第4天 微信小程序环境搭建与设置

今日目标

- 了解面面小程序端功能与设计
- 初始化微信小程序工程
- 构建微信小程序API工程
- 完成微信小程序登录与注册
- 完成小程序城市与题目方向设置

1. 面面小程序功能与设计

1.1 功能结构图

通过结构图了解到，小程序端核心功能在黑马刷题，首页主要提供的其他功能的入口，当前我们主要实现个人中心及黑马刷题的业务。

1.2 功能列表

序号	模块	子模块	描述
1	用户登录	用户登录	当前系统必须授权登录方可访问
2	设置城市及学科方向	设置城市及学科方向	后续看到的题目数据全部根据当前用户所选的城市及学科方向来提取数据
3	题库分类列表	题库分类列表	分类有三种方式（按技术、按企业、按方向） 按技术实际是按学科目录，后台接口根据当前学科的学科目录来提前学科目录列表 按企业，后台接口根据当前城市提前企业所属城市列表 按方向，后台接口根据所选城市和学科选取行业方向列表 列表中包含所有分类数据及用户记录数据（已完成题目记录）
4	题库分类题目列表	题库分类题目列表	根据所选分类，提前对应的题目列表，包含题目详情信息
5	题目操作	收藏	针对某一题目，用户可以收藏这个题目
	答案提交	答题	答题是在客户端完成 单选题目，只要选中某一选项，自动判断对错 多项选择，需要选择选项后，单独提交答案，完成判断对错 简单题，需要用户根据自己对题目的分析判断，通过查看解析后，完成理想与不理想操作提交
		提交答案	无论单选、多选还是简单，最终需要把当前题目信息提交到后端，后端保存用户做题记录。
6	个人中心	个人中心	获取用户信息数据，展示在个人中心
		继续答题	跳转到最后一次完成答题的位置，继续答题

以上是我们后续章节要实现的功能。

1.3 产品原型

参考资料-产品原型-小程序原型，格式.html


6:00 PM
100%
选择题库方向
北京市
北京
上海
广州
天津
深圳
西安
题库方向
JAVA
Python
大数据
PHP
设计
前端
测试
运维
产品
确定
设置城市及题库方向
6:00 PM
100%
刷题
TAG
企业
方向
HTML/CSS
3212题 | 已做11题
Mysql
3212题 | 已做11题
Redis
3212题 | 已做11题
Sprint
3212题 | 已做11题
Vue.js
3212题 | 已做11题
HTML/CSS
3212题 | 已做11题
Mysql
3212题 | 已做11题
Redis
3212题 | 已做11题
Sprint
3212题
面试题库
个人中心
选择刷题类型页面


6:00 PM
100%
题库
HTML/CSS
现有字典 d={'a':24，'g':52，'l':12，'k':33}请按字 典中的 value 值进行排序？
名企
简单
查看解析
收藏
理想
不理想
1 / 2341
题目刷题页面
WeChat
1:21 AM
100%
个人中心
JACK-LI
一起努力奋斗吧~
26
累计答题数
继续做题
拼了，Fighting~
错题本
我的练习
收藏题库
切换方向
面试题库
模拟面试
我的主页
个人中心页面

