开发项目的流程

后端程序员：

1. 产品经理
2. 前端
3. 测试

开发流程：

1. BRD，MRD
2. 产品经理 出 PRD文档 产品需求文档
3. 评审
4. 下发需求，会和后端程序员开会
5. 产品经理 出一个 原型图
6. 需求评估会议，前端，后端，UI，产品等共同参与，估时的工作，领任务的过程
    1. 估时 需要把测试的时间 估算进来
    2. 1h，0.5d，1d，2d，3d
7. 开发内部工作
8. 分配任务，估算时间
9. 项目设计文档 参数的定义，接口的说明，流程的说明，返回的数据说明，技术栈等等
10. 数据库设计阶段，技术leader，资深开发带领
11. 评审 设计文档 数据库设计
12. 开发人员 会搭建整体的项目，上传git，其他开发人员 拉取代码，开始在此框架上进行对应的开发
13. 需要给前端人员 提供一些 接口的定义（接口路径，请求方式，参数，返回结果等）
14. 编码阶段，开例会或者叫早会
15. 自测
16. 提交测试
17. 测试人员介入
18. 改bug
19. 测试人员 可以达到上线标准
20. 上线，部署线上的环境了，打包，数据库导入，自动化运维

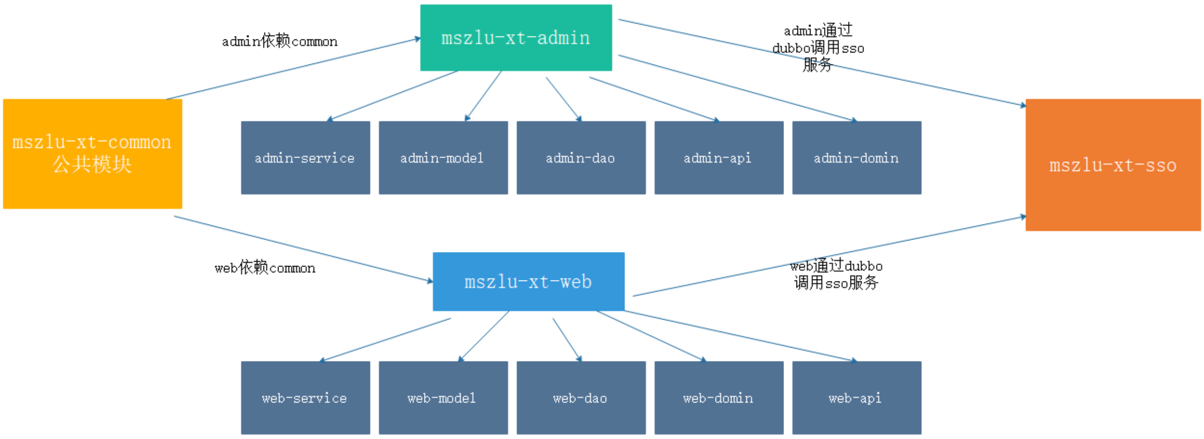
项目背景

> *中小学生能取得好成绩的前提是，进行大量刷题，但是题必须有质量，一些名师和一些已经鸡娃结束的家长，联合起来，将一些有超高价值的题总结起来，其中不同城市对题的要求是不一样的。*
>
> *需求开发一个在线的刷题平台，考虑到后期人数多了之后，可以请名师录制一些教学视频或者开设对应的直播课，所以本系统定义为综合性的在线教育平台。*
>
> *这些题是以课程的形式售卖的，只有买了对应的课程，才能去做题或者学习视频教程，系统的初期是以刷题为主，习题分为填空题，单选，多选，判断，问答题，题目和选项均包含图片。*
>
> *需要有分销商，帮忙卖课程，也就是涉及到分佣，优惠等，这些都需要考虑，同时不只要支持pc端，公众号端也需要支持，因为推广都是微信群推广，需要在微信端直接完成交易，锁定用户。*

第一章 登录注册

1. 项目搭建

1.1 项目结构



1.2 初始化

本章节只涉及登录注册功能，故只搭建mszlu-xt-common以及mszlu-xt-sso

1. 搭建父工程mszlu-xt-parent

   pom文件：

```xml
<?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">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.mszlu</groupId>
    <artifactId>mszlu-xt-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>mszlu-xt-common</module>
        <module>mszlu-xt-sso</module>
        <module>mszlu-xt-web</module>
        <module>mszlu-xt-admin</module>
        <module>mszlu-xt-common-pojo</module>
    </modules>


    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.0</version>
    </parent>

    <packaging>pom</packaging>
    <properties>
        <java.version>1.8</java.version>
        <shardingsphere.version>5.0.0-beta</shardingsphere.version>
        <weixin.version>3.8.0</weixin.version>
        <dubbo.version>2.7.8</dubbo.version>
    </properties>
    <dependencyManagement>
        <dependencies>
            <!--微信支付-->
            <dependency>
                <groupId>com.github.binarywang</groupId>
                <artifactId>weixin-java-mp</artifactId>
                <version>${weixin.version}</version>
            </dependency>
            <dependency>
                <groupId>com.github.binarywang</groupId>
                <artifactId>weixin-java-pay</artifactId>
                <version>${weixin.version}</version>
            </dependency>
            <dependency>
                <groupId>com.github.binarywang</groupId>
                <artifactId>weixin-java-open</artifactId>
                <version>${weixin.version}</version>
            </dependency>
            <!--mybatis-->
            <dependency>
                <groupId>org.mybatis.spring.boot</groupId>
                <artifactId>mybatis-spring-boot-
starter</artifactId>
                <version>1.3.2</version>
            </dependency>
            <!--mybatis plus-->
            <dependency>
                <groupId>com.baomidou</groupId>
                <artifactId>mybatis-plus-boot-starter</artifactId>
                <version>3.4.1</version>
            </dependency>
            <dependency>
                <groupId>com.baomidou</groupId>
                <artifactId>mybatis-plus-annotation</artifactId>
                <version>3.4.1</version>
                <scope>compile</scope>
            </dependency>
            <!--sharding-jdbc 读写分离 分库分表-->
            <dependency>
                <groupId>org.apache.shardingsphere</groupId>
                <artifactId>shardingsphere-jdbc-core-spring-boot-
starter</artifactId>
                <version>${shardingsphere.version}</version>
            </dependency>
            <!--时间处理工具类-->
            <dependency>
                <groupId>joda-time</groupId>
                <artifactId>joda-time</artifactId>
                <version>2.10.10</version>
```

```
        </dependency>
        <!--POI报表-->
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
            <version>3.14</version>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>3.14</version>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.70</version>
        </dependency>

        <!-- Dubbo Spring Boot Starter -->
        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>${dubbo.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo</artifactId>
            <version>${dubbo.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>javax.servlet</groupId>
                    <artifactId>servlet-api</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>log4j</groupId>
```

```
                        <artifactId>log4j</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>
            <!-- Dubbo Registry Nacos -->
            <dependency>
                <groupId>com.alibaba.nacos</groupId>
                <artifactId>nacos-client</artifactId>
                <version>2.0.3</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

</project>
```

2. 搭建mszlu-xt-common工程

   pom文件:

```
<?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>mszlu-xt-parent</artifactId>
        <groupId>com.mszlu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-common</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <!--时间处理工具类-->
```

```xml
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
        </dependency>
        <!--编解码工具类-->
        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
        </dependency>
        <!--wx-java相关，公众号，支付，开放平台等-->
        <dependency>
            <groupId>com.github.binarywang</groupId>
            <artifactId>weixin-java-mp</artifactId>
        </dependency>
        <dependency>
            <groupId>com.github.binarywang</groupId>
            <artifactId>weixin-java-pay</artifactId>
        </dependency>
        <dependency>
            <groupId>com.github.binarywang</groupId>
            <artifactId>weixin-java-open</artifactId>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>
        <dependency>
```

```
            <groupId>com.qiniu</groupId>
            <artifactId>qiniu-java-sdk</artifactId>
            <version>[7.7.0, 7.7.99]</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
        </dependency>
    </dependencies>

</project>
```

