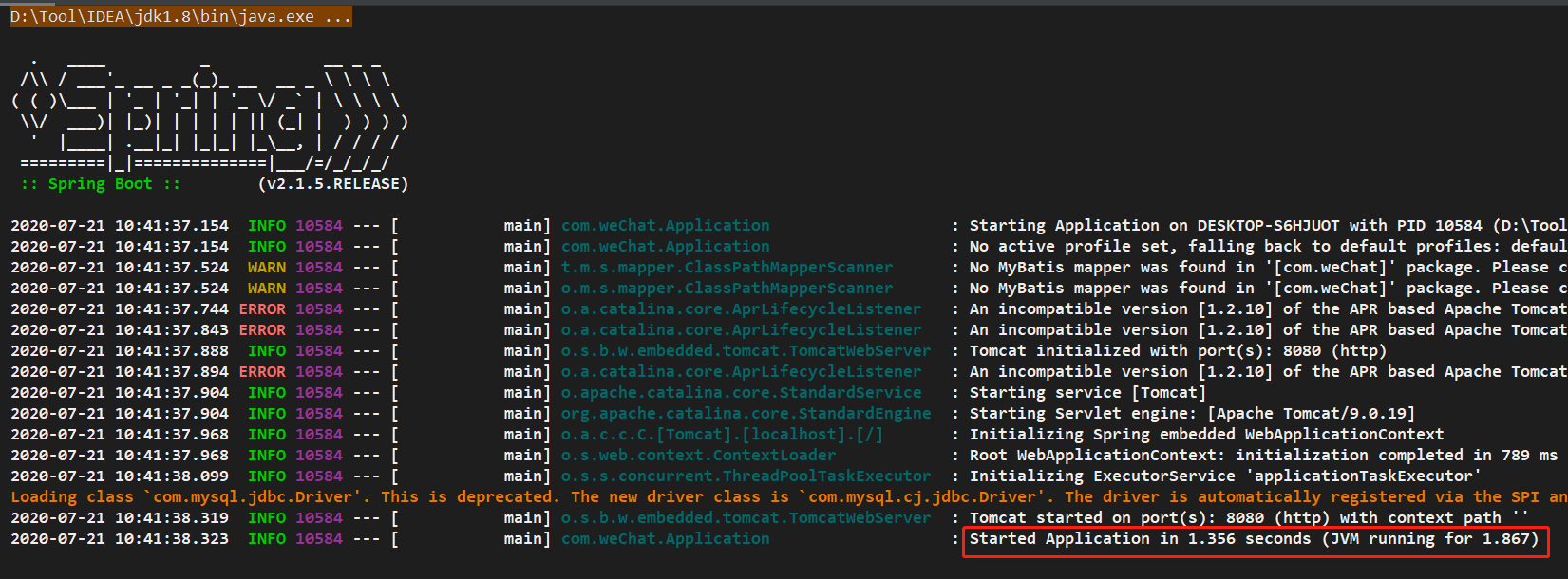


package com.weChat.Controller;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HelloController {  
 @GetMapping("/helloMapping")  
 public String hello(){  
 return "你好啊！";  
 }  
}

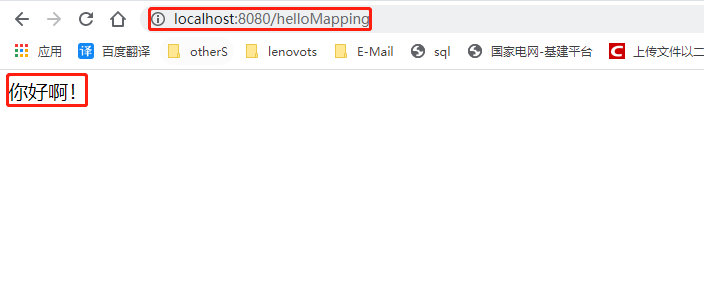
配置此文件完成后项目即算初步搭建完成，可以尝试启动项目，首先使用Application启动项文件进行启动，如下：



启动时工作台日志信息如下：



工作台日志最后一行出现如图片中框住的信息技术与启动成功，然后再到浏览器中进行访问controller类中的方法，以上面代码为例，输入的路径时：localhost:8080/ helloMapping，8080是之前在application.properties中配置的服务端口号，helloMapping是controller方法中的getMapping定义的请求此方法的路径，结果如下：