1.4 UI设计

参考资料-UI设计-小程序UI设计，格式PNG

1.5 接口文档

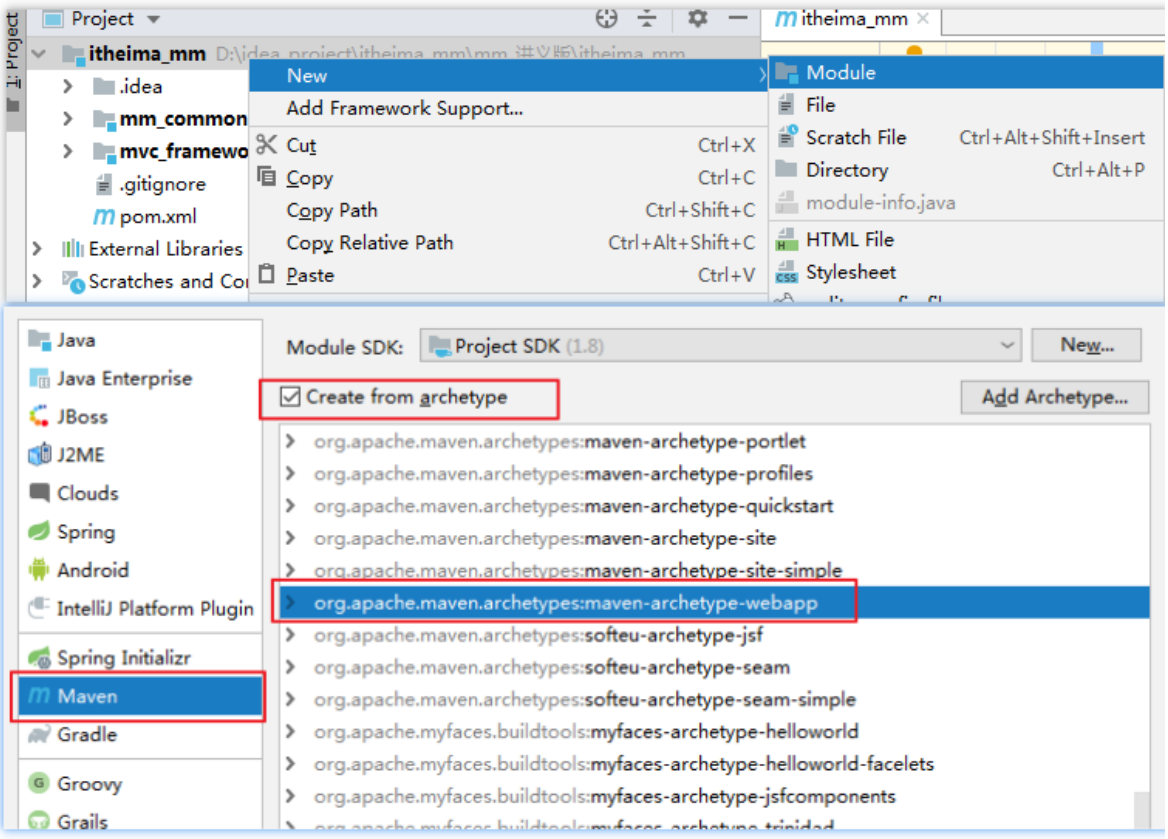
参考资料-接口文档-小程序_api.html

2. 构建微信小程序API模块

在目前工程基础上，创建管理后台子模块，按如下步骤操作：

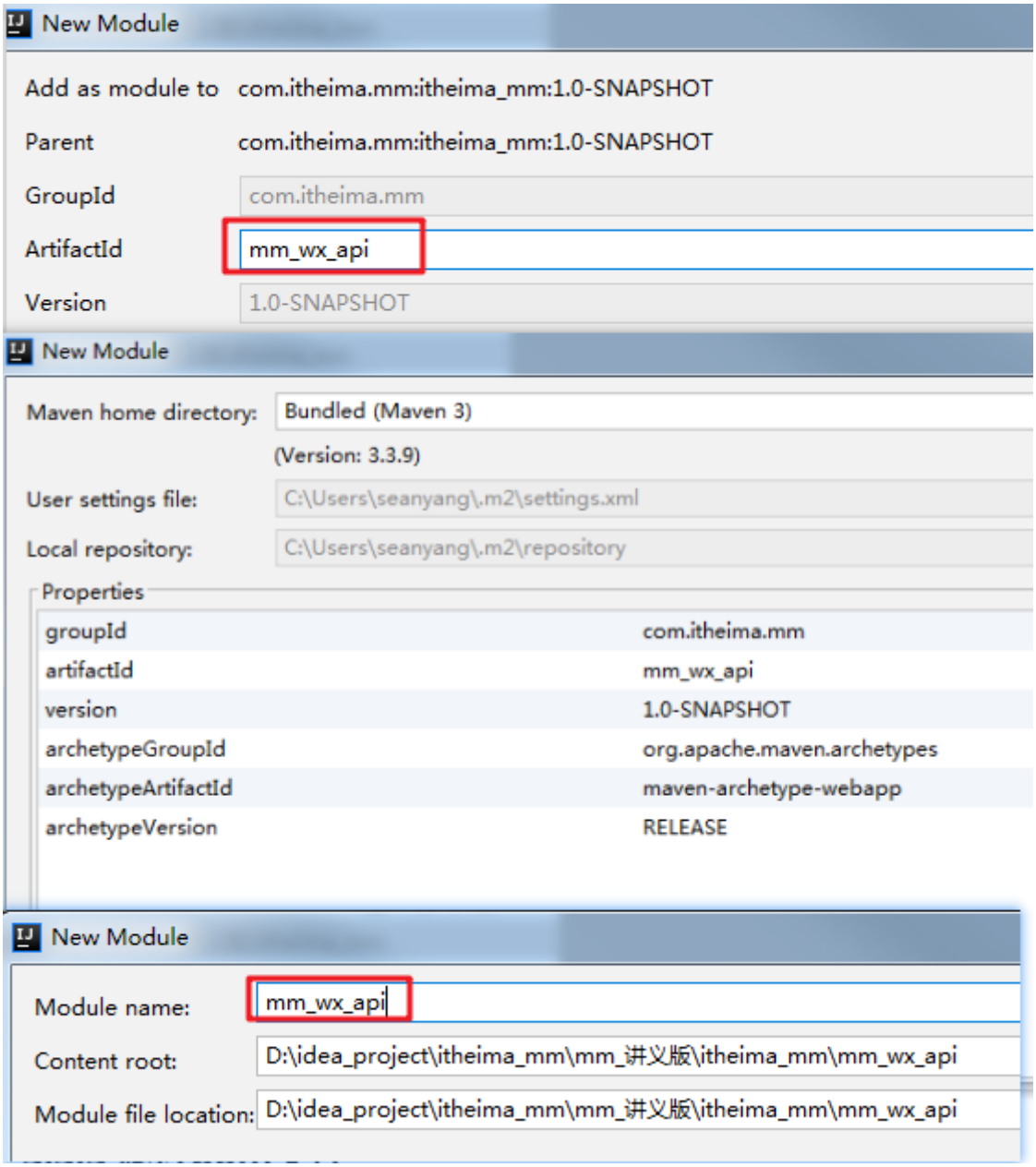
2.1 新建模块

右键项目属性，新建模块，选择maven,基于骨架，选择webapp骨架，如图操作：



2.2 设置模块信息

artifactId、选择maven（默认即可）、设置module名称（默认即可），如图所示：



2.3 修订pom文件

删除pluginManagement节点，加入tomcat7插件，修订配置信息，加入所需依赖，mm_common已经导入的依赖（scope为provided，需要再次导入），本模块无需再次导入，只需把必须加入的导入即可。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<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">
    <parent>
        <artifactId>itheima_mm</artifactId>
        <groupId>com.itheima.mm</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mm_wx_api</artifactId>
    <packaging>war</packaging>
```

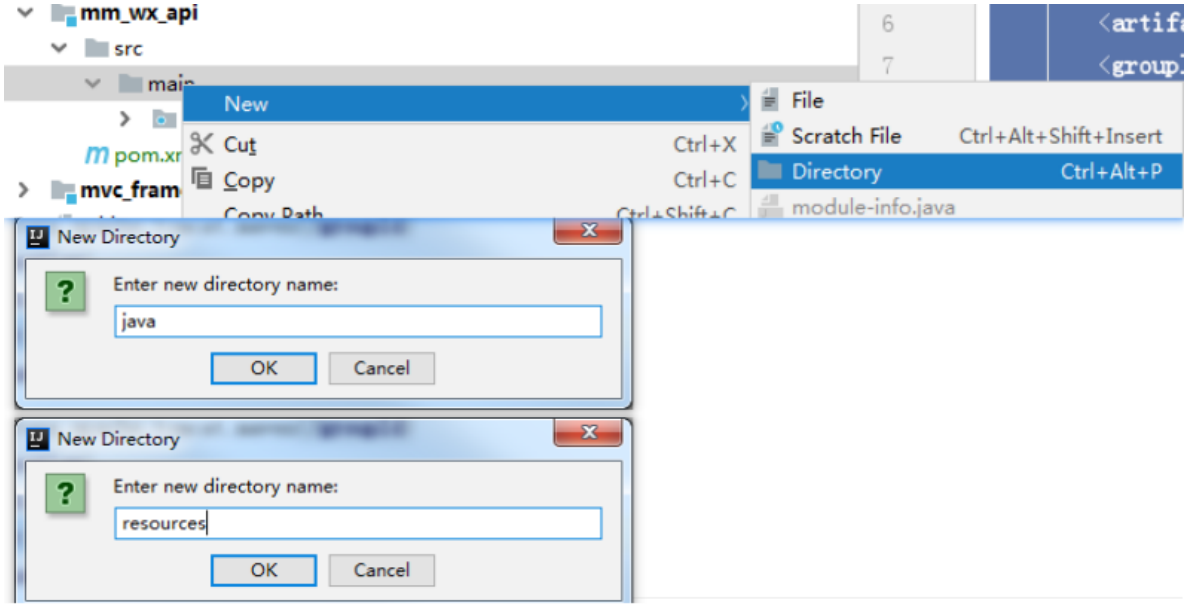
```
    <name>mm_wx_api</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.itheima.mm</groupId>
            <artifactId>mm_common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcprov-jdk16</artifactId>
        </dependency>
    </dependencies>
    <build>
    <finalName>mm_wx_api</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.xml</include>
                <include>**/*.properties</include>
            </includes>
            <filtering>false</filtering>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.xml</include>
                <include>**/*.properties</include>
            </includes>
            <filtering>false</filtering>
        </resource>
```

```
    </resources>
    <plugins>
        <plugin>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <groupId>org.apache.tomcat.maven</groupId>
            <configuration>
                <path>/wx</path>
                <port>7070</port>
            </configuration>
        </plugin>
    </plugins>
    </build>
</project>
```

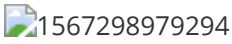
2.4 初始化源代码及资源目录

1. 创建java和resources目录

   右键模块 main 目录，然后创建两个子文件夹分别为java和resources

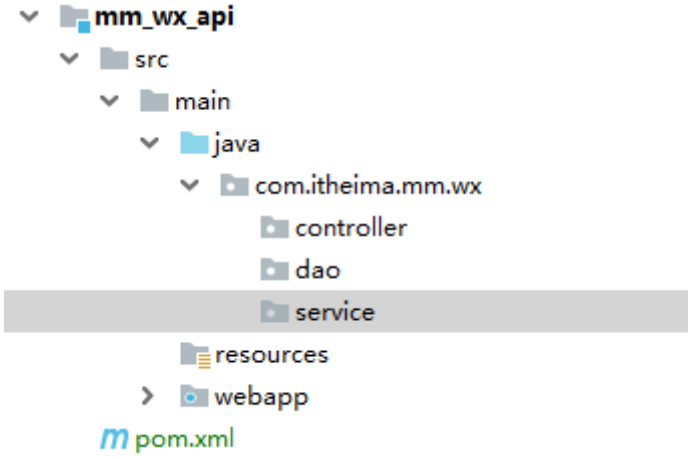


完成后如图所示：



3. 初始化源代码包结构

   创建com.itheima.mm.wx基础包，然后创建controller、dao、service子包，完成后如图所示：



4. 初始化资源目录，加入log4j.properties文件

可参考如下：

```
log4j.rootLogger=DEBUG,stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
#[%-5p] %t %l %d %rms:%m%n
#%d{yyyy-MM-dd HH:mm:ss,SSS\} %-5p [%t] {%c}-%m%n
log4j.appender.stdout.layout.ConversionPattern=[%-5p] %t %l %d %rms:%m%n
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=D:\\idea_project\\itheima_mm_wx_api.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS\} %-5p
[%t] {%c}-%m%n
```

5. 初始化资源目录，加入db.properties文件及mybatis-config.xml配置文件
    - db.properties文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/itheima_mm?characterEncoding=utf-8
jdbc.username=root
jdbc.password=123456
```

    - mybatis-config.xml配置文件

      配置文件内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//com.itheima.mm.database.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 配置属性
      先加载内部属性，再加载外部属性，如果有同名属性会覆盖。
   -->
    <properties resource="db.properties"/>
    <!-- 配置pojo别名 -->
    <typeAliases>
        <!-- 扫描包的形式创建别名，别名就是类名，不区分大小写 -->
        <package name="com.itheima.mm.pojo"/>
    </typeAliases>
    <!--environments配置-->
    <environments default="development">
        <environment id="development">
            <!-- 使用jdbc事务管理-->
            <transactionManager type="JDBC"/>
            <!-- 数据库连接池-->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
                <property name="poolMaximumIdleConnections" value="0"/>
                <property name="poolMaximumActiveConnections" value="1000"/>
                <property name="poolPingQuery" value="SELECT 1 FROM DUAL" />
                <property name="poolPingEnabled" value="true" />
            </dataSource>
```

```
        </environment>
    </environments>
    <mappers>
        <!-- mapper文件和接口在同一包下，可以批量注册 -->
        <!-- 使用扫描包的形式加载dao和mapper文件 -->
        <package name="com.itheima.mm.wx.dao"/>
    </mappers>
</configuration>
```

2.5 配置自定义MVC框架

1. 配置包扫描

   在resources目录下，创建一个.xml文件，名称自定义（这里起名为hm-mvc.xml），内容非常简单，仅需要配置一个扫描包的全名称即可。如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <component-scan package="com.itheima.mm.wx"></component-scan>
</beans>
```