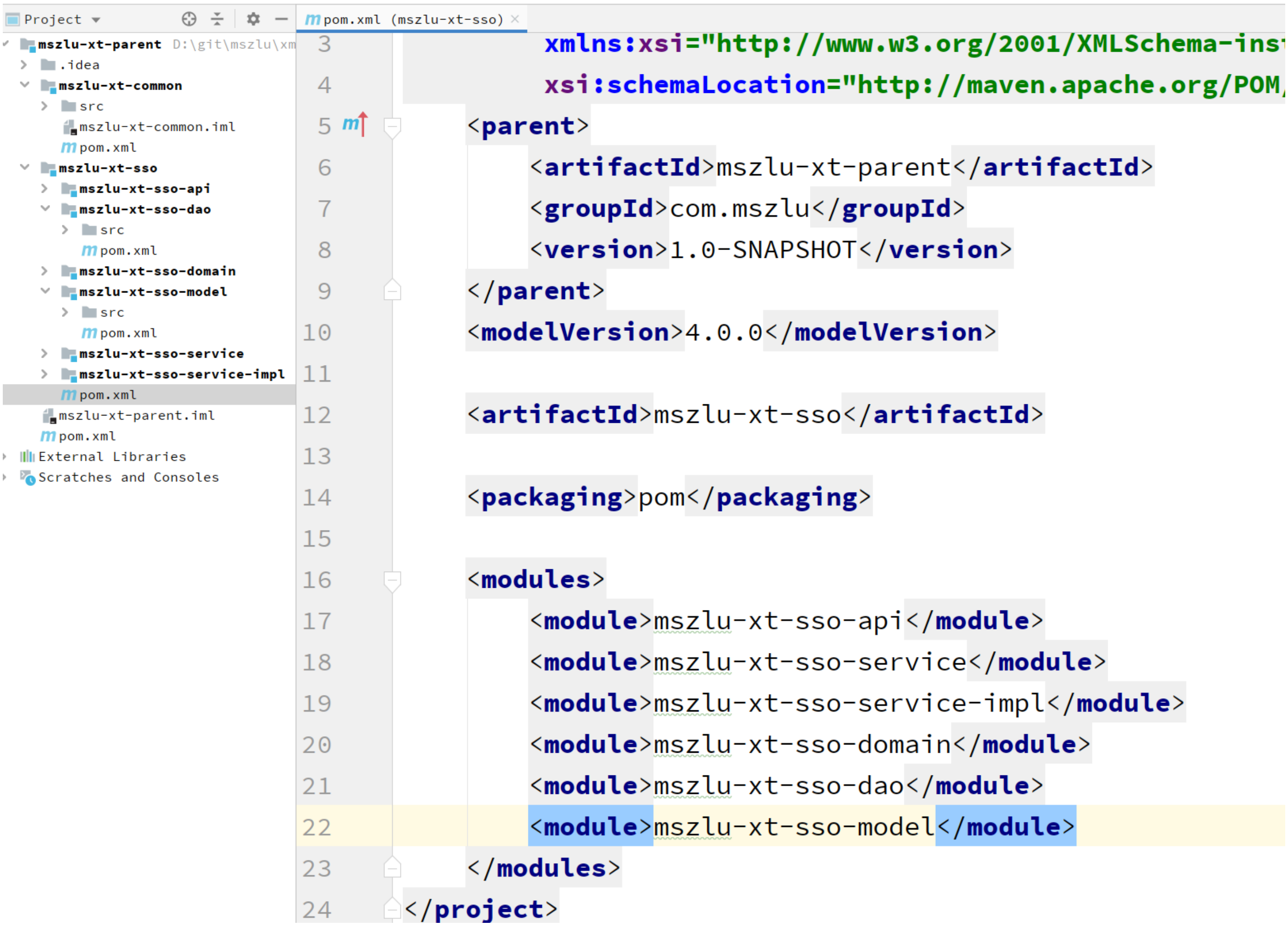
3. 搭建mszlu-xt-sso工程

   pom文件:

```
<?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>mszlu-xt-parent</artifactId>
        <groupId>com.mszlu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-sso</artifactId>

    <packaging>pom</packaging>
</project>
```

> *mszlu-xt-sso是父工程，在下面创建对应的子模块,api,service,domain,dao,model等*

搭建好的项目如图所示：



```xml
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
         xsi:schemaLocation="http://maven.apache.org/POM/
    <parent>
        <artifactId>mszlu-xt-parent</artifactId>
        <groupId>com.mszlu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-sso</artifactId>

    <packaging>pom</packaging>

    <modules>
        <module>mszlu-xt-sso-api</module>
        <module>mszlu-xt-sso-service</module>
        <module>mszlu-xt-sso-service-impl</module>
        <module>mszlu-xt-sso-domain</module>
        <module>mszlu-xt-sso-dao</module>
        <module>mszlu-xt-sso-model</module>
    </modules>
</project>
```

1.3 统一返回对象

CallResult: 所有的API接口统一返回此对象，便于和前端进行交互, 类上使用的@Data等注解使用lombok。

*lombok*是一个可以通过简单的注解的形式来帮助我们简化消除一些必须有但显得很臃肿的Java 代码的工具。比如可以省去set,get方法。

```java
package com.mszlu.xt.common.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;
```

```
/**
 * @author Jarno
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class CallResult<T> implements Serializable {

    private int code;

    private String message;

    private T result;

    public static <T>CallResult<T> success() {
        return new CallResult<T>
(BusinessCodeEnum.DEFAULT_SUCCESS.getCode(),
BusinessCodeEnum.DEFAULT_SUCCESS.getMsg(), null);
    }

    public static <T>CallResult<T> success(T result) {
        return new CallResult<T>
(BusinessCodeEnum.DEFAULT_SUCCESS.getCode(),
BusinessCodeEnum.DEFAULT_SUCCESS.getMsg(), result);
    }
    public static <T>CallResult<T> success(int code, String message,T
result) {
        return new CallResult<T>(code, message, result);
    }

    public static <T>CallResult<T> fail() {
        return new CallResult<T>
(BusinessCodeEnum.DEFAULT_SYS_ERROR.getCode(),
BusinessCodeEnum.DEFAULT_SYS_ERROR.getMsg(), null);
    }
    public static <T>CallResult<T> fail(T result) {
        return new CallResult<T>
(BusinessCodeEnum.DEFAULT_SYS_ERROR.getCode(),
BusinessCodeEnum.DEFAULT_SYS_ERROR.getMsg(), result);
    }

    public static <T>CallResult<T> fail(int code, String message) {
        return new CallResult<T>(code, message, null);
    }

    public static <T>CallResult<T> fail(int code, String message, T
result) {
        return new CallResult<T>(code, message, result);
    }

    public  boolean isSuccess(){
        return this.code == BusinessCodeEnum.DEFAULT_SUCCESS.getCode();
    }

}
```

BusinessCodeEnum: 枚举，返回的状态码以及消息定义

```
package com.mszlu.xt.common.model;

import java.util.HashMap;
import java.util.Map;

/**
 * @author Jarno
 */
public enum BusinessCodeEnum {
    /**
     * 码样式：【CCCBBOOXXX】
     * 编码示例说明：
     * CCC 中心编码&业务系统
     * BB   业务类型
     * OO  操作类型
     * XXX具体编码（000：表示成功，999：系统异常，998：数据库异常，NNN：其它，
100：参数异常，200：业务异常）
     * 200开头代码系统默认，其余系统使用10-199之间
     * */
    DEFAULT_SUCCESS(2000000000,"default success"),
    DEFAULT_SYS_ERROR(2000000999,"系统错误"),
    CHECK_PARAM_NO_RESULT(2000000100,"检测参数无结果"),
    CHECK_BIZ_NO_RESULT(2000000101,"检查业务无结果"),
    CHECK_ACTION_NO_RESULT(2000000102,"检查执行情况无结果"),
    CHECK_PARAM_NOT_MATCH(1000000001,"检查参数不匹配"),
```

```java
    CHECK_BIZ_ERROR_VALIDATE_CODE(1000000002,"检查业务验证码错误"),
    CHECK_BIZ_ERROR_MOBILE_USED(1000000003,"检查业务电话号码已被使用"),
    //login
    NO_LOGIN(1000000200,"未登录"),
    //topic
    TOPIC_PARAM_ERROR(1011004100,"参数有误"),
    TOPIC_RAND_ERROR(1011004101,"没有习题了"),
    TOPIC_NOT_EXIST(1011004102,"topic 不存在"),
    TOPIC_NO_PRACTICE(1011004103,"没有练习题"),
    TOPIC_NO_PROBLEM(1011004104,"没有错题"),
    PRACTICE_NO_EXIST(1011004105,"练习题不存在"),
    PRACTICE_FINISHED(1011004106,"练习已经完成"),
    PRACTICE_CANCEL(1011004107,"练习已经取消"),
    //course
    COURSE_NO_BUY(1021004101,"not buy course"),
    COURSE_NOT_EXIST(1021004102,"course not exist"),
    //wx
    LOGIN_WX_NO_LEGAL(1031001101,"不合法的请求"),
    LOGIN_WX_NOT_USER_INFO(1031001102,"无法获取用户信息"),
    //wx pay
    PAY_ORDER_CREATE_FAIL(1041001101,"创建订单失败"),
    ORDER_NOT_EXIST(1041001102,"订单号不存在"),
    ORDER_NOT_CANCEL(1041001103,"已付款的订单不能取消"),
    ;


    private static final Map<Integer, BusinessCodeEnum> codeMap = new HashMap<Integer, BusinessCodeEnum>((int)(BusinessCodeEnum.values().length/0.75)+1);

    static{
        for(BusinessCodeEnum businessCodeEnum: values()){
            codeMap.put(businessCodeEnum.getCode(), businessCodeEnum);
        }
    }

    /**
     * 根据code获取枚举值
     * @param code
     * @return
     */
    public static BusinessCodeEnum valueOfCode(int code){
        return codeMap.get(code);
    }

    private int code;
    private String msg;

    BusinessCodeEnum(int code, String msg) {
        this.code = code;
        this.msg = msg;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }
}
```