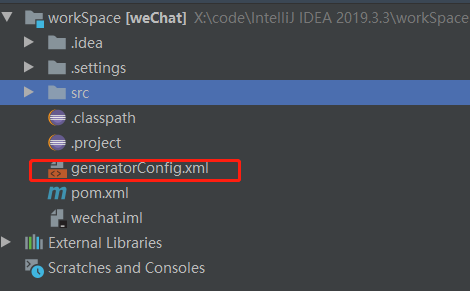
### 自动生成生成Entry与Mapper

项目已经可以跑起来了，但是对于数据而言还未开始进行勾连，需要一个完整的数据引入流通的体系，因此，我们在使用mybatis数据持久层框架时就需要用到Entry和Mapper这两文件，前者是数据库表字段的映射对象，Mapper文件是数据持久层中的数据操作文件，说白了它主要就是放SQL语句的地方，而这两个文件都是可以通过springbooot中的代码生成器进行完成的，所以，我们只需要配置这个代码自动生成器及其相关文件即可；

#### 创建代码自动生成器：generatorConfig.xml

代码自动生成器（generatorConfig.xml），是 mybatis-generator-maven-plugin插件的配置文件，配置该插件，用于连接数据库自动生成mybatis需要的代码文件；

创建位置：项目的pom.xml文件同层级



其中代码如下：

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE generatorConfiguration  
 PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  
 "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  
<!-- 配置生成器 -->  
<generatorConfiguration>  
 <context id="mysqlContext" defaultModelType="flat" targetRuntime="MyBatis3Simple">  
  
 <!-- beginningDelimiter和endingDelimiter：指明数据库的用于标记数据库对象名的符号，比如ORACLE就是双引号，MYSQL默认是`反引号； -->  
 <property name="beginningDelimiter" value="`"/>  
 <property name="endingDelimiter" value="`"/>  
  
 <!-- MyMapper 工具类引入 -->  
 <plugin type="tk.mybatis.mapper.generator.MapperPlugin">

<!--注意这里需要预先指定写好的通用mapper接口,自动生成的接口全部继承这个接口,并且在扫描mapper的时候,这个类不能被扫描到的 -->  
 <property name="mappers" value="com.weChat.Utils.MainMapper

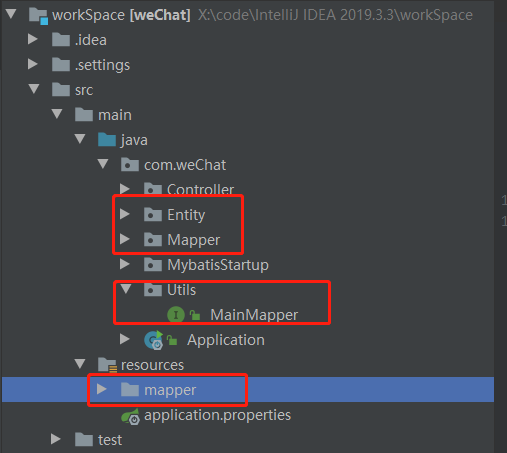
"/>  
 </plugin>  
  
 <!-- 必须要有的，使用这个配置链接数据库 -->  
 <jdbcConnection driverClass="com.mysql.cj.jdbc.Driver"  
 connectionURL="jdbc:mysql://127.0.0.1:3306/db-1?useUnicode=true&amp;characterEncoding=UTF-8&amp;serverTimezone=Asia/Shanghai"  
 userId="root" password="root123">  
 <!-- 这里面可以设置property属性，每一个property属性都设置到配置的Driver上 -->  
 </jdbcConnection>  
  
 <!-- java模型创建器，是必须要的元素  
 负责：1，key类（见context的defaultModelType）；2，java类；3，查询类  
 targetPackage：生成的类要放的包，真实的包受enableSubPackages属性控制；  
 targetProject：目标项目，指定一个存在的目录下，生成的内容会放到指定目录中，如果目录不存在，MBG不会自动建目录  
 -->  
 <javaModelGenerator targetPackage="com.weChat.Entity" targetProject="src/main/java">  
 <!-- 从数据库返回的值被清理前后的空格 -->  
 <property name="trimStrings" value="true"/>  
 <property name="comments" value="true"/>  
 </javaModelGenerator>  
  
 <!-- 生成SQL map的XML文件生成器，  
 注意，在Mybatis3之后，我们可以使用mapper.xml文件+Mapper接口（或者不用mapper接口），  
 或者只使用Mapper接口+Annotation，所以，如果 javaClientGenerator配置中配置了需要生成XML的话，这个元素就必须配置  
 targetPackage/targetProject:同javaModelGenerator  
 -->  
 <sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources"/>  
  
 <!-- 对于mybatis来说，即生成Mapper接口，注意，如果没有配置该元素，那么默认不会生成Mapper接口  
 targetPackage/targetProject:同javaModelGenerator  
 type：选择怎么生成mapper接口（在MyBatis3/MyBatis3Simple下）：  
 1，ANNOTATEDMAPPER：会生成使用Mapper接口+Annotation的方式创建（SQL生成在annotation中），不会生成对应的XML；  
 2，MIXEDMAPPER：使用混合配置，会生成Mapper接口，并适当添加合适的Annotation，但是XML会生成在XML中；  
 3，XMLMAPPER：会生成Mapper接口，接口完全依赖XML；  
 注意，如果context是MyBatis3Simple：只支持ANNOTATEDMAPPER和XMLMAPPER  
 -->  
 <javaClientGenerator targetPackage="com.weChat.Mapper" type="XMLMAPPER" targetProject="src/main/java"/>  
  