2. 配置web.xml

   配置字符编码过滤器、上下文监听器及初始化参数、MVC总控制器，这些都是由自定义框架提供，直接配置即可，如下所示：

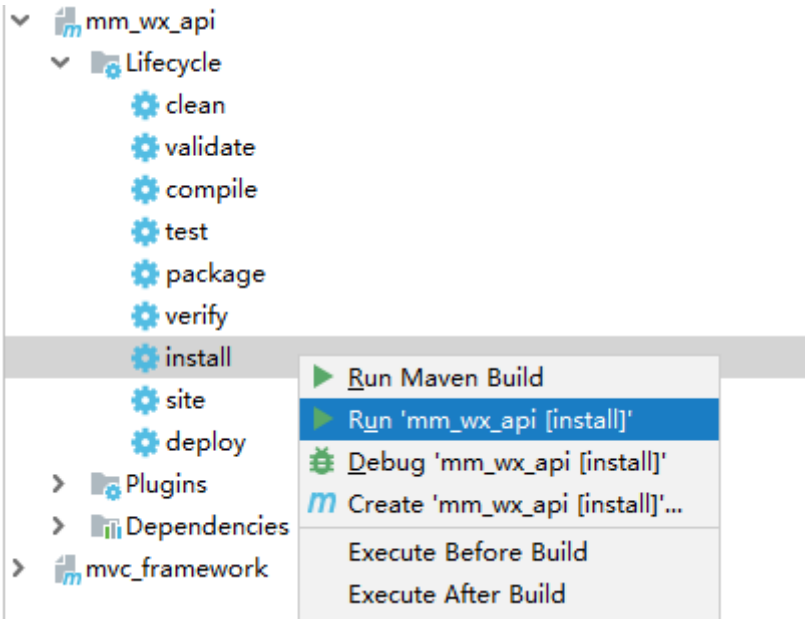
```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <!--上下文参数，默认从resources目录读取-->
  <context-param>
    <param-name>HmContextConfigLocation</param-name>
    <param-value>hm-mvc.xml</param-value>
  </context-param>
  <!--编码过滤器-->
  <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>com.itheima.framework.mvc.CharacterEncodingFilter</filter-
class>
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!--上下文监听器-->
  <listener>
    <listener-
class>com.itheima.framework.mvc.HmContextLoaderListener</listener-class>
  </listener>

  <!-- mvc控制器 -->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
```

```
    <servlet-class>com.itheima.framework.mvc.HmDispatherServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

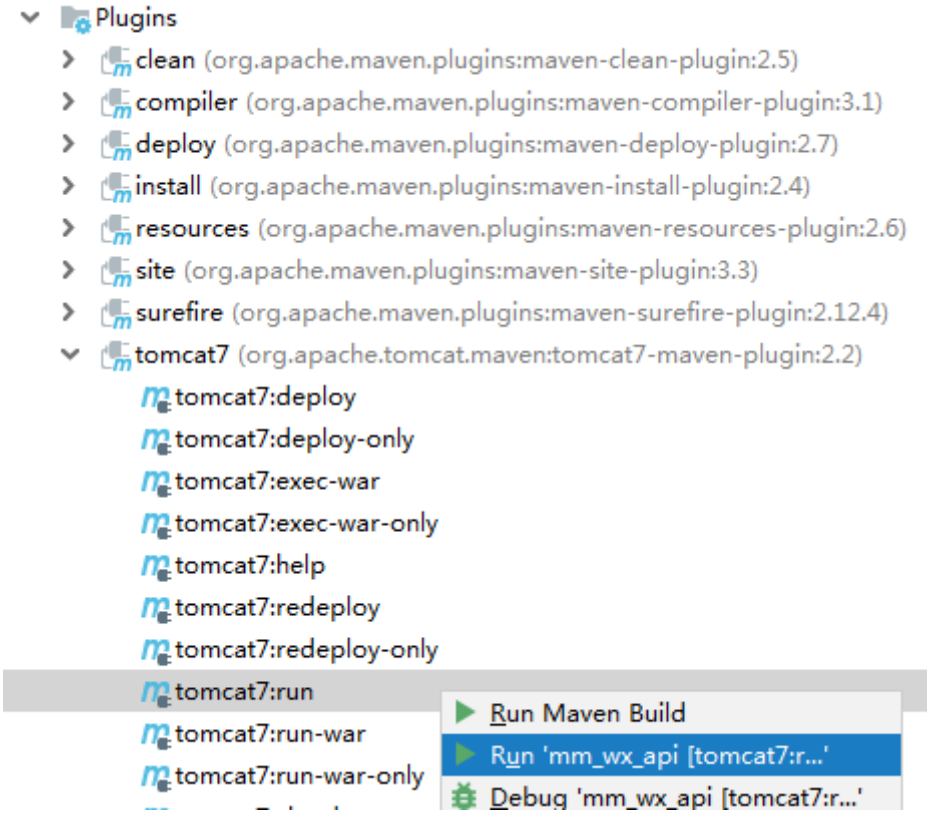
2.6 编译与配置启动设置

1. 编译并启动API模块按如图标注操作即可：

   编译并安装：

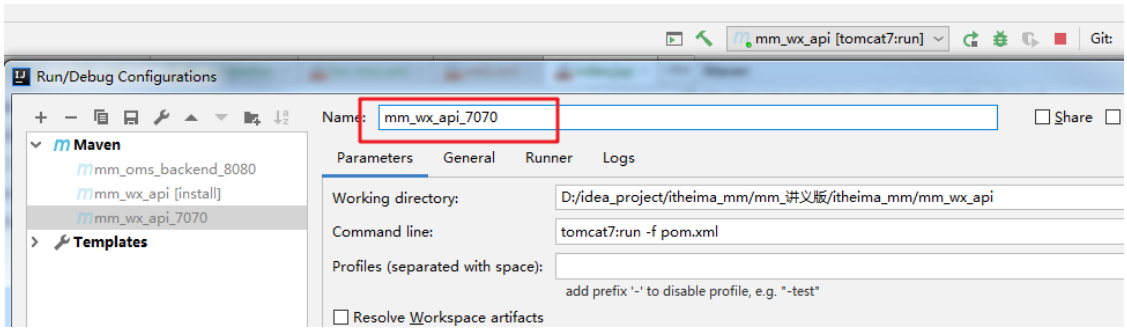


   启动Tomcat7插件：



2. 配置启动设置

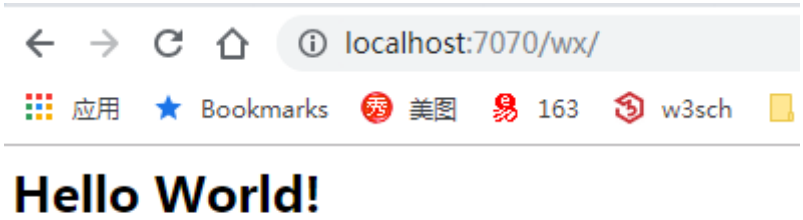
   配置是为了通过工具栏查看方便是哪个服务在启动。



3. 测试访问

   在地址栏输入http://localhost:7070/wx，访问正常且后台不报任何错误，即为当前工程环境搭建及子模块构建成功。

   默认登录页面如下：



3.城市定位

3.1 业务分析

面面小程序端要求第一次启动，先定位当前的位置，然后根据微信提供的地理位置信息，从服务端获取其所在的城市名称及城市ID及城市列表。如果城市不是当前用户需要的，可以进入城市列表，寻找意向的城市名称。

3.2 实现思路

定位需要授权才可以获取，而且是第一次的时候进行授权，在模拟测试时，需要通过小程序的客户端全部清除缓存才可以。微信端获取的是定位的源数据，即地址位置经纬度数据，需要通过服务器或第三方平台，解析出定位位置执行的城市或地址字符串。这个过程称为地理位置逆解析。

当前的思路是，在微信端获取经纬度数据，然后把其传递给服务端，服务端通过第三方平台（百度地图、腾讯地图或高德地图）API，解析地址位置信息，从而获取城市名称，再从我们的数据库获取到城市ID，最终返回给客户端。

根据提供的API文档，小程序客户端会传递经纬度数据到服务端，服务端根据地址解析工具解析出城市，然后根据城市获取相关ID及城市列表返回给客户端。

3.3 解析地址信息

3.3.1 注册地图平台用户

使用高德地图API，先进入官方https://lbs.amap.com/注册一个用户，然后创建一个应用，如图

最终获得一个应用的Key,如图：

Key名称	Key
itheima_mm	8f21643950153e066e4bfefc3d244e19

3.3.2 地址信息解析API

参考高德地图API https://lbs.amap.com/api/webservice/guide/api/georegeo 逆地址解析来完成一个地理位置信息的解析工具。API如图所示：

逆地理编码

- 逆地理编码API服务地址

URL	https://restapi.amap.com/v3/geocode/regeo?parameters
请求方式	GET

*parameters*代表的参数包括必填参数和可选参数。所有参数均使用和号字符(&)进行分隔。下面的列表枚举了这些参数及其使用规则。

- 请求参数

参数名	含义	规则说明	是否必须	缺省值
key	高德Key	用户在高德地图官网申请Web服务API类型Key	必填	无
location	经纬度坐标	传入内容规则：经度在前，纬度在后，经纬度间以“,”分割，经纬度小数点后不要超过 6 位。如果需要解析多个经纬度的话，请用"|"进行间隔，并且将 batch 参数设置为 true，最多支持传入 20 对坐标点。每对点坐标之间用"|"分割。	必填	无

此为GET请求，仅需要Key和location即可。

3.3.3 地址信息解析工具

在utils包下，创建LocationUtils类，使用HttpUtil来完成Http请求的发送与处理。

- HttpUtil类

  利用了OkHttpAPI，代码很简单，如下：

```
/**
 * http请求工具类
 */
public class HttpUtil {
    /**
     * 发起get请求
     *
     * @param url
     * @return
     */
    public static String httpGet(String url) {
        String result = null;
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder().url(url).build();
        try {
            Response response = client.newCall(request).execute();
            result = response.body().string();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result;
    }
}
```

- LocationUtils类

  数据库数据字典城市信息中，城市名称没有包含“市”，故需要把从地图API获取的“市”去掉。

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/21
```

```
 * @description : 地理信息工具类
 * @version: 1.0
 */
public class LocationUtil {
   /**
    * 通过地址位置信息，解析城市信息
    * @param location 地理信息，格式 经度,纬度
    *  114.05,22.55
    * @return
    */
   public static String parseLocation(String location){
      // https://lbs.amap.com/api/webservice/guide/api/georegeo 逆地址解析
      // amap_api 注册高德地图开发者，创建应用，获取apikey
      String amap_api_key = "8f21643950153e066e4bfefc3d244e19";
      String url = "https://restapi.amap.com/v3/geocode/regeo?
key="+amap_api_key+"&"+"location="+location;
      String jsonData =  HttpUtil.httpGet(url);
      JSONObject jsonLocation = JSON.parseObject(jsonData);
      String city = "";
      if("1".equals(jsonLocation.getString("status"))){
         JSONObject addressComponent
=jsonLocation.getJSONObject("regeocode").getJSONObject("addressComponent");
         Object obj = null;
         // 如果是非直辖市，city返回数据
         if((obj = addressComponent.get("city")) instanceof String){
            city=  (String)obj;
         }else if ((obj = addressComponent.get("province")) instanceof String){
            // 如果是直辖市，通过province返回数据
            city= (String)obj;
         }
         city = city.replace("市","");
      }
      return city;
   }
}
```