1.4 service模板

本项目采用了controller-service-dao的开发模式，但考虑到service层有大量的业务逻辑代码，一是复用起来较为困难，在团队规模比较大的时候，合作开发容易出现问题，二是service层的代码逻辑较为固定，可以进行抽取。

综合以上考虑，定义了serviceTemplate 来实现service层代码，业务逻辑代码放入domain模块。

权责归属

service模板代码放入common模块

ServiceTemplate: 模板定义

代码中有注释

```java
package com.mszlu.xt.common.service;
```

```
import com.mszlu.xt.common.model.CallResult;

/**
 * @author Jarno
 */
public interface ServiceTemplate {


    /**
     * run in  datasource and execute Transaction
     * @param action
     * @param <T>
     * @return
     */
    <T> CallResult<T> execute(TemplateAction<T> action);

    /**
     * run in  datasource and not execute Transaction
     * @param action
     * @param <T>
     * @return
     */
    <T> CallResult<T> executeQuery(TemplateAction<T> action);
}
```

TemplateAction：定义模板要执行的动作

```
package com.mszlu.xt.common.service;


import com.mszlu.xt.common.model.CallResult;

/**
 * @author Jarno
 */
public interface TemplateAction<T> {
    //第一步 检查参数
    CallResult<T> checkParam();
  //第二步 检查业务逻辑是否符合需求
    CallResult<T> checkBiz();
  //第三步 执行
```

```
    CallResult<T> doAction();
  //第四步 执行完成之后 要进行的操作
    void finishUp(CallResult<T> callResult);
}
```

AbstractTemplateAction: 抽象类，将checkParam，checkBiz，finishUp进行默认实现，一些情况下，service代码无需执行所有的动作

```
package com.mszlu.xt.common.service;


import com.mszlu.xt.common.model.CallResult;

/**
 * @author Jarno
 */
public abstract class AbstractTemplateAction<T> implements
TemplateAction<T> {
    @Override
    public CallResult<T> checkParam() {
        return CallResult.success();
    }

    @Override
    public CallResult<T> checkBiz() {
        return CallResult.success();
    }

    @Override
    public void finishUp(CallResult<T> callResult) {

    }
}
```

ServiceTemplateImpl: 模板实现类

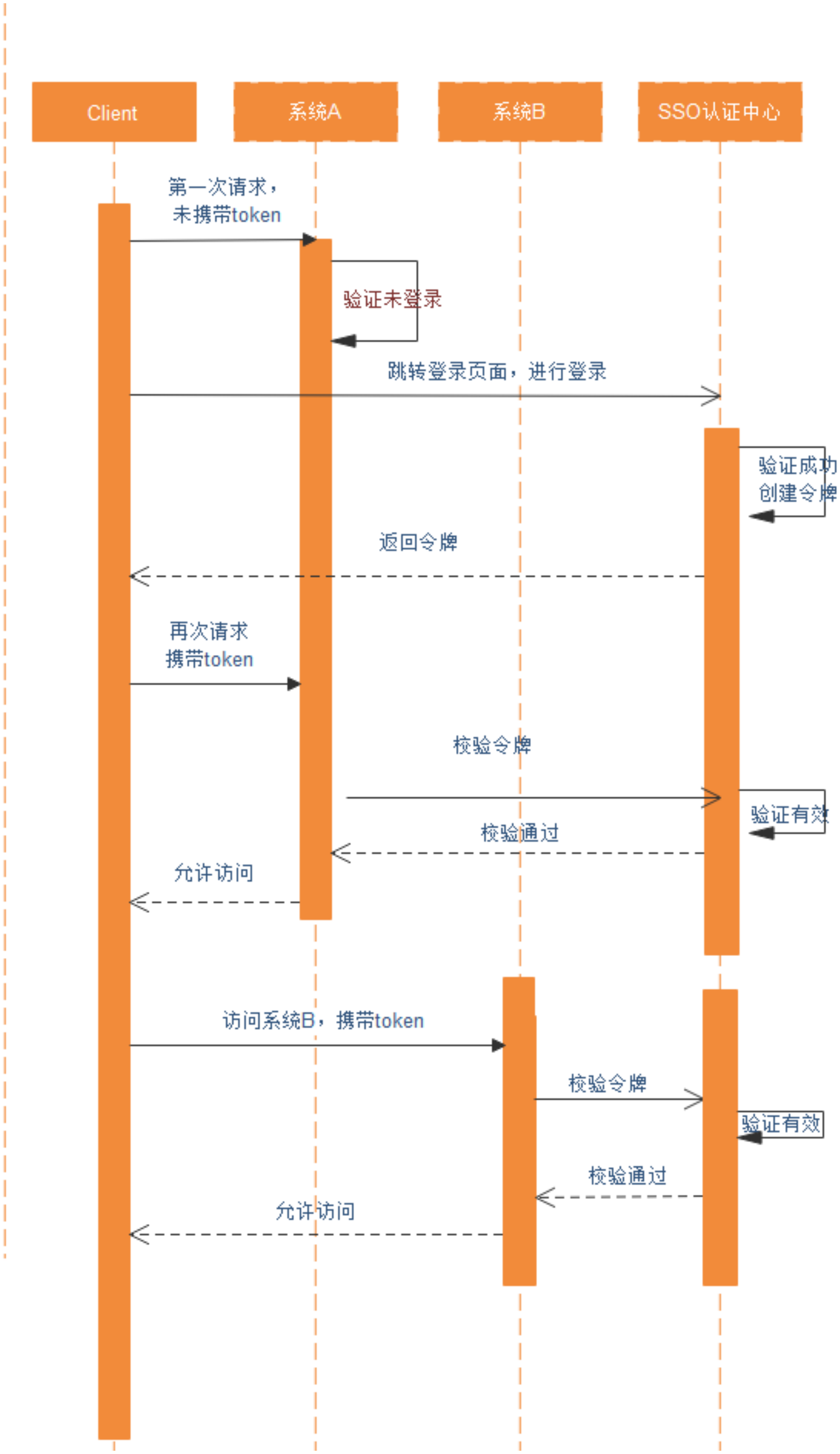
```
package com.mszlu.xt.common.service;


import com.mszlu.xt.common.model.BusinessCodeEnum;
import com.mszlu.xt.common.model.CallResult;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import
org.springframework.transaction.interceptor.TransactionAspectSupport;

/**
 * @author Jarno
 */
@Component
@Slf4j
public class ServiceTemplateImpl implements ServiceTemplate {

    @Override
    public <T> CallResult<T> execute(TemplateAction<T> action) {
        try{
            CallResult<T> callResult = action.checkParam();
            if(callResult==null){
                log.warn("execute: Null result while checkParam");
                return
CallResult.fail(BusinessCodeEnum.CHECK_PARAM_NO_RESULT.getCode(),
BusinessCodeEnum.CHECK_PARAM_NO_RESULT.getMsg());
            }
            if(!callResult.isSuccess()){
                return callResult;
            }
            callResult = action.checkBiz();
            if(callResult==null){

 TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
                log.warn("execute: Null result while checkBiz");
                return
CallResult.fail(BusinessCodeEnum.CHECK_BIZ_NO_RESULT.getCode(),
BusinessCodeEnum.CHECK_BIZ_NO_RESULT.getMsg());
            }
            if(!callResult.isSuccess()){

 TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
                return callResult;
            }
            long start = System.currentTimeMillis();
            CallResult<T> cr= action.doAction();
            log.info("execute datasource method run time:{}ms",
System.currentTimeMillis() - start);
            if (cr == null){
```

```
 TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
                return
CallResult.fail(BusinessCodeEnum.CHECK_ACTION_NO_RESULT.getCode(),
BusinessCodeEnum.CHECK_ACTION_NO_RESULT.getMsg());
            }
            if (!cr.isSuccess()){

 TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
                return cr;
            }
            if(cr.isSuccess()){
                action.finishUp(cr);
            }
            return cr;
        }catch(Exception e){

 TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
            e.printStackTrace();
            log.error("excute error", e);
            return CallResult.fail();
        }
    }

    @Override
    public <T> CallResult<T> executeQuery(TemplateAction<T> action) {
        try{
            CallResult<T> callResult = action.checkParam();
            if(callResult==null){
                log.warn("executeQuery: Null result while checkParam");
                return
CallResult.fail(BusinessCodeEnum.CHECK_PARAM_NO_RESULT.getCode(),
BusinessCodeEnum.CHECK_PARAM_NO_RESULT.getMsg());
            }
            if(!callResult.isSuccess()){
                return callResult;
            }
            callResult = action.checkBiz();
            if(callResult==null){
                log.warn("executeQuery: Null result while checkBiz");
                return
CallResult.fail(BusinessCodeEnum.CHECK_BIZ_NO_RESULT.getCode(),
BusinessCodeEnum.CHECK_BIZ_NO_RESULT.getMsg());
            }
            if(!callResult.isSuccess()){
                return callResult;
            }
            long start = System.currentTimeMillis();
            CallResult<T> cr= action.doAction();
            log.info("executeQuery datasource method run time:{}ms",
System.currentTimeMillis() - start);
            if (cr == null){
                return
CallResult.fail(BusinessCodeEnum.CHECK_ACTION_NO_RESULT.getCode(),
BusinessCodeEnum.CHECK_ACTION_NO_RESULT.getMsg());
            }
            if (!cr.isSuccess()){
                return cr;
            }
            if(cr.isSuccess()){
                action.finishUp(cr);
            }
            return cr;
        }catch(Exception e){
            e.printStackTrace();
            log.error("executeQuery error", e);
            return CallResult.fail();
        }
    }
}
```