  
 <!-- 选择一个table来生成相关文件，可以有一个或多个table，必须要有table元素 -->  
 <table tableName="article\_type"></table>  
 <table tableName="article\_publisher"></table>  
 <table tableName="article"></table>  
  
 </context>  
  
</generatorConfiguration>

其中有几个点需要注意一下：

1. MySQL 8版本的数据库在配置connectionURL时需要对“;”转码，转成‘&amp;’;
2. MySQL 8版本的数据库在配置connectionURL时需要加上时区设置，设置如下代码：&amp;serverTimezone=Asia/Shanghai并且此时区设置没有beijing等的时区在中国国内，所以不要想这设置成北京的了；
3. 有几个目录需要手动创建一下

* 存放SQL Map的XML文件的mapper文件夹（在resources中）
* 存放java模型Entry文件的Entity文件夹
* 存放mybatis特有的Mapper接口文件的Mapper文件夹
* 存放‘自动生成mapper实现类继承接口文件’的MainMapper文件夹

目录结构如下图：

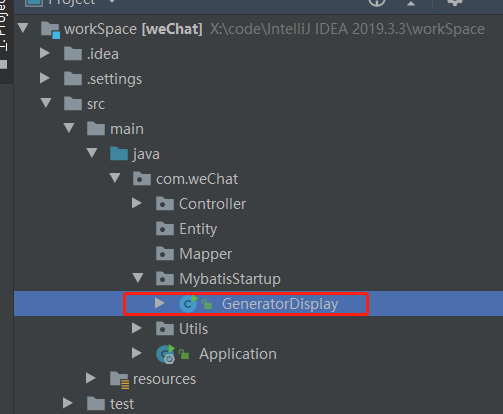


* <table>标签中的‘tableName’要对应数据库中的表名；

#### 创建代码生成器启动类：GeneratorDisplay

启动类：GeneratorDisplay，它是个class类。这个类是用于一键启动运行generatorConfig.xml文件的，当运行启动类时根据其中代码会调用指定的generatorConfig.xml文件，然后生成对应的实体类文件、mapper实现类文件和mapper的sql映射文件；

启动类位置：（不做强求，启动类再src/main下即可，我的如下：）

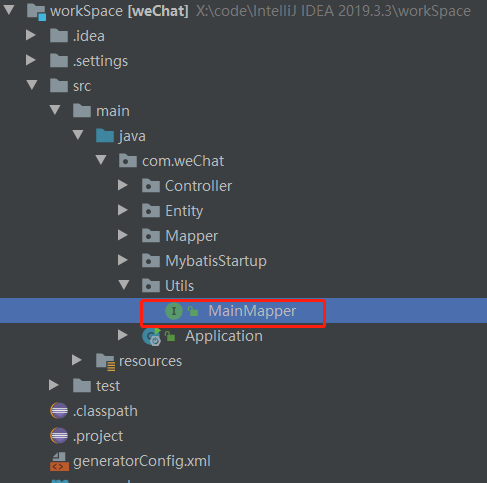


启动类代码：

package com.weChat.MybatisStartup;  
  
import org.mybatis.generator.api.MyBatisGenerator;  
import org.mybatis.generator.config.Configuration;  
import org.mybatis.generator.config.xml.ConfigurationParser;  
import org.mybatis.generator.internal.DefaultShellCallback;  
  
import java.io.File;  
import java.util.ArrayList;  
import java.util.List;  
  
public class GeneratorDisplay {  
  
 public static void main(String[] args) {  
 try {  
 GeneratorDisplay generatorSqlmap = new GeneratorDisplay();  
 generatorSqlmap.generator();  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
  
 public void generator() throws Exception {  
 List<String> warnings = new ArrayList<String>();  
 boolean overwrite = true;  
  
 // 指定 逆向工程配置  
 File configfile = new File("generatorConfig.xml");  
 ConfigurationParser cp = new ConfigurationParser(warnings);  
 Configuration config = cp.parseConfiguration(configfile);  
 DefaultShellCallback callback = new DefaultShellCallback(overwrite);  
 MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config, callback, warnings);  
 myBatisGenerator.generate(null);  
 }  
  
}

#### 自动生成mapper实现类通用继承类：MainMapper

这个类为了那些自动生成的mapper实现类用于继承的通用接口，所有的自动生成的mapper实现类都需要继承这个接口，存放位置在generatorConfig.xml文件中配置的有，并且之前有创建，如下所示：