3.4 构建城市API接口

3.4.1 新增DictDao接口及映射文件

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
 * @description :
 * @version: 1.0
 */
public interface DictDao {
   /**
    * 根据标志获取城市列表
    * 1 推荐列表
    * 0 全部列表
    * @param tag
    * @return
    */
   List<Dict> selectCityListByTag(@Param("tag") Integer tag);

   /**
```

```
    * 根据城市名称，获取城市信息
    * @param cityName
    * @return
    */
   Dict selectByCityName(String cityName);
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.wx.dao.DictDao">
    <select id="selectCityListByTag" resultType="com.itheima.mm.pojo.Dict">
        SELECT id,data_value as title
        FROM t_dict
        <where>
            data_type = 1
            <if test=" tag == 1 ">
               and  data_tag = #{tag}
            </if>
        </where>
    </select>
    <select id="selectByCityName" resultType="com.itheima.mm.pojo.Dict">
        select id,data_value as title
        FROM t_dict
        WHERE data_value LIKE  "%"#{cityName}"%" and data_type = 1
    </select>
</mapper>
```

3.4.2 新增CommonService接口及实现类

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
 * @description : 公共业务接口
 * @version: 1.0
 */
public interface CommonService {
   /**
    * 获取城市列表
    * @param tag
    * @return
    */
   public List<Dict> getCityList(Integer tag);

   /**
    * 根据地理信息，获取某一具体城市
    * @param cityName
    * @return
    */
   public Dict getCityInfoByName(String cityName);
}
```

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
```

```
 * @description : 公共业务接口
 * @version: 1.0
 */
@HmComponent("commonService")
@Slf4j
public class CommonServiceImpl implements CommonService {
    public List<Dict> getCityList(Integer tag){
        log.info("getCityList,tag:{}",tag);
        SqlSession sqlSession = SqlSessionUtils.openSession();
        DictDao dictDao = sqlSession.getMapper(DictDao.class);
        List<Dict> dictList = dictDao.selectCityListByTag(tag);
        sqlSession.close();
        return dictList;
    }

    @Override
    public Dict getCityInfoByName(String cityName) {
        log.info("getCityInfoByName,cityName:{}",cityName);
        SqlSession sqlSession = SqlSessionUtils.openSession();
        DictDao dictDao = sqlSession.getMapper(DictDao.class);
        Dict city = dictDao.selectByCityName(cityName);
        sqlSession.close();
        return city;
    }
}
```

3.4.3 新增CommonController控制器类

在com.itheima.mm.wx.controller包中，创建CommonController控制器类，代码如下：

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
 * @description : 公共控制器
 * @version: 1.0
 */
@HmComponent
@Slf4j
public class CommonController extends BaseController {

   @HmSetter("commonService")
   private CommonService commonService;

   /**
    * 根据参数获取数据
    * fs 0 全部 1 首屏推荐
    * @param request
    * @param response
    * @throws ServletException
    * @throws IOException
    */
   @HmRequestMapping("/common/citys")
    public void getCitys (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        try{
            HashMap<String,Object> mapData =
parseJSON2Object(request,HashMap.class);
```

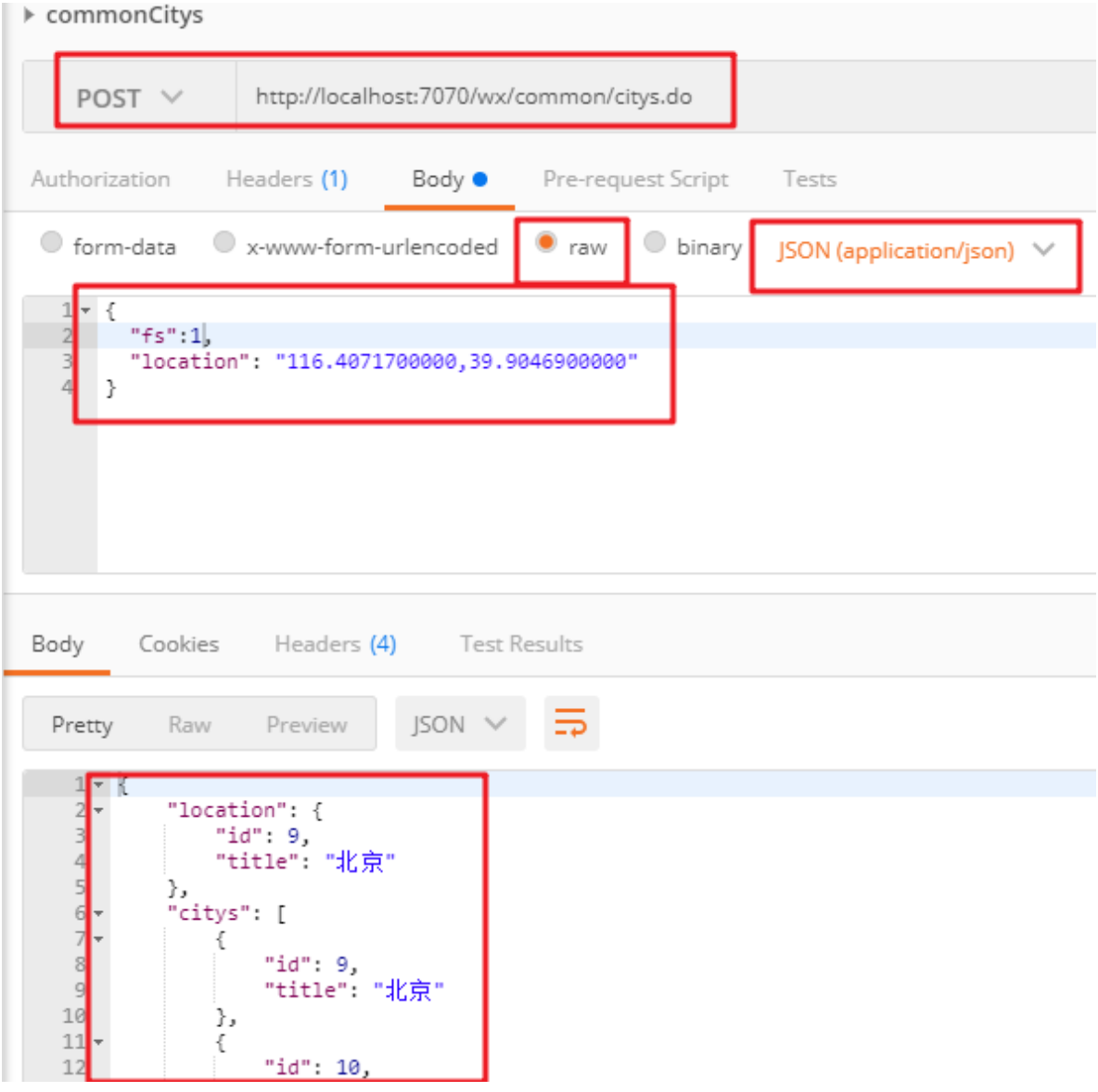
```
            log.info("mapData:{}",mapData);
            String cityName = LocationUtil.parseLocation((String)
mapData.get("location"));
            Dict city  = commonService.getCityInfoByName(cityName);
            List<Dict> cityList =
commonService.getCityList((Integer)mapData.get("fs"));
            Map result = new HashMap();
            result.put("location",city);
            result.put("citys",cityList);
            // 需要特别注意，这里直接返回了result对象到客户端，是为了与前端代码一致。
            printResult(response,result);
        }catch(RuntimeException e){
            log.error("getCitys",e);
           printResult(response,new Result(false,"获取失败"));
        }
    }
}
```

3.4.4 测试

使用如下地理位置信息，测试API接口的调用

"location": "116.4071700000,39.9046900000"

如图：



3.4.5 编写前端代码

以下操作，需要打开微信小程序开发工具，打开小程序客户端代码。

- 定位城市获取首页推荐列表

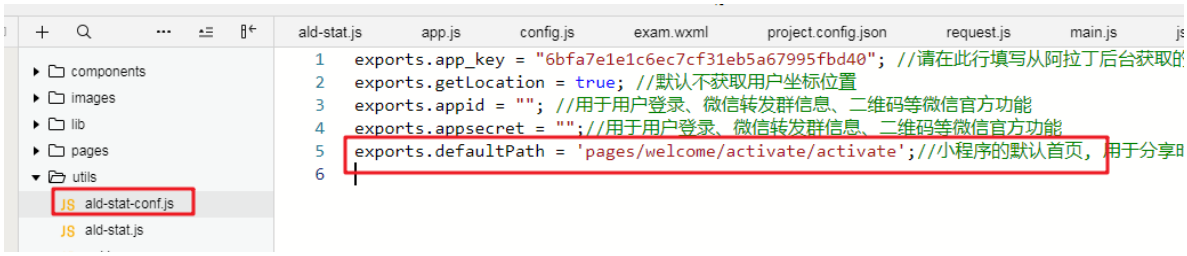
1. 找到工程utils/api.js文件，修改如下代码：

```
// 基础数据
//const baseCitys = data => request('post', `/base/citys/`, data)
const baseCitys = data => request('post', `/common/citys.do`, data,true)
```