1.5 单点登录介绍

> *举个简单的例子，阿里巴巴旗下有淘宝网，飞猪网等，用于如果登录了淘宝网，在访问飞猪网的时候，还需要登录吗?*
>
> *很明显不需要了，如果用户多次登录，体验性较差，对阿里技术团队来说，淘宝和飞猪都属于集团旗下的，用户表是一个，开发一次登录功能即可。*
>
> *单点登录：简称为SSO，SSO的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。*

单点登录流程：


Client
系统A
系统B
SSO认证中心
第一次请求，
未携带token
验证未登录
跳转登录页面，进行登录
验证成功
创建令牌
返回令牌
再次请求
携带token
校验令牌
验证有效
校验通过
允许访问
访问系统B，携带token
校验令牌
验证有效
校验通过
允许访问

2. 微信登录注册

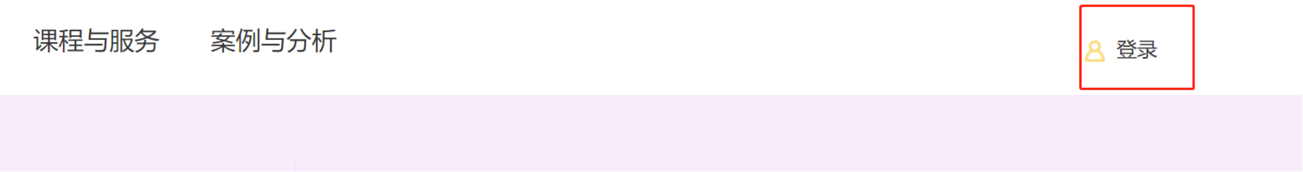
> *现在登录主流有两种方式，一种是手机验证码登录，一种就是第三方登录，其中微信登录最为主流。*
>
> *代码行数900行左右*

文档地址：https://developers.weixin.qq.com/doc/oplatform/Website_App/WeChat_Login/Wechat_Login.html

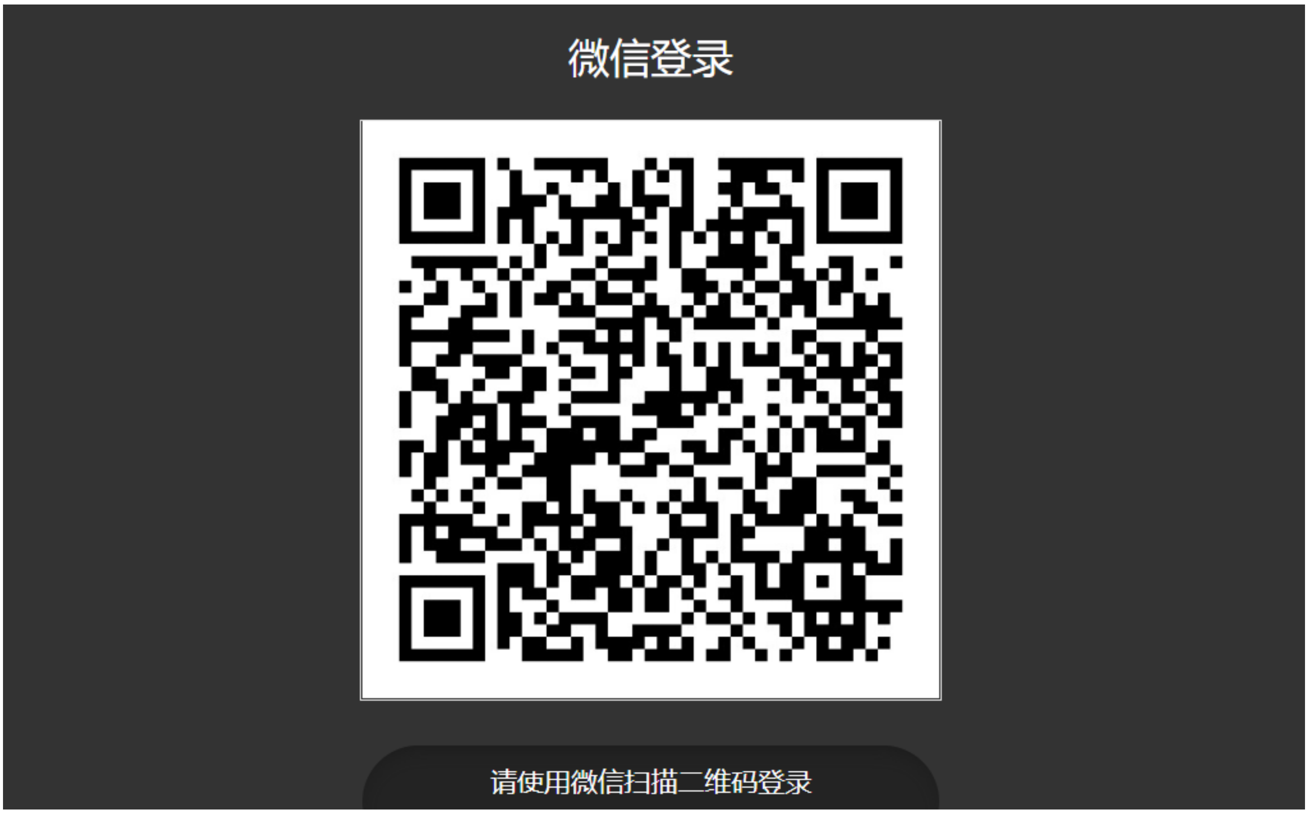
公众号登录：https://developers.weixin.qq.com/doc/offiaccount/OA_Web_Apps/Wechat_webpage_authorization.html