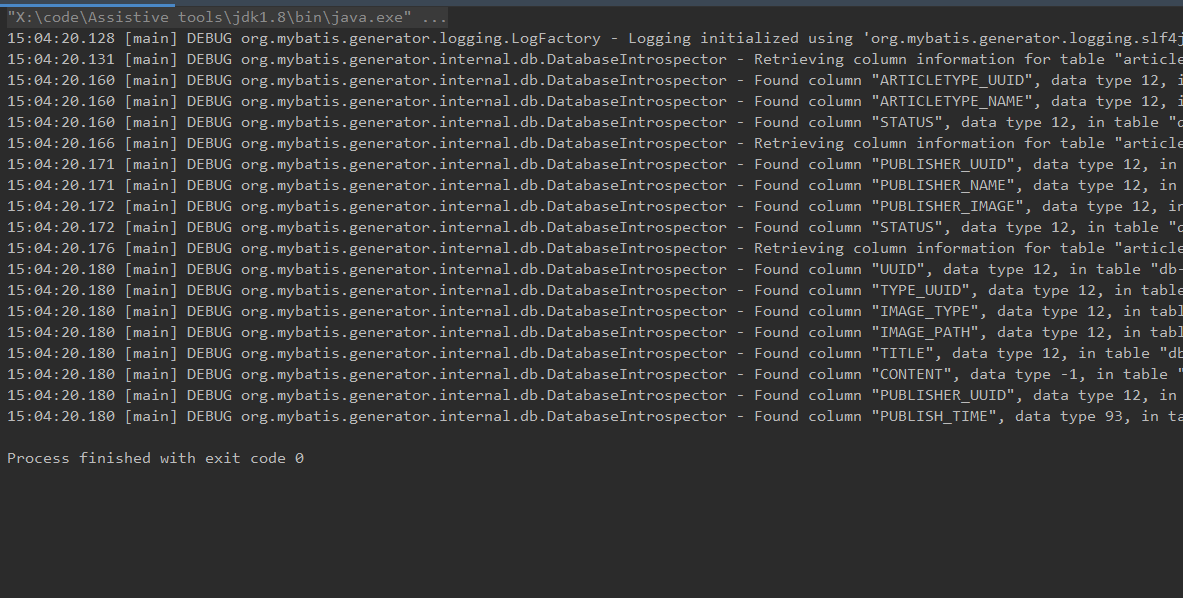
代码如下：

package com.weChat.Utils;  
  
import tk.mybatis.mapper.common.Mapper;  
import tk.mybatis.mapper.common.MySqlMapper;  
  
// 注意这里需要预先指定写好的通用mapper接口,自动生成的接口全部继承这个接口  
public interface MainMapper<T> extends Mapper<T>, MySqlMapper<T> {  
  
}

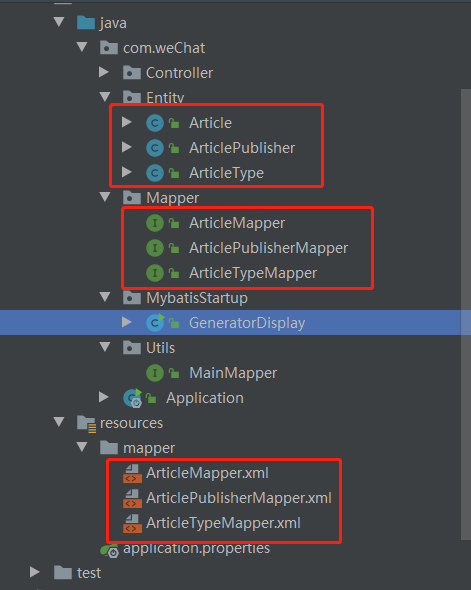
作为一个通用的被继承的接口，其中代码肯定是越少越好，越多越容易对后边的代码产生影响；

#### 开始自动生

经过上述操作，已经可以对数据库的表进行映射来生成成实体类、mapper实现类、mapper SQL文件了，右击GeneratorDisplay，点击运行，控制台日志栏出现如下日志代码：



即算成功！查看结构目录，会发现相应文件生成，如下图：



### 自定义响应数据结构

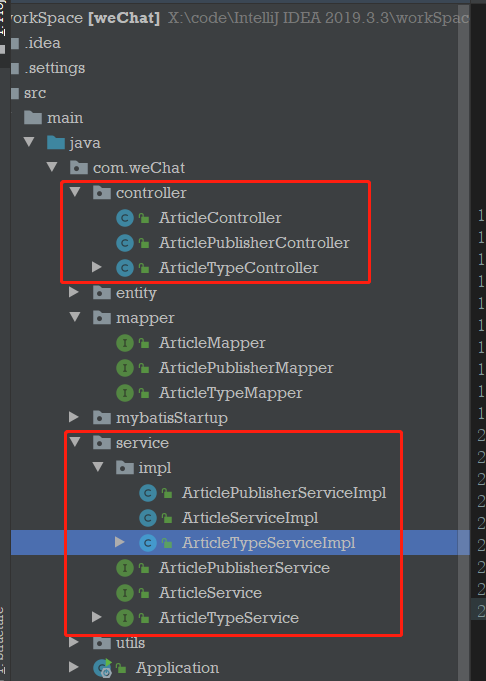
这个自定义响应数据结构的意思就是说，写一个工具类来把所有的方法返回结果集都走这个工具类，然后会有一些统一的元素最终返给前台，就比如接口调用结果（是否成功）、返回结果数据、错误数据等，省的每个接口都要写一些类似的数据，而且每个都写的情况下特别是多人开发，会出现定义的属性名各式各样，不便于后期维护和前端的操作，具体代码如下：