注释的部分为原来测试代码，新增代码为使用刚刚完成的城市API来完成的业务。

2. 找到小程序默认首页

在utils下的ald-start=-conf.js中，内容如下：



pages/welcome/activate/activate是默认首页，故需要找到activate.js中的loadCities城市列表接口，确定传参是否正确，如图所示：

```
// ////////////////////////////////////////////////////////////////
// 城市列表接口
loadCities() {
  let _this = this
  let data = {
    fs: 1
  }
  wx.getLocation({
    success: function(res) {
      let lat = res.longitude
      let lng = res.latitude
      data = {
        ...data,
        location: `${lat},${lng}`
      }
    },
    fail: function() {
      $Message({
        content: '当前城市获取失败',
        type: 'error',
        duration: 5
      })
    },
    complete: function() {
      app.api
        .baseCitys(data)
        .then(res => {
          let result = res.data.result;
          console.log(result.citys)
          _this.setData({
            currentCityID: result.location.id,
```

fs 1代表首页推荐城市列表

微信定位API

获取到经纬度，并封装到data

调用api的baseCitys，并传data

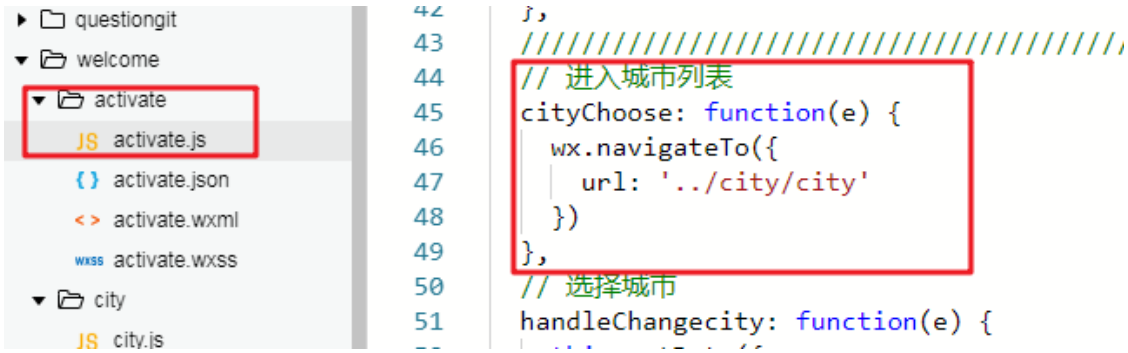
3. 编译并运行

小程序客户端运行测试。看是否能获取到城市信息，如图所示：

- 获取城市全部列表

  刚才是完成了城市定位及首页推荐列表，点击如图箭头获取全部城市列表

  跳转代码，前端已完成，参考如图：



点击右侧箭头获取全部城市列表数据，加载的是pages/welcome/city，找出city.js，确认传参是否正确，如图所示：

编译运行，如图所示：

4. 获取学科列表

刚才已完成了城市定位及城市列表的获取，

在这个页面还需要获取学科列表，需要用户设置城市及学科，后续业务都是根据这两个大条件进行的数据筛选。

4.1 新增CourseDao接口及映射文件

```java
/**
 * @author : seanyang
 * @date : Created in 2019/8/19
 * @description : 学科Dao
 * @version: 1.0
 */
public interface CourseDao {
    /**
     * 获取学科列表
     * @return
     */
    List<Course> getCourseList();
}
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.wx.dao.CourseDao">
    <select id="getCourseList" resultType="com.itheima.mm.pojo.Course">
        SELECT id,name as title,if(icon is null,'',icon) as icon
        FROM t_course
        WHERE is_show = 0
    </select>
</mapper>
```

4.2 修订CommonService接口及实现类

```java
/**
 * 获取学科列表
 * @return
 */
public List<Course> getCourseList();
```

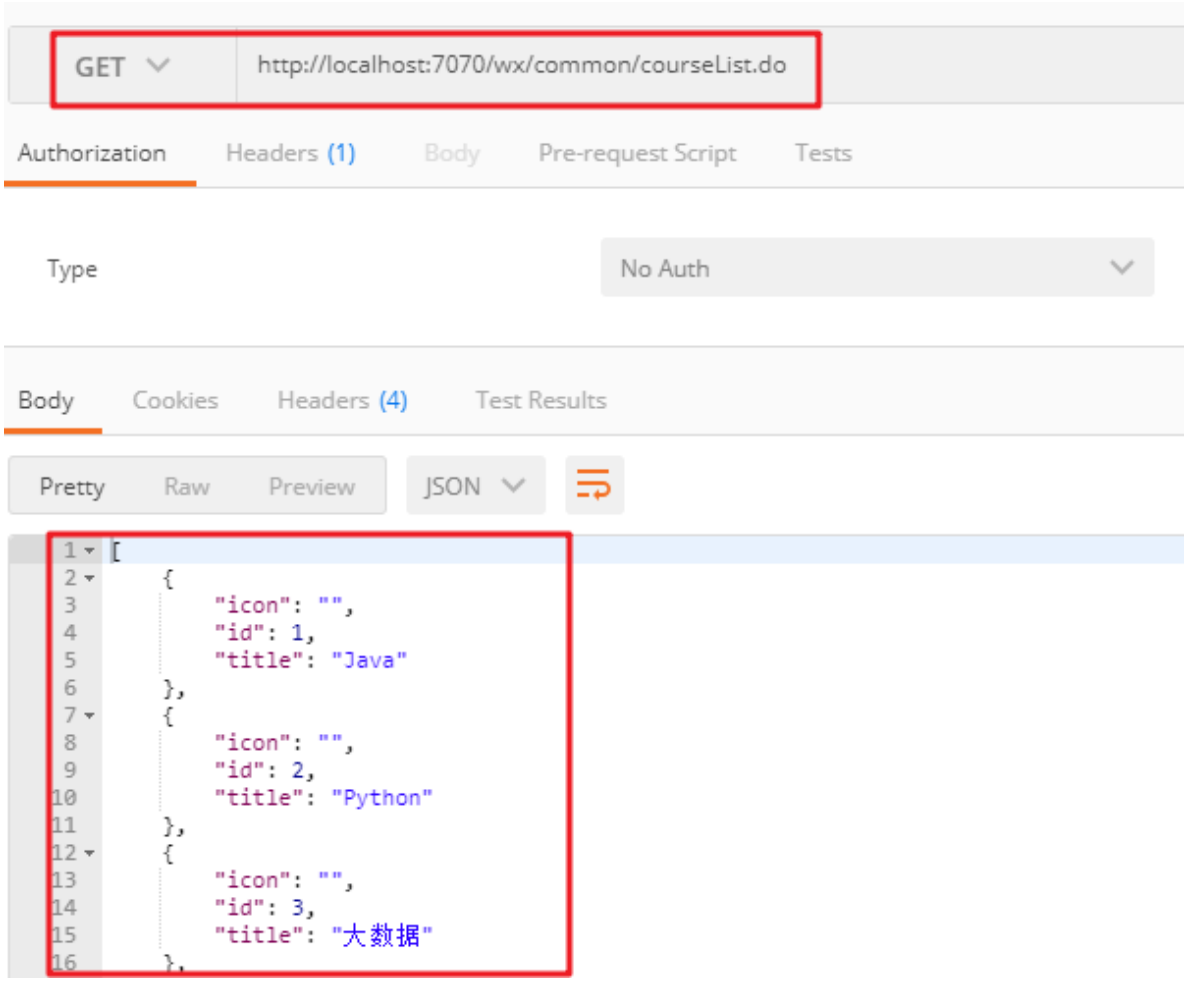
```java
@Override
public List<Course> getCourseList() {
    log.info("getCourseList");
    SqlSession sqlSession = SqlSessionUtils.openSession();
    CourseDao courseDao = sqlSession.getMapper(CourseDao.class);
    List<Course> courseList = courseDao.getCourseList();
    sqlSession.close();
    return courseList;
}
```

4.3 修订CommonController及Service调用

```java
/**
 * 获取学科列表
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@HmRequestMapping("/common/courseList")
```

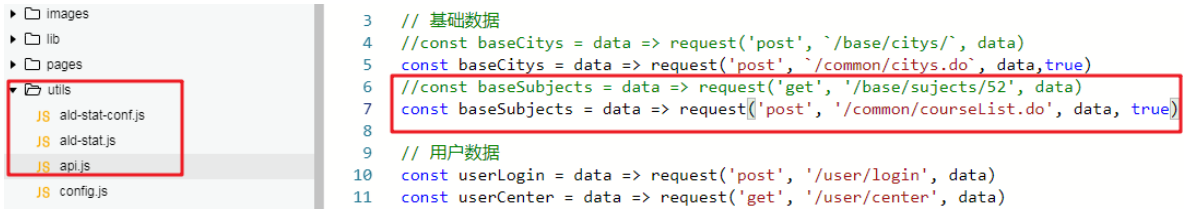
```
public void getCourseList (HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException{
        try{
                List<Course> courseList = commonService.getCourseList();
                printResult(response,courseList);
        }catch(Exception e){
            e.printStackTrace();
            printResult(response,new Result(false,"获取失败"));
        }
}
```