2.1 需求

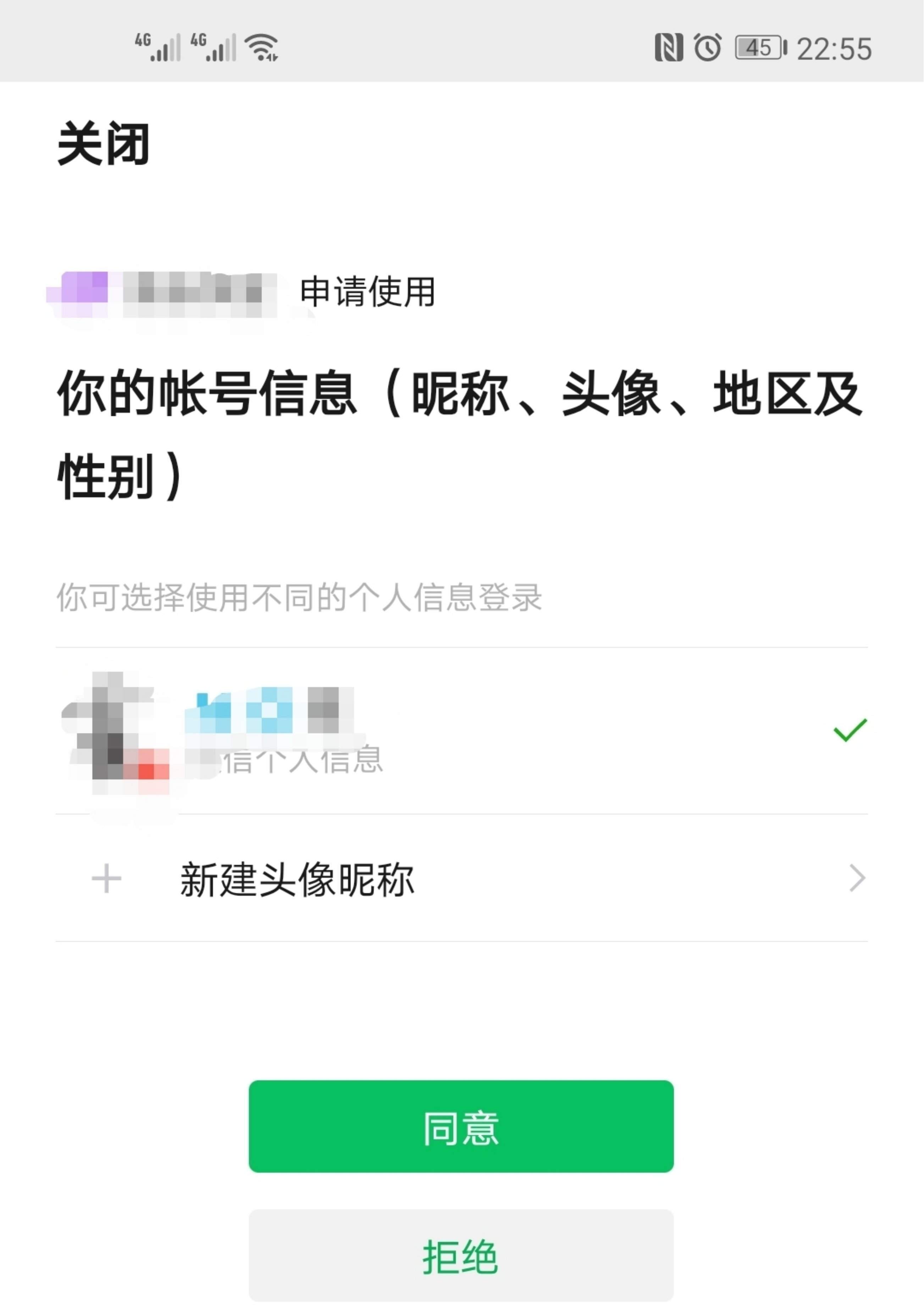
1. 点击页面的登录按钮



2. 弹出微信登录二维码



3. 微信扫码，授权登录



关闭

[redacted] 申请使用

你的帐号信息（昵称、头像、地区及性别）

你可选择使用不同的个人信息登录

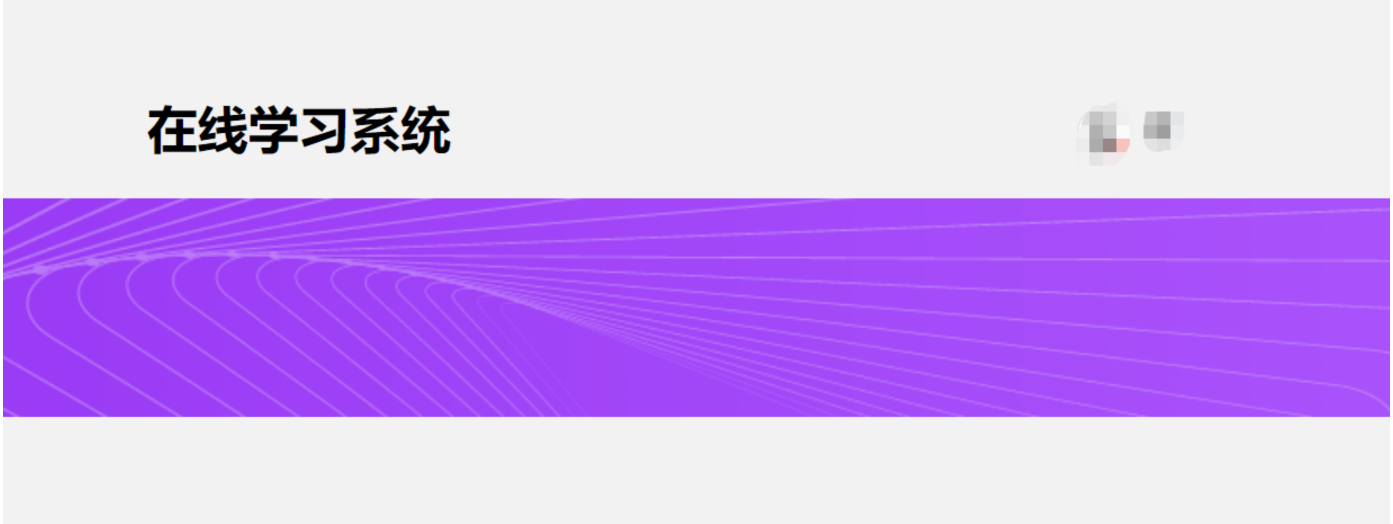
[redacted]
[redacted]信个人信息 ✓

＋ 新建头像昵称 ›

同意

拒绝

4. 页面跳转,登录成功

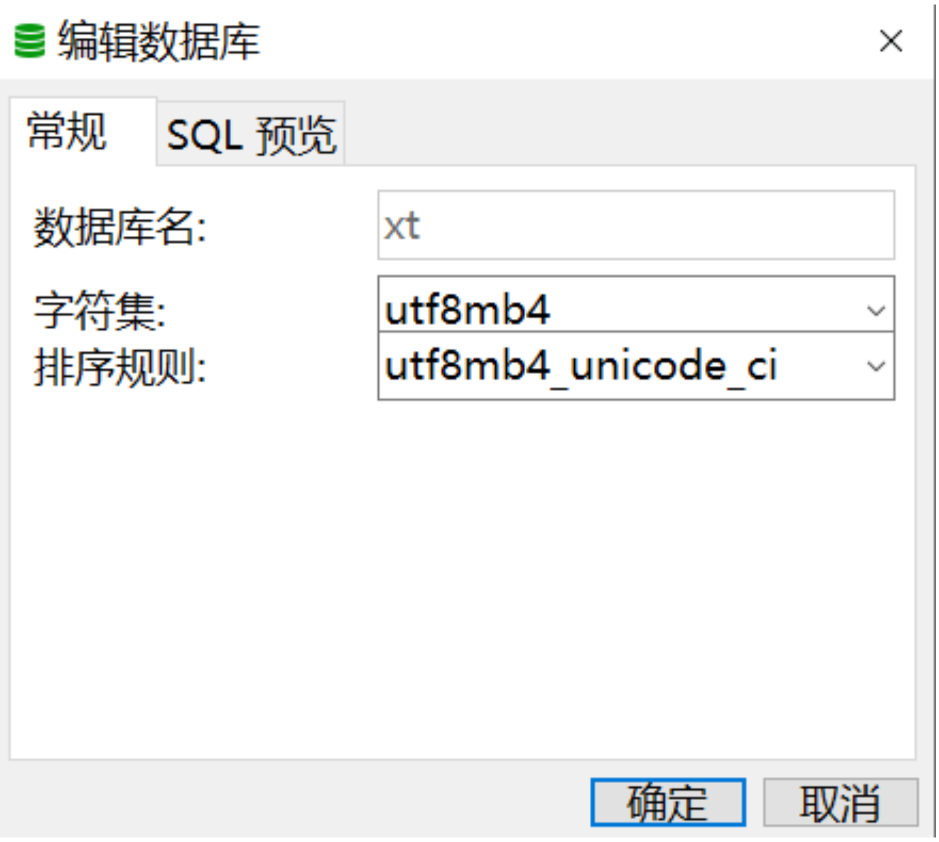


在线学习系统

2.2 数据库设计

登录需要涉及到用户表，使用微信登录，表中要涉及到微信的相关字段。

1. 新建数据库，注意字符集编码



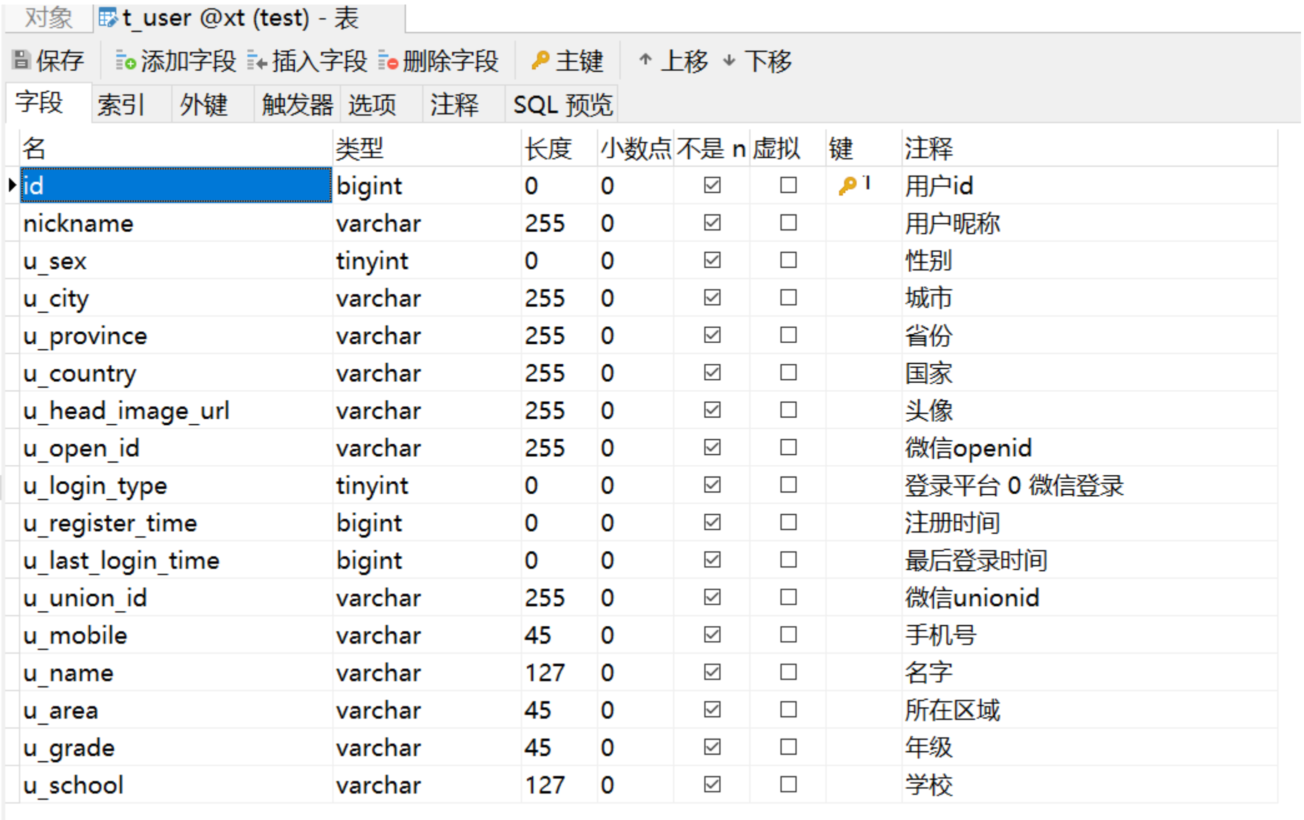
数据库名: xt

字符集: utf8mb4

排序规则: utf8mb4_unicode_ci

为什么要有utf8mb4这个编码?

2. 设计用户表t_user



名	类型	长度	小数点	不是 n	虚拟	键	注释
id	bigint	0	0	☑	☐	🔑1	用户id
nickname	varchar	255	0	☑	☐		用户昵称
u_sex	tinyint	0	0	☑	☐		性别
u_city	varchar	255	0	☑	☐		城市
u_province	varchar	255	0	☑	☐		省份
u_country	varchar	255	0	☑	☐		国家
u_head_image_url	varchar	255	0	☑	☐		头像
u_open_id	varchar	255	0	☑	☐		微信openid
u_login_type	tinyint	0	0	☑	☐		登录平台 0 微信登录
u_register_time	bigint	0	0	☑	☐		注册时间
u_last_login_time	bigint	0	0	☑	☐		最后登录时间
u_union_id	varchar	255	0	☑	☐		微信unionid
u_mobile	varchar	45	0	☑	☐		手机号
u_name	varchar	127	0	☑	☐		名字
u_area	varchar	45	0	☑	☐		所在区域
u_grade	varchar	45	0	☑	☐		年级
u_school	varchar	127	0	☑	☐		学校

3. 微信会关联微信开放平台，微信公众号等，所以微信的唯一标识以微信unionid为准。

2.3 步骤分析

1. 点击登录，调用sso服务，获取微信登录的二维码登录链接
2. sso服务根据微信提供的接口文档，生成二维码登录链接，返回数据到前端(生成链接，需要提供登录成功后，微信回调的链接地址)
3. 前端拿到链接，进行访问，微信自动跳转到微信登录页面
4. 手机扫码登录，授权登录
5. 微信调用回调地址
6. 回调的接口业务中，做登录相关的业务
    1. 如果用户已经存在，登录
    2. 如果用户不存在，注册
    3. 使用jwt生成token，记录到cookie中，同时记录到redis当中
    4. 还需要记录用户上一次登录的信息，如果在其他机器登录，那么之前的登录踢掉线
    5. 用户访问其余资源的时候，需要验证token，token认证通过之后，才能访问

2.4 编码

先将sso模块的依赖关系理清：

api依赖service-impl, service-impl依赖service，domain等，service依赖model，domain依赖dao和common,model,dao谁都不依赖.

sso-parent:

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.mszlu.xt</groupId>
                <artifactId>mszlu-xt-sso-service-impl</artifactId>
                <version>1.0-SNAPSHOT</version>
            </dependency>
            <dependency>
                <groupId>com.mszlu.xt</groupId>
                <artifactId>mszlu-xt-sso-service</artifactId>
                <version>1.0-SNAPSHOT</version>