package com.weChat.utils;  
  
  
import com.fasterxml.jackson.databind.JsonNode;  
import com.fasterxml.jackson.databind.ObjectMapper;  
  
import java.util.List;  
  
/\*\*  
 \* @ClassName: JSONResult  
 \* @Description: 《自定义响应数据结构》  
 \* 这个类是提供给门web，ios，安卓，微信公众号、小程序用的  
 \* 门户接受此类数据后需要使用本类的方法转换成对于的数据类型格式（类，或者list）  
 \* 其他自行处理  
 \* 200：表示成功  
 \* 500：表示错误，错误信息在msg字段中  
 \* 501：bean验证错误，不管多少个错误都以map形式返回  
 \* 502：拦截器拦截到用户token出错  
 \* 555：异常抛出信息  
 \* @Author: QiTL  
 \* @Date: 2020/8/3 15:47  
 \*\*/  
public class JSONResult {  
 // 定义jackson对象  
 private static final ObjectMapper ***MAPPER*** = new ObjectMapper();  
  
 // 响应业务状态  
 private Integer status;  
  
 // 响应消息  
 private String msg;  
  
 // 响应中的数据  
 private Object data;  
  
 public JSONResult() {  
  
 }  
  
 public JSONResult(Integer status, String msg, Object data) {  
 this.status = status;  
 this.msg = msg;  
 this.data = data;  
 }  
  
 public JSONResult(Object data) {  
 this.status = 200;  
 this.msg = "OK";  
 this.data = data;  
 }  
  
 public static JSONResult build(Integer status, String msg, Object data) {  
 return new JSONResult(status, msg, data);  
 }  
  
 public static JSONResult ok(Object data) {  
 return new JSONResult(data);  
 }  
  
 public static JSONResult ok() {  
 return new JSONResult(null);  
 }  
  
 public static JSONResult errorMsg(String msg) {  
 return new JSONResult(500, msg, null);  
 }  
  
 public static JSONResult errorMap(Object data) {  
 return new JSONResult(501, "error", data);  
 }  
  
// public static LeeJSONResult build(Integer status, String msg) {  
// return new LeeJSONResult(status, msg, null);  
// }  
  
 public static JSONResult errorTokenMsg(String msg) {  
 return new JSONResult(502, msg, null);  
 }  
  
 public static JSONResult errorException(String msg) {  
 return new JSONResult(555, msg, null);  
 }  
  
 /\*\*  
 \* @param jsonData  
 \* @param clazz  
 \* @return  
 \* @Description: 将json结果集转化为LeeJSONResult对象  
 \* 需要转换的对象是一个类  
 \* @author leechenxiang  
 \* @date 2016年4月22日 下午8:34:58  
 \*/  
 public static JSONResult formatToPojo(String jsonData, Class<?> clazz) {  
 try {  
 if (clazz == null) {  
 return ***MAPPER***.readValue(jsonData, JSONResult.class);  
 }  
 JsonNode jsonNode = ***MAPPER***.readTree(jsonData);  
 JsonNode data = jsonNode.get("data");  
 Object obj = null;  
 if (clazz != null) {  
 if (data.isObject()) {  
 obj = ***MAPPER***.readValue(data.traverse(), clazz);  
 } else if (data.isTextual()) {  
 obj = ***MAPPER***.readValue(data.asText(), clazz);  
 }  
 }  
 return build(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
 /\*\*  
 \* @param json  
 \* @return  
 \* @Description: 没有object对象的转化  
 \* @author leechenxiang  
 \* @date 2016年4月22日 下午8:35:21  
 \*/  
 public static JSONResult format(String json) {  
 try {  
 return ***MAPPER***.readValue(json, JSONResult.class);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 /\*\*  
 \* @param jsonData  
 \* @param clazz  
 \* @return  
 \* @Description: Object是集合转化  
 \* 需要转换的对象是一个list  
 \* @author leechenxiang  
 \* @date 2016年4月22日 下午8:35:31  
 \*/  
 public static JSONResult formatToList(String jsonData, Class<?> clazz) {  
 try {  
 JsonNode jsonNode = ***MAPPER***.readTree(jsonData);  
 JsonNode data = jsonNode.get("data");  
 Object obj = null;  
 if (data.isArray() && data.size() > 0) {  
 obj = ***MAPPER***.readValue(data.traverse(),  
 ***MAPPER***.getTypeFactory().constructCollectionType(List.class, clazz));  
 }  
 return build(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
 public Boolean isOK() {  
 return this.status == 200;  
 }  
  