4.4 测试接口



4.5 编写前端代码

找到工程utils/api.js文件，修改如下代码：



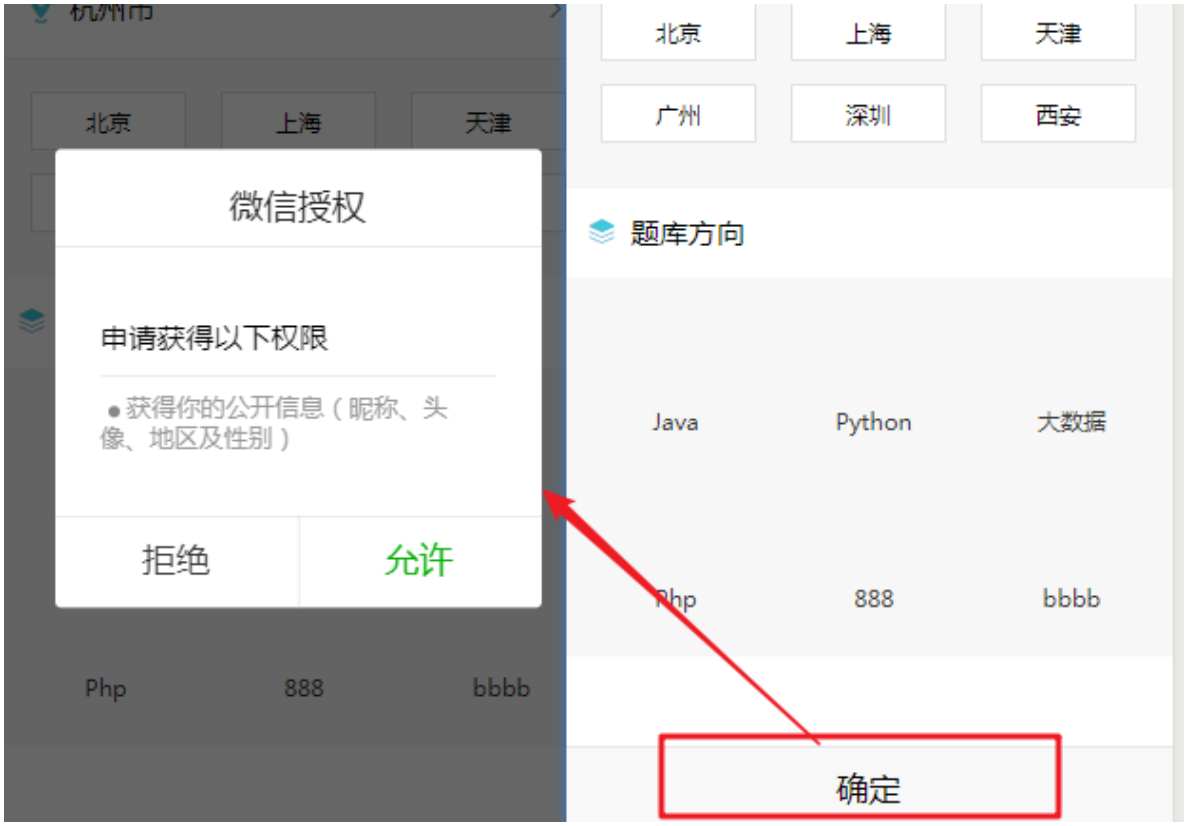
编译并运行小程序客户端，结果如下：

返回的数据因没有图标数据，故仅仅显示的学科名称。

5.登录与注册

5.1 业务分析

定位城市及选择了学科后，需要点击确定按钮保存到服务端。但是这些信息必须与用户关联，如果用户第一次使用该客户端，点击下方确定按钮，会出现如图授权登录提示：

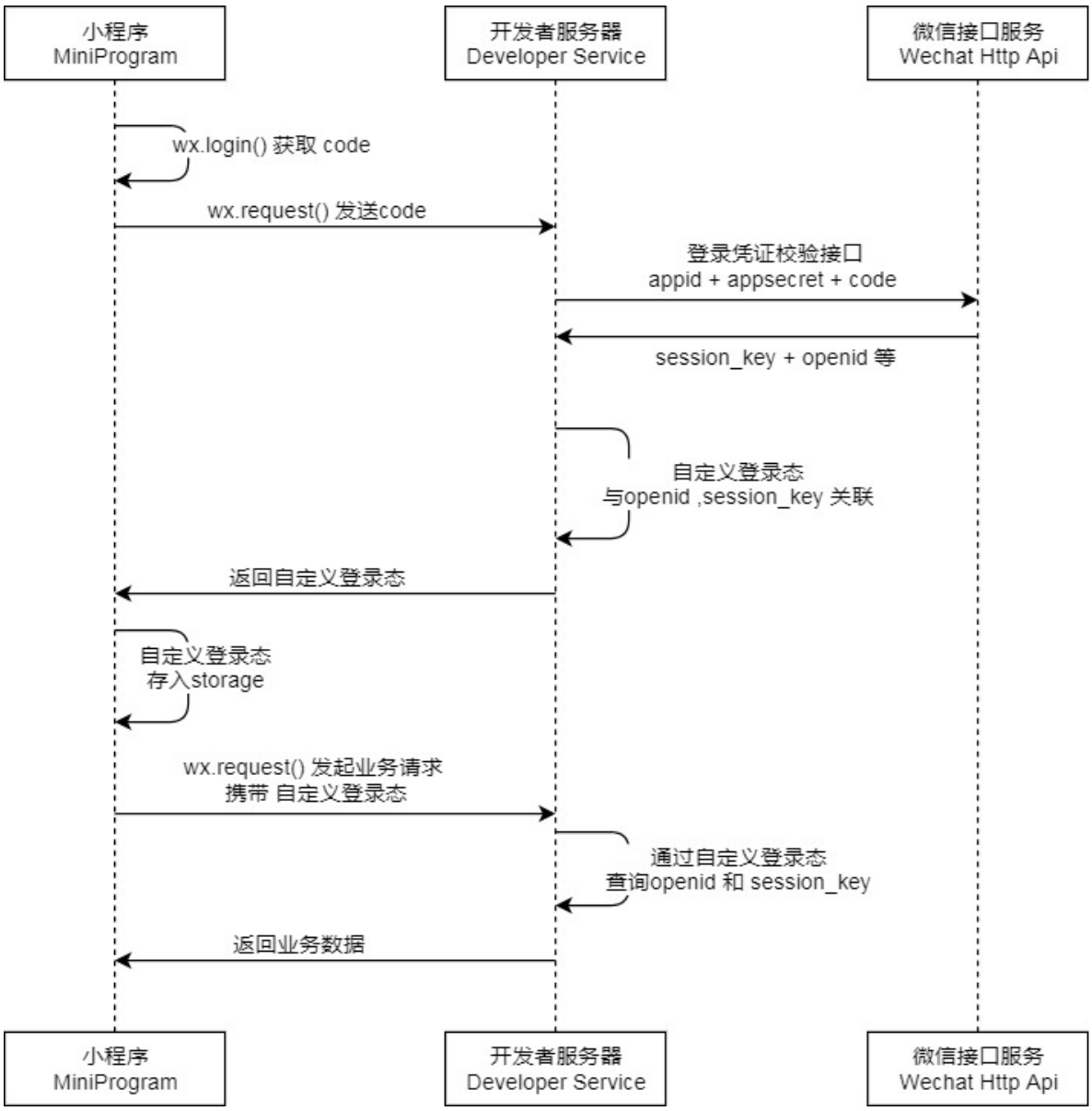


用户允许后为保证当前用户数据的安全性，把用户加密数据传到给服务端，服务端验证且解密该数据后获取到授权用户数据。这个过程可以理解为登录与注册的过程。如果之前服务端没有保存过用户数据，可理解为注册过程，如果保存过只要匹配数据存在即为登录

5.2 基本思路

通过刚才的分析，

用户登录与注册仅是第一次时会有提示，故调试时需要清除全部缓存数据，另外点击允许在微信客户端获取到的用户加密数据，经过服务端验证且解密获取到真正的用户数据，这个验证解密过程是在服务端进行的。这里有一个官方小程序登录流程图：



说明：

1. 调用 wx.login() 获取 **临时登录凭证code** ，并回传到开发者服务器。
2. 调用 auth.code2Session 接口，换取 **用户唯一标识 OpenID** 和 **会话密钥 session_key**。

   这个接口连接地址是https://developers.weixin.qq.com/miniprogram/dev/api-backend/open-api/login/auth.code2Session.html，根据这个地址需要的参数即可获取到需要的数据。

   注意：
3. 会话密钥 `session_key` 是对用户数据进行 加密签名 的密钥。为了应用自身的数据安全，开发者服务器**不应该把会话密钥下发到小程序，也不应该对外提供这个密钥。**
4. 临时登录凭证 code 只能使用一次
5. 开发者服务器可以根据用户标识来生成自定义登录状态，用于后续业务逻辑中前后端交互时识别用户身份。在这里可以使用唯一标识openId作为登录状态标识，正式上线时建议做加密处理。

以上流程图主要说明的是验证过程，解密微信数据是需要的是session_key及元素加密数据通过通用解密算法来完成的。接下来先完成用户的登录，然后再完成用户数据解析与注册。用户登录与否主要根据数据表中是否有openId来判断。

当前用户后续成为会员，目前主要用户t_wx_member表。

5.3 会员登录

需要在WxMemberPOJO类中，增加两个属性，学科ID和城市ID，如下所示：

```
public class WxMember {

    private int id;
    private String nickName;
    private String avatarUrl;
    private String gender;
    private String city;
    private String province;
    private String country;
    private String language;
    private String openId;
    private String unionId;
    private String createTime;
    private Integer courseId;
    private Integer cityId;
    private Integer lastCategoryKind;  // 最后做题分类种类 按技术
    private Integer lastCategoryType;  // 最后做题分类类型  101-
    private Integer lastCategoryId;    // 最后做题分类种类列表项
    private Integer lastQuestionId;    // 最后题目Id

}
```

5.3.1 新增WxMember接口及映射文件

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
 * @description : 微信用户Dao
 * @version: 1.0
 */
public interface WxMemberDao {
   /**
    * 根据openId，获取用户信息
    * @param openId
    * @return
    */
   WxMember selectByOpenId(String openId);

}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.wx.dao.WxMemberDao">
    <resultMap id="baseResultMap" type="WxMember">
        <id property="id" column="id"/>
        <result column="nick_name" property="nickName"/>
```