            </dependency>
            <dependency>
                <groupId>com.mszlu.xt</groupId>
                <artifactId>mszlu-xt-sso-model</artifactId>
                <version>1.0-SNAPSHOT</version>
            </dependency>
            <dependency>
                <groupId>com.mszlu.xt</groupId>
                <artifactId>mszlu-xt-sso-domain</artifactId>
                <version>1.0-SNAPSHOT</version>
            </dependency>
            <dependency>
                <groupId>com.mszlu.xt</groupId>
                <artifactId>mszlu-xt-sso-dao</artifactId>
                <version>1.0-SNAPSHOT</version>
            </dependency>
            <dependency>
                <groupId>com.mszlu.xt</groupId>
                <artifactId>mszlu-xt-common</artifactId>
                <version>1.0-SNAPSHOT</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

sso-api:

```
  <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-sso-service-impl</artifactId>
        </dependency>
```

sso-service-impl:

```
<?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>mszlu-xt-sso</artifactId>
        <groupId>com.mszlu.xt</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-sso-service-impl</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-sso-service</artifactId>
        </dependency>
        <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-sso-domain</artifactId>
        </dependency>
    </dependencies>
</project>
```

sso-domain:

```
<?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>mszlu-xt-sso</artifactId>
        <groupId>com.mszlu.xt</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-sso-domain</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-sso-model</artifactId>
        </dependency>
        <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-sso-dao</artifactId>
        </dependency>
        <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-common</artifactId>
        </dependency>
    </dependencies>
</project>
```

sso-service:

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

```xml
<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>mszlu-xt-sso</artifactId>
        <groupId>com.mszlu.xt</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-sso-service</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.mszlu.xt</groupId>
            <artifactId>mszlu-xt-sso-model</artifactId>
        </dependency>
    </dependencies>
</project>
```

2.4.1 获取微信登录二维码

1. 在mszlu-xt-sso-api 中新建LoginApi

```xml
<?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>mszlu-xt-sso</artifactId>
        <groupId>com.mszlu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mszlu-xt-sso-api</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
    </dependencies>

</project>
```

```
package com.mszlu.xt.sso.api;

import com.mszlu.common.model.CallResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("login")
public class LoginApi {

    @Autowired
    private LoginService loginService;

    /**
     * 获取微信登陆二维码地址
     * @return
     */
    @RequestMapping("/getQRCodeUrl")
    @ResponseBody
    public CallResult getQRCodeUrl() {
        return loginService.getQRCodeUrl();
    }
}
```

2. 在mszlu-xt-sso-service中新建LoginService

```
    <dependency>
            <groupId>com.mszlu</groupId>
            <artifactId>mszlu-xt-common</artifactId>
        </dependency>
```

```
package com.mszlu.xt.sso.service;

import com.mszlu.common.model.CallResult;

public interface LoginService {

    /**
     * 获取微信登录的二维码地址
     * @return
     */
    CallResult getQRCodeUrl();

}
```

3. 在mszlu-xt-sso-service-impl中新建AbstractService，方便使用service模板

```
package com.mszlu.xt.sso.service.impl;


import com.mszlu.common.service.ServiceTemplate;
import org.springframework.beans.factory.annotation.Autowired;

public abstract class AbstractService {

    @Autowired
    protected ServiceTemplate serviceTemplate;
}
```

4. 在mszlu-xt-sso-service-impl中新建LoginServiceImpl

```
package com.mszlu.xt.sso.service.impl;

import com.mszlu.common.model.CallResult;
import com.mszlu.xt.sso.service.LoginService;
import me.chanjar.weixin.mp.api.WxMpService;
import org.apache.commons.codec.digest.DigestUtils;
import org.joda.time.DateTime;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class LoginServiceImpl extends AbstractService implements
LoginService {

    @Autowired
    private WxMpService wxMpService;
    @Autowired
    private WxOpenConfig wxOpenConfig;

    @Override
    public CallResult getQRCodeUrl() {
        // 生成 state 参数，用于防止 csrf
        String date = new DateTime().toString("yyyyMMddHHmmss");
        String state =
DigestUtils.md5Hex(wxOpenConfig.csrfKey+date);
        String url =
wxMpService.buildQrConnectUrl(wxOpenConfig.redirectUrl,
wxOpenConfig.scope, state);
        return CallResult.success(url);
    }
}
```

5. 代码中涉及到的配置，依赖，第三方工具等

    1. WxMpService类 是开源微信java sdk库，地址 https://github.com/Wechat-Group/WxJava中用于微信公众号API的Service。

       在common模块中新建WxOpenConfig

       ```
       package com.mszlu.common.wx.config;

       import lombok.Data;
       import me.chanjar.weixin.mp.api.WxMpService;
       ```

```
import me.chanjar.weixin.mp.api.impl.WxMpServiceImpl;
import me.chanjar.weixin.mp.config.impl.WxMpDefaultConfigImpl;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@Data
public class WxOpenConfig {

    @Value("${wx.open.config.appid}")
    private String loginAppid;
    @Value("${wx.open.config.secret}")
    private String loginSecret;
    @Value("${wx.open.config.csrfKey}")
    public String csrfKey;
    @Value("${wx.open.config.redirectUrl}")
    public String redirectUrl;
    @Value("${wx.open.config.scope}")
    public String scope;

    @Bean
    public WxMpService wxMpService() {
        WxMpService service = new WxMpServiceImpl();
        WxMpDefaultConfigImpl wxMpConfigStorage = new WxMpDefaultConfigImpl();
        wxMpConfigStorage.setAppId(loginAppid);
        wxMpConfigStorage.setSecret(loginSecret);
        service.setWxMpConfigStorage(wxMpConfigStorage);
        return service;
    }

}
```

配置文件

application.properties

```
##启动端口号
server.port=8118
spring.application.name=xt-sso
##接口访问路径
server.servlet.context-path=/api/sso
##本机开发环境
spring.profiles.active=dev

mybatis-plus.global-config.db-config.table-prefix=t_
```

application-dev.properties

```
#数据库配置
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/xt?useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root

#微信开放平台创建的网站应用的appid
wx.open.config.appid=wxec42a03d0976866f
#微信开放平台创建的网站应用的appsecret
wx.open.config.secret=ae395841b30780e0819d08d43983f386
#登录作用域
wx.open.config.scope=snsapi_login
#回调认证状态，用于认证回调的有效性
wx.open.config.csrfKey=smartpig

#微信开放平台创建的网站 设置的授权回调域
wx.open.config.redirectUrl=http://xxxx/sso/login/wxLoginCallBack
```

2. 使用到的依赖
    1. common模块添加依赖

```
<!--时间处理工具类-->
        <dependency>
          <groupId>joda-time</groupId>
          <artifactId>joda-time</artifactId>
        </dependency>
      <!--编解码工具类-->
        <dependency>
```

```
          <groupId>commons-codec</groupId>
          <artifactId>commons-codec</artifactId>
        </dependency>
      <!--wx-java相关，公众号，支付，开放平台等-->
        <dependency>
          <groupId>com.github.binarywang</groupId>
          <artifactId>weixin-java-mp</artifactId>
        </dependency>
        <dependency>
          <groupId>com.github.binarywang</groupId>
          <artifactId>weixin-java-pay</artifactId>
        </dependency>
        <dependency>
          <groupId>com.github.binarywang</groupId>
          <artifactId>weixin-java-open</artifactId>
        </dependency>
      <dependency>
      <groupId>com.alibaba</groupId>
          <artifactId>fastjson</artifactId>
    </dependency>
```

2. 在mszlu-xt-sso-api中的config包下，扫描common中有关的配置类，使之生效

```
package com.mszlu.xt.sso.config;

import
org.springframework.context.annotation.ComponentScan;
import org.springframework.stereotype.Component;

@Component
@ComponentScan({"com.mszlu.common.wx.config","com.mszlu.co
mmon.service"})
public class InitConfig {
}
```

> *启动测试*

```
package com.mszlu.xt.sso;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SSOApp {

    public static void main(String[] args) {
        SpringApplication.run(SSOApp.class,args);
    }
}
```

浏览器访问http://localhost:8118/api/sso/login/getQRCodeUrl，得到结果后，点击result中的链接，获得微信的二维码登录页面

使用nginx，用前端页面进行测试：

```
server {
        listen       80;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        location / {
              proxy_pass http://127.0.0.1:8080;
        }
        location /api/sso {
              proxy_pass http://127.0.0.1:8118;
            }
         location /api {
              proxy_pass http://127.0.0.1:8128;
            }
      }
```

2.4.2 微信登录回调实现

1. LoginApi中添加接口，用于实现回调，做登录注册相关的业务

```
//登录参数，放入model模块
package com.mszlu.xt.model.params.login;

import lombok.Data;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Data
public class LoginParam {
    private String username;
    private String password;
    //wx回调的传参
    private String code;
    private String state;
    private HttpServletResponse response;
    private HttpServletRequest request;

    private String cookieUserId;

}
```

model依赖：

```
<dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-core</artifactId>
        </dependency>
    </dependencies>
```

```
package com.mszlu.xt.sso.api;