 public Integer getStatus() {  
 return status;  
 }  
  
 public void setStatus(Integer status) {  
 this.status = status;  
 }  
  
 public String getMsg() {  
 return msg;  
 }  
  
 public void setMsg(String msg) {  
 this.msg = msg;  
 }  
  
 public Object getData() {  
 return data;  
 }  
  
 public void setData(Object data) {  
 this.data = data;  
 }  
  
}

### 业务开发

#### 查询所有文章类型业务

根据之前创建的数据库数据，首先开发一个查询所有文章类型的业务，使用Article\_type 表，并且创建每个mapper接口文件对应的service层（接口、实现类）和controller层，创建结构如下所示：



创建完成后便可以开始编码；

**Controller层：ArticleTypeController.java**

package com.weChat.controller;  
  
import com.weChat.service.ArticleTypeService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("articleType")  
public class ArticleTypeController {  
  
 @Autowired  
 private ArticleTypeService articleTypeService;  
  
 /\*\*  
 \* @Description: 查询所有文章类型；  
 \* @return List<ArticleType>  
 \*/  
 @GetMapping("/queryAllType")  
 public List queryAllType(){  
 List<com.weChat.entity.ArticleType> result = articleTypeService.queryAllType();  
 return result;  
 }  
}

**Service层：**

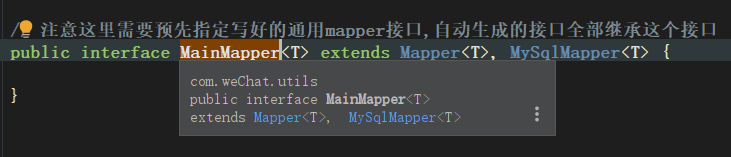
**——接口：ArticleTypeService.java**

package com.weChat.service;  
  
import com.weChat.entity.ArticleType;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public interface ArticleTypeService {  
  
 /\*\*  
 \* @Description: 查询所有文章类型；  
 \* @return List<ArticleType>  
 \*/  
 public List<ArticleType> queryAllType();  
}