```
        <result column="avatar_url" property="avatarUrl"/>
        <result column="open_id" property="openId"/>
        <result column="union_id" property="unionId"/>
        <result column="city_id" property="cityId"/>
        <result column="course_id" property="courseId"/>
        <result column="create_time" property="createTime"/>
        <result column="last_category_kind" property="lastCategoryKind"/>
        <result column="last_category_type" property="lastCategoryType"/>
        <result column="last_question_id" property="lastQuestionId"/>
    </resultMap>
    <select id="selectByOpenId" resultMap="baseResultMap">
        SELECT *
        FROM t_wx_member
        WHERE open_id = #{openId}
    </select>
</mapper>
```

5.3.2 新增WxMemberService接口及实现类

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
 * @description : 会员业务接口
 * @version: 1.0
 */
public interface WxMemberService {
   /**
    * 根据openId,获取会员信息
    * @param openId
    * @return
    */
   WxMember findByOpenId(String openId);

}
```

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/17
 * @description : 会员业务实现类
 * @version: 1.0
 */
@HmComponent("wxMemberService")
public class WxMemberServiceImpl implements WxMemberService {

    @Override
    public WxMember findByOpenId(String openId) {
        SqlSession sqlSession = SqlSessionUtils.openSession();
        WxMemberDao wxMemberDao = sqlSession.getMapper(WxMemberDao.class);
        WxMember wxMember = wxMemberDao.selectByOpenId(openId);
        sqlSession.close();
        return wxMember;
    }
}
```

5.3.3 新增WxUtil类

```
package com.itheima.mm.wx.utils;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;

/**
 * 微信工具类
 */
public class WxUtil {
    // 从小程序管理后台获取AppId及secret
    private static final String appid = "wx26d7b4c2dd96dacf";
    private static final String secret = "3cebdc40c5312161fd18b9d4bc897cfa";
    public static void main(String[] args) throws Exception {
        JSONObject object = get("061YvyXS1Nyc641OBRYS1lmnXS1YvyXh");
        System.out.println(object);
    }
    //可获取openid及session_key,其实这里openid不需要获取，encryptedData解密后包含
openid
    public static JSONObject get(String js_code) throws RuntimeException {
        //官方接口，需要自己提供appid，secret和js_code
        String requestUrl = "https://api.weixin.qq.com/sns/jscode2session?
appid=" + appid + "&secret=" + secret + "&js_code=" + js_code +
"&grant_type=authorization_code";
        //HttpRequestor是一个网络请求工具类
        String result = HttpUtil.httpGet(requestUrl);
        return JSON.parseObject(result);
    }
}
```

5.3.3 新增WxMemberController及Service调用

注册稍后完成，目前暂时先使用数据库已存在的用户信息，测试登录过程。

```
/**
 * @author ：seanyang
 * @date ：Created in 2019/8/17
 * @description ：会员控制器
 * @version: 1.0
 */
@HmComponent
@Slf4j
public class WxMemberController extends BaseController {

   @HmSetter("wxMemberService")
   private WxMemberService wxMemberService;

   /**
    * 用户授权登录，未查询到添加用户信息到数据库
    * @param req
    * @param response
    * @throws ServletException
    * @throws IOException
    */
   @HmRequestMapping("/member/login")
    public void login (HttpServletRequest req, HttpServletResponse response)
throws ServletException, IOException {
        try {
```

```
            HashMap<String,String> mapData =
parseJSON2Object(req,HashMap.class);
            //获取参数
            String encryptedData = mapData.get("encryptedData");
            String iv = mapData.get("iv");
            String code = mapData.get("code");
            //获取session_key和openid（微信用户唯一标识）
            JSONObject wx = WxUtil.get(code);
            String sessionKey = wx.getString("session_key");
            String openId = wx.getString("openid");
            //查询微信用户表，存在就直接返回微信用户信息，不存在就新增后再返回微信用户信息
            //Object o = wus.dtl(openId);
            log.debug("openId:{},mapData:{}",openId,mapData);
            // 模拟实现
            //openId = "oiu565SzoTCXctUDOy6Ll-RQOkFg";
            WxMember wxUser = wxMemberService.findByOpenId(openId);
            if (wxUser == null) {
                log.debug("注册会员......");
            }
            //返回微信用户信息
            String wxUid = wxUser.getOpenId();
            Map map = new HashMap();
            map.put("token", wxUid);
            map.put("userInfo", wxUser);
            printResult(response,map);
        } catch (RuntimeException e) {
            //访问失败返回{}
            log.error("login",e);
            printResult(response,new Result(false,"登录失败"));
        }
    }
}
```

5.5 编写前端代码

点击确定按钮，activate.js代码如下：

```
// 点击确定，并保存选择
handleLogin: function(e) {
  let _this = this
  let data = {
    cityID: _this.data.currentCityID,
    subjectID: _this.data.subjectID // FIXME: subjectID
  }

  let userInfo = wx.getStorageSync('userInfo') || null
  // let categoryType = wx.getStorageSync('categoryType') || null
  if (userInfo) {   判断用户是否登录过
    app.api
    })
  } else {
    app.getUserInfo(e.detail, function() {
      app.api
        .questionsConfirm(data)
        .then(res => {
          // wx.setStorageSync('categoryType', data) // 写缓存 //
          // console.log(res)
          wx.redirectTo({
            url:
              '/pages/main/main?cityID=' +
              data.cityID +
              '&subjectID=' +
```

未登录过，调用app.getUserInfo方法

app.js中getUserInfo的定义如下：

```
  // 新版 按钮方式登录
  getUserInfo(detail, cb) {
    if (this.globalData.userInfo) {
      typeof cb == 'function' && cb(this.globalData.userInfo)
    } else {
      let data = {
        code: wx.getStorageSync('code') || null,
        encryptedData: detail.encryptedData,
        iv: detail.iv
      }                把用户加密数据进行封装
      console.log('data => ', data)
      this.api.userLogin(data).then(res => {
        console.log('res => ', res)    调用API把数据传给服务端
        wx.setStorageSync('token', res.data.token)
        wx.setStorageSync('userInfo', res.data.userInfo)
        typeof cb == 'function' && cb()
      })
    }
  },
```

修订api.js，调用用户登录API

```
// 用户数据
//const userLogin = data => request('post', '/user/login', data)
const userLogin = data => request('post', '/member/login.do', data,true)
```

后台控制台调用如下：

```
mapData:{code=null, encryptedData=ixAHP18YGaBE5+HGYHNRr4hnU+CnErAyzrB1jLA+OXTA+w9ckG/bgKix4RXT
pening JDBC Connection
93ms:Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@43463091]
SELECT * FROM t_wx_member WHERE open_id = ?
oiu565SzoTCXctUD0y6Ll-RQOkFg(String)
```

5.4 解密数据

微信用户加密数据解密是通过sessionkey和iv两个参数，通过微信官方提供的通用算法来完成解密过程。解密算法是通用算法，不需要自己去研究，该方法返回JSON格式，把如下解密方法加入到WxUtil中。

```
public static JSONObject getUserInfo(String encryptedData, String sessionKey,
String iv){
    // 被加密的数据
    byte[] dataByte = Base64.decode(encryptedData);
    // 加密秘钥
    byte[] keyByte = Base64.decode(sessionKey);
    // 偏移量
    byte[] ivByte = Base64.decode(iv);

    try {
        // 如果密钥不足16位，那么就补足.  这个if 中的内容很重要
        int base = 16;
        if (keyByte.length % base != 0) {
            int groups = keyByte.length / base + (keyByte.length % base != 0 ? 1
: 0);
            byte[] temp = new byte[groups * base];
            Arrays.fill(temp, (byte) 0);
            System.arraycopy(keyByte, 0, temp, 0, keyByte.length);
            keyByte = temp;
        }
        // 初始化
        Security.addProvider(new BouncyCastleProvider());
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding","BC");
        SecretKeySpec spec = new SecretKeySpec(keyByte, "AES");
        AlgorithmParameters parameters = AlgorithmParameters.getInstance("AES");
        parameters.init(new IvParameterSpec(ivByte));
        cipher.init(Cipher.DECRYPT_MODE, spec, parameters);// 初始化
        byte[] resultByte = cipher.doFinal(dataByte);
        if (null != resultByte && resultByte.length > 0) {
            String result = new String(resultByte, "UTF-8");
            return JSONObject.parseObject(result);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

5.5 会员注册

所谓用户注册就是根据openId在数据库中未发现该用户，需要先解析数据再把数据注册到数据。

5.5.1 更新WxMemberDao接口及映射文件

```
/**
 * 添加会员
 * @param wxMember
 */
Integer addWxMemeber(WxMember wxMember);
```

```
<insert id="addWxMemeber">
    INSERT  INTO t_wx_member (nick_name, avatar_url, gender, city, province,
country,language,open_id, union_id, create_time)
    VALUES (#{nickName},#{avatarUrl},#{gender},#{city},#{province},#{country},#
{language},#{openId},#{unionId},#{createTime})
    <selectKey keyProperty="id" order="AFTER" resultType="integer">
        SELECT LAST_INSERT_ID()
    </selectKey>
</insert>
```

5.5.2 更新WxMemberService接口及实现类

```
/**
 * 添加会员
 * @param wxMember
 * @return
 */
 void add(WxMember wxMember);
```

```
@Override
public void add(WxMember wxMember) {
    SqlSession sqlSession = SqlSessionUtils.openSession();
    WxMemberDao wxMemberDao = sqlSession.getMapper(WxMemberDao.class);
    Integer result = wxMemberDao.addWxMemeber(wxMember);
    sqlSession.commit();
    sqlSession.close();
    if (result == 0){
        throw new MmDaoException("添加会员失败");
    }
}
```

5.5.3 更新WxMemberController及Service调用

```
// 去掉模拟实现
//openId = "oiu565SzoTCXctUDOy6Ll-RQOkFg";
WxMember wxUser = wxMemberService.findByOpenId(openId);
if (wxUser == null) {
   log.debug("注册会员......");
   //解析微信用户信息
   JSONObject userJson = WxUtil.getUserInfo(encryptedData, sessionKey, iv);
   wxUser = new WxMember();
   wxUser.setNickName(userJson.getString("nickName"));
   wxUser.setAvatarUrl(userJson.getString("avatarUrl"));
   wxUser.setGender(userJson.getString("gender"));
   wxUser.setCity(userJson.getString("city"));
   wxUser.setProvince(userJson.getString("province"));
```

```
    wxUser.setCountry(userJson.getString("country"));
    wxUser.setLanguage(userJson.getString("language"));
    wxUser.setOpenId(userJson.getString("openId"));
    wxUser.setUnionId(userJson.getString("unionId"));
    wxUser.setCreateTime(DateUtils.parseDate2String(new Date()));
    //调用微信用户服务，执行新增操作
    wxMemberService.add(wxUser);
}
```

5.5.4 测试注册

清除缓存数据，编译并运行微信小程序客户端。

6. 设置城市及学科

6.1 业务分析

点击确定按钮，除了授权登录与注册，还需要同时存储当前用户设置的城市与学科信息，在会员信息表中，有两个属性，分别储存的是城市ID与学科Id，后续提取数据时，需要根据这两个大方向获取数据。比如获取主页中的学科目录列表（根据学科Id）、企业列表（根据所选城市）、学科方向列表（根据学科Id ）。

6.2 实现思路

提供专门的接口，设置城市及学科，主要更新的会员信息表中的城市ID及学科ID字段。

6.3 修订WxMemberDao接口及映射文件

```
/**
 * 更新城市和学科方向
 * @param updateData
 */
Integer updateCityAndCourse(Map updateData);
```

```
<update id="updateCityAndCourse">
    UPDATE t_wx_member
    set city_id = #{cityID},course_id=#{subjectID}
    where open_id=#{openId}
</update>
```

6.4 修订WxMemberService接口及实现类

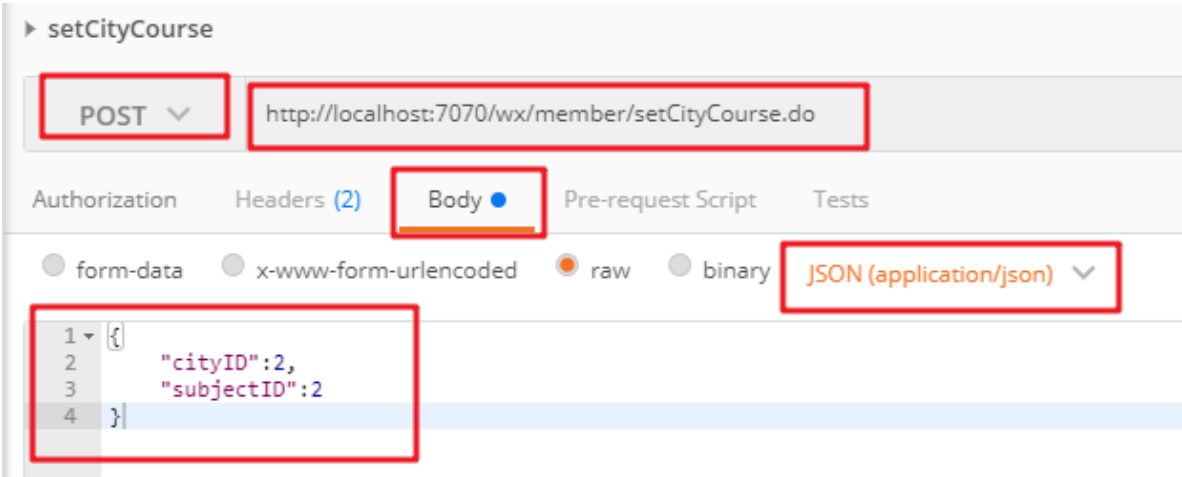
```
/**
 * 更新城市及学科方向
 * @param updateData
 */
 void updateCityCourse(Map updateData);
```

```
@Override
public void updateCityCourse(Map updateData) {
    SqlSession sqlSession = SqlSessionUtils.openSession();
    WxMemberDao wxMemberDao = sqlSession.getMapper(WxMemberDao.class);
    Integer result = wxMemberDao.updateCityAndCourse(updateData);
    sqlSession.commit();
    sqlSession.close();
    if (result == 0){
        throw new MmDaoException("更新数据失败");
    }
}
```

6.5 修订WxMemberController及Service调用

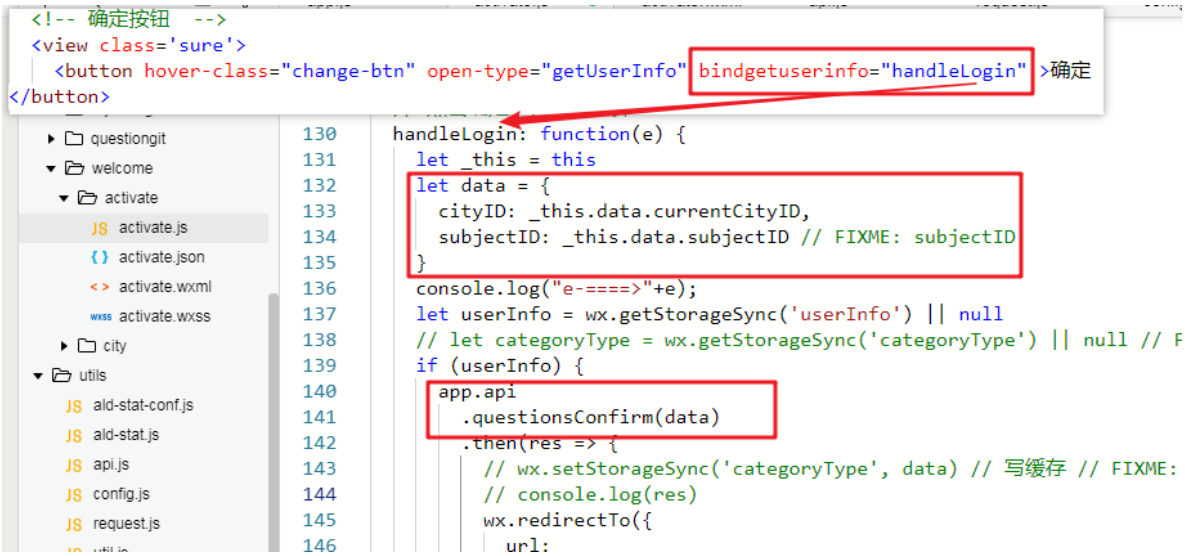
```
/**
 * 更新用户城市及科学方向
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@HmRequestMapping("/member/setCityCourse")
public void updateCityCourse (HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException{
    try{
        HashMap<String,String> mapData =
parseJSON2Object(request,HashMap.class);
        String openId = getHeaderAuthorization(request);
        mapData.put("openId",openId);
        log.debug("updateCityCourse mapData:{}",mapData);
        wxMemberService.updateCityCourse(mapData);
        printResult(response,new Result(true,"更新成功"));
    }catch(RuntimeException e){
        log.error("updateCityCourse",e);
        printResult(response,new Result(false,"更新失败"));
    }
}
```

6.6 测试接口



6.7 编写前端代码

点击确定按钮，视图事件调用过程如下：



修订api.js,代码如下：

```
// const questionsConfirm = data => request('post', `/questions/confirm`, data)
const questionsConfirm = data => request('post', `/member/setCityCourse.do`, data,true)
```

编译并运行，查看数据库的数据是否正确。

7. 优化学科列表

7.1 业务分析

学科列表经常获取，且学科列表不经常发生变化，可以借助redis来优化学科列表的获取速度。

7.2 实现思路

可以在第一次获取学科列表时，判断redis中是否有缓存数据，如果没有缓存数据，需要先从数据库获取学科列表，然后转换为JSON字符串后存入redis。如果redis中有学科缓存数据，需要把json字符串转化为JSON对象然后返回给客户端。

7.3 初始化redis资源

7.3.1 配置文件

在resources目录下，创建jedis.properties文件，内容如下：

```
jedis.host=localhost
jedis.port=6379
jedis.maxTotal=30
jedis.maxIdle=10
```

7.3.2 初始化jedis工具

在CommonController类中，加入静态初始化块，代码如下：

```
static {
    JedisUtils.init(ResourceBundle.getBundle("jedis"));
}
```

7.4 修订CommonController类

主要修订getCourseList方法，加入redis实现。

```
/**
 * 获取学科列表
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@HmRequestMapping("/common/courseList")
public void getCourseList (HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException{
    try{
        if(JedisUtils.isUsed()){
            Jedis jedis = JedisUtils.getResource();
            // 如果redis可用
            String jsonCourseList =
jedis.get(GlobalConst.REDIS_KEY_WX_COURSE_LIST);
            if(jsonCourseList!=null && jsonCourseList.length() >0 ){
                log.debug("redis 获取数据");
                jedis.close();
                printResult(response, JSON.parse(jsonCourseList));
                return;
            }
            List<Course> courseList = commonService.getCourseList();
            log.debug("redis 设置数据");

jedis.set(GlobalConst.REDIS_KEY_WX_COURSE_LIST,JSON.toJSONString(courseList));
            jedis.close();
            printResult(response,courseList);
        }else{
            List<Course> courseList = commonService.getCourseList();
            printResult(response,courseList);
        }
    }catch(RuntimeException e){
        log.error("getCourseList",e);
        printResult(response,new Result(false,"获取失败"));
    }
}
```

7.5 测试效果

启动Redis服务器，然后编译并运行当前项目，然后启动小程序客户端，来验证数据的获取过程。

va:123) 2019-09-04 10:54:04,633 20668ms:Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC
9-09-04 10:54:04,634 20669ms:Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@1ee42ecd]
2019-09-04 10:54:09,564 25599ms:redis 获取数据