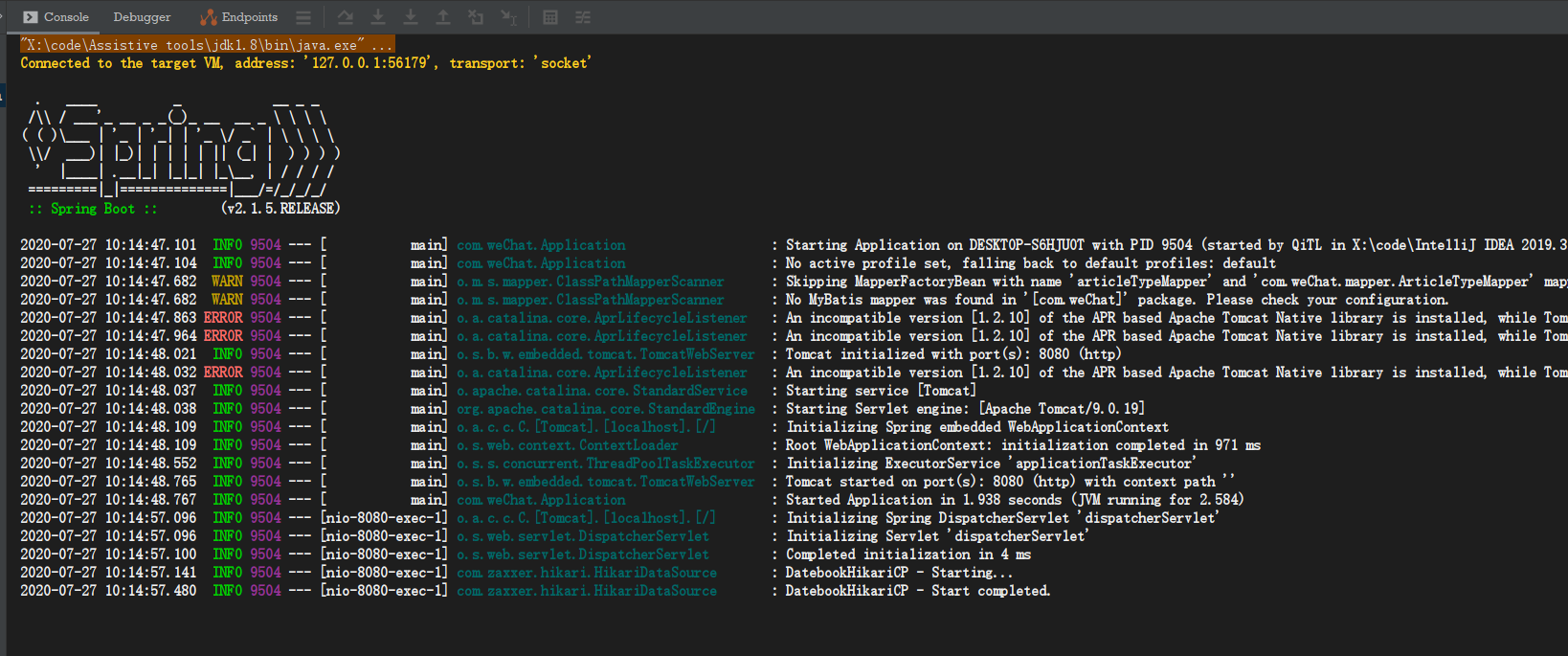
**——实现类：ArticleTypeServiceImpl.java**

package com.weChat.service.impl;  
  
import com.weChat.entity.ArticleType;  
import com.weChat.mapper.ArticleTypeMapper;  
import com.weChat.service.ArticleTypeService;  
import org.apache.ibatis.annotations.Mapper;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Service  
public class ArticleTypeServiceImpl implements ArticleTypeService {  
  
 @Autowired  
 private ArticleTypeMapper articleTypeMapper;  
  
 /\*\*  
 \* @Description: 查询所有文章类型；  
 \* @return List<ArticleType>  
 \*/  
 @Override  
 public List<ArticleType> queryAllType() {  
 List<ArticleType> EffectivearticleTypes = new ArrayList<ArticleType>();  
 List<ArticleType> articleTypes = articleTypeMapper.selectAll();  
 // 只获取有效数据：status:0  
 for (ArticleType at : articleTypes) {  
 if ("0".equalsIgnoreCase(at.getStatus())){  
 EffectivearticleTypes.add(at);  
 }  
 }  
 return EffectivearticleTypes;  
 }  
}

**List<ArticleType> articleTypes = articleTypeMapper.selectAll();这段代码使用的是自动生成的mapper实现的MainMapper的方法，MainMapper类继承了Mapper类和MySqlMapper类，而这两个类就是tk.mybatis的延伸类，其实真正使用的是封装的tk.mybatis方法，里面的方法很丰富，有兴趣的童鞋可以去深入了解一下；**



完成上面的一系列代码之后，点击运行“Application”结果如下所示：



#### 根据类型查所有文章业务

在这个业务中进行查询某一类型下的所有文章，并且配合自定义响应数据结构，其中要涉及的是文章发布者是否被封号、未审核通过或者被封的文章这两种情况，当有被封号的作者的文章时不与展示，当文章处于审核未通过或者被封时不予显示，因此这一个事务就涉及到两个表的查询，而且是相对独立的查询，首先查询出可以正常显示的文章数据，然后根据文章数据中的发布者id去查询排除被封的作者的文章，具体代码如下：

**根据发布者ID查询发布者信息：**

ArticlePublisherController.java

package com.weChat.controller;  
  
import com.weChat.entity.ArticlePublisher;  
import com.weChat.service.ArticlePublisherService;  
import com.weChat.utils.JSONResult;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/publisher")  
public class ArticlePublisherController {  
  
 @Autowired  
 private ArticlePublisherService articlePublisherService;  
  
  
 /\*\*  
 \* 功能描述:  
 \* 〈根据作者id查询作者信息〉  
 \*  
 \* @Param: [pId]  
 \* @Return: com.weChat.utils.JSONResult  
 \* @Author: https://home.cnblogs.com/u/90s-ITBoy/  
 \* @Date: 2020/8/4 11:14  
 \*\*/  
 @GetMapping("/queryPublisherByPId")  
 public JSONResult queryPublisherByPId(String pId) {  
 ArticlePublisher result = articlePublisherService.queryPublisherByPId(pId);  
 return JSONResult.ok(result);  
 }  
}

ArticlePublisherService.java

package com.weChat.service;  
  
import com.weChat.entity.ArticlePublisher;  
  
public interface ArticlePublisherService {  
  
 /\*\*  
 \* 功能描述:  
 \* 〈根据作者id查询作者信息〉  
 \*  
 \* @Param: [pId]  
 \* @Return: ArticlePublisher  
 \* @Author: https://home.cnblogs.com/u/90s-ITBoy/  
 \* @Date: 2020/8/4 11:16  
 \*\*/  
 ArticlePublisher queryPublisherByPId(String pId);  
}

ArticlePublisherServiceImpl.java

package com.weChat.service.impl;  
  
import com.weChat.entity.ArticlePublisher;  
import com.weChat.mapper.ArticlePublisherMapper;  
import com.weChat.service.ArticlePublisherService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
  
@Service  
public class ArticlePublisherServiceImpl implements ArticlePublisherService {  
  
 @Autowired  
 private ArticlePublisherMapper articlePublisherMapper;  
  
 /\*\*  
 \* 功能描述:  
 \* 〈根据作者id查询作者信息〉  
 \*  
 \* @Param: [pId]  
 \* @Return: ArticlePublisher  
 \* @Author: https://home.cnblogs.com/u/90s-ITBoy/  
 \* @Date: 2020/8/4 11:16  
 \*\*/  
 @Override  
 public ArticlePublisher queryPublisherByPId(String pId) {  
 // 查询pid所代指的作者信息  
 ArticlePublisher articlePublisher = articlePublisherMapper.selectByPrimaryKey(pId);  
 return articlePublisher;  
 }  
}

**然后开始进行下一步：**

**查询所有此类型下的有效文章：**

ArticleController.java

package com.weChat.controller;  
  
import com.weChat.service.ArticleService;  
import com.weChat.utils.JSONResult;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
import java.util.Map;  
  
@RestController  
@RequestMapping("/Article")  
public class ArticleController {  
  
 /\*\*  
 \* 日志  
 \*/  
 private static Logger logger = LoggerFactory.getLogger(ArticleController.class);  
  
 @Autowired  
 private ArticleService articleService;  
  
 /\*\*  
 \* 功能描述:  
 \* 〈查询不同类型的文章〉  
 \*  
 \* @Param: [articleTypeId]  
 \* @Return: com.weChat.utils.JSONResult  
 \* @Author: https://home.cnblogs.com/u/90s-ITBoy/  
 \* @Date: 2020/8/4 11:18  
 \*\*/  
 @GetMapping("/queryAllArticleByType/{articleTypeId}")  
 public JSONResult queryAllArticleByType(@PathVariable String articleTypeId) {  
 logger.info("根据类型查文章：queryAllArticleByType");  
 List<Map<String, Object>> result = articleService.queryAllArticleByType(articleTypeId);  
 return JSONResult.ok(result);  
 }  
}

ArticleService.java

package com.weChat.service;  
  
import java.util.List;  
import java.util.Map;  
  
public interface ArticleService {  
  
 /\*\*  
 \* 功能描述:  
 \* 〈根据类型查文章〉  
 \*  
 \* @return  
 \* @Param: [articleTypeId]  
 \* @Return: java.util.List<com.weChat.entity.Article>  
 \* @Author: https://home.cnblogs.com/u/90s-ITBoy/  
 \* @Date: 2020/8/4 11:16  
 \*/  
 List<Map<String, Object>> queryAllArticleByType(String articleTypeId);  
}

ArticleServiceImpl.java

package com.weChat.service.impl;  
  
import com.weChat.entity.Article;  
import com.weChat.entity.ArticlePublisher;  
import com.weChat.mapper.ArticleMapper;  
import com.weChat.service.ArticlePublisherService;  
import com.weChat.service.ArticleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import tk.mybatis.mapper.entity.Example;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
@Service  
public class ArticleServiceImpl implements ArticleService {  
  
 @Autowired  
 private ArticleMapper articleMapper;  
  
 @Autowired  
 private ArticlePublisherService articlePublisherService;  
  
 /\*\*  
 \* 功能描述:  
 \* 〈根据类型查文章〉  
 \*  
 \* @return  
 \* @Param: [articleTypeId]  
 \* @Return: java.util.List<com.weChat.entity.Article>  
 \* @Author: https://home.cnblogs.com/u/90s-ITBoy/  
 \* @Date: 2020/8/4 11:17  
 \*/  
 @Override  
 public List<Map<String, Object>> queryAllArticleByType(String articleTypeId) {  
 //Object  
 List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();  
  
 // 查询选择文章类型下的所有处于审核通过的文章  
 Example example = new Example(Article.class);  
 Example.Criteria criteria = example.createCriteria();  
 criteria.andEqualTo("typeUuid", articleTypeId);  
 criteria.andEqualTo("status", "1");  
 List<Article> listR = articleMapper.selectByExample(example);  
  
 // 循环遍历  
 if (listR.size() > 0) {  
 for (Article at : listR) {  
 // 获取此篇文章的作者  
 ArticlePublisher articlePublisher = articlePublisherService.queryPublisherByPId(at.getPublisherUuid());  
 // 判断作者现在状态：作者信息不为空且作者位处于被封号状态  
 final String status = articlePublisher.getStatus();  
 Map<String, Object> map = new HashMap<String, Object>();  
 if ("1".equalsIgnoreCase(status)) {  
 map.put("article", at);  
 map.put("publisher", articlePublisher);  
 list.add(map);  
 }  
 }  
 }  
 return list;  
 }  
}