```

```java
import com.mszlu.xt.common.model.CallResult;
import com.mszlu.xt.sso.model.params.LoginParam;
import com.mszlu.xt.sso.service.LoginService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

///api/sso/login/getQRCodeUrl

//controller注解 默认是调整页面的
@Controller
@RequestMapping("login")
public class LoginApi {

    @Autowired
    private LoginService loginService;

    @PostMapping("getQRCodeUrl")
    @ResponseBody
    public CallResult getQRCodeUrl(){
        //controller 职责 接收参数  一个处理结果
        return loginService.getQRCodeUrl();
    }

    //redirect_uri?code=CODE&state=STATE
    @GetMapping("wxLoginCallBack")
    public String wxLoginCallBack(HttpServletRequest request,
                                  HttpServletResponse response,
                                  String code,
                                  String state){
        //为了service层统一，所有的api层的参数处理 ，都放入loginParams中
        LoginParam loginParam = new LoginParam();
        loginParam.setCode(code);
        loginParam.setState(state);
        loginParam.setRequest(request);
        //后续 登录成功之后，要生成token，提供给前端，把token放入对应的cookie
        // response.addCookie();
        loginParam.setResponse(response);
        CallResult callResult =
loginService.wxLoginCallBack(loginParam);
        if (callResult.isSuccess()){
            return "redirect:http://www.lzxtedu.com/course";
        }else{
            return "redirect:http://www.lzxtedu.com";
        }
    }
}
```

2. LoginServiceImpl:

```
package com.mszlu.xt.sso.service.impl;

import com.mszlu.xt.common.constants.RedisKey;
import com.mszlu.xt.common.model.CallResult;
import com.mszlu.xt.common.service.AbstractTemplateAction;
import com.mszlu.xt.common.wx.config.WxOpenConfig;
import com.mszlu.xt.sso.domain.LoginDomain;
import com.mszlu.xt.sso.domain.repository.LoginDomainRepository;
import com.mszlu.xt.sso.model.params.LoginParam;
import com.mszlu.xt.sso.service.LoginService;
import me.chanjar.weixin.mp.api.WxMpService;
import org.joda.time.DateTime;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.UUID;
import java.util.concurrent.TimeUnit;

@Service
public class LoginServiceImpl extends AbstractService implements
LoginService {


    @Autowired
    private LoginDomainRepository loginDomainRepository;

    @Override
    public CallResult getQRCodeUrl() {
        LoginDomain loginDomain =
loginDomainRepository.createDomain(new LoginParam());
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>(){
            @Override
            public CallResult<Object> doAction() {
                return loginDomain.buildQrConnectUrl();
            }
        });
    }

    @Override
    @Transactional
    public CallResult wxLoginCallBack(LoginParam loginParam) {

        LoginDomain loginDomain =
loginDomainRepository.createDomain(loginParam);
        //带有事务的执行操作
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>(){

            @Override
            public CallResult<Object> checkBiz() {
                //检查业务参数
                return loginDomain.checkWxLoginCallBackBiz();
            }

            @Override
            public CallResult<Object> doAction() {
                //写业务逻辑的
                return loginDomain.wxLoginCallBack();
            }
        });
    }
}
```

3. 在domain模块中创建LoginDomainRepository和LoginDomain:

   domain依赖:

```
     <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
```

```
package com.mszlu.xt.sso.domain;

import com.mszlu.xt.common.constants.RedisKey;
import com.mszlu.xt.common.model.BusinessCodeEnum;
import com.mszlu.xt.common.model.CallResult;
import com.mszlu.xt.common.utils.JwtUtil;
import com.mszlu.xt.sso.dao.data.User;
import com.mszlu.xt.sso.domain.repository.LoginDomainRepository;
import com.mszlu.xt.sso.model.enums.LoginType;
import com.mszlu.xt.sso.model.params.LoginParam;
import com.mszlu.xt.sso.model.params.UserParam;
import me.chanjar.weixin.mp.bean.result.WxMpOAuth2AccessToken;
import me.chanjar.weixin.mp.bean.result.WxMpUser;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;
import java.util.concurrent.TimeUnit;

/**
 * 专门处理和登录相关的操作
 */
public class LoginDomain {

    private LoginDomainRepository loginDomainRepository;

    private LoginParam loginParam;

    private String secretKey = "mszlu!@#$%xtsso&^#$#@@";

    public LoginDomain(LoginDomainRepository loginDomainRepository,
LoginParam loginParam) {
        this.loginDomainRepository = loginDomainRepository;
        this.loginParam = loginParam;
    }

    public CallResult<Object> buildQrConnectUrl() {
        String url = this.loginDomainRepository.buildQrUrl();
```

```java
        return CallResult.success(url);
    }

    public CallResult<Object> checkWxLoginCallBackBiz() {
        //主要检查 state是否是合法的
        //csrf的检测
        String state = loginParam.getState();
        //去redis检测 是否 state 为key的值存在，如果不存在 ，证明不合法
        boolean isVerify = loginDomainRepository.checkState(state);
        if (!isVerify){
            return CallResult.fail(BusinessCodeEnum.CHECK_BIZ_NO_RESULT.getCode(),"参数不合法");
        }
        return CallResult.success();
    }

    public CallResult<Object> wxLoginCallBack() {
        String code = loginParam.getCode();
        try {
            //2. 下次进行登录的时候，如果refreshToken存在，可以直接获取accessToken，不需要用户重新授权
            String refreshToken = loginDomainRepository.redisTemplate.opsForValue().get(RedisKey.REFRESH_TOKEN);
            WxMpOAuth2AccessToken wxMpOAuth2AccessToken = null;
            if (refreshToken == null) {
                //1. 通过code获取到accessToken和refreshToken ,
                wxMpOAuth2AccessToken = loginDomainRepository.wxMpService.oauth2getAccessToken(code);
                refreshToken = wxMpOAuth2AccessToken.getRefreshToken();
                // 需要保存refreshToken 保存在redis当中，过期时间 设置为28天

 loginDomainRepository.redisTemplate.opsForValue().set(RedisKey.REFRESH_TOKEN, refreshToken, 28, TimeUnit.DAYS);
            }else{
                wxMpOAuth2AccessToken = loginDomainRepository.wxMpService.oauth2refreshAccessToken(refreshToken);
            }
            //3. 通过accessToken获取到了微信的用户信息（openid,unionId）unionId在web端，公众号端，手机端唯一的
            WxMpUser wxMpUser = loginDomainRepository.wxMpService.oauth2getUserInfo(wxMpOAuth2AccessToken, "zh_CN");
            String unionId = wxMpUser.getUnionId();
            //4. 需要判断unionId在数据库中的user表中 是否存在 存在就更新最后登录时间 不存在 注册
            User user = this.loginDomainRepository.createUserDomain(new UserParam()).findUserByUnionId(unionId);
            boolean isNew = false;
            if (user == null){
                //注册
                user = new User();
                Long currentTime = System.currentTimeMillis();
                user.setNickname(wxMpUser.getNickname());
                user.setHeadImageUrl(wxMpUser.getHeadImgUrl());
                user.setSex(wxMpUser.getSex());
                user.setOpenid(wxMpUser.getOpenId());
                user.setLoginType(LoginType.WX.getCode());
                user.setCountry(wxMpUser.getCountry());
                user.setCity(wxMpUser.getCity());
                user.setProvince(wxMpUser.getProvince());
                user.setRegisterTime(currentTime);
                user.setLastLoginTime(currentTime);
                user.setUnionId(wxMpUser.getUnionId());
                user.setArea("");
                user.setMobile("");
                user.setGrade("");
                user.setName(wxMpUser.getNickname());
                user.setSchool("");
                this.loginDomainRepository.createUserDomain(new UserParam()).saveUser(user);
                isNew = true;
            }

            //5. 使用jwt技术，生成token，需要把token存储起来
            String token = JwtUtil.createJWT(7 * 24 * 60 * 60 * 1000, user.getId(), secretKey);

 loginDomainRepository.redisTemplate.opsForValue().set(RedisKey.TOKEN+token,String.valueOf(user.getId()),7,TimeUnit.DAYS);
            //6. 因为付费课程，所以账号只能在一端登录，如果用户在其他地方登录，需要将当前的登录用户踢下线
            String oldToken =
loginDomainRepository.redisTemplate.opsForValue().get(RedisKey.LOGI
N_USER_TOKEN + user.getId());
            if (oldToken != null){
                //当前用户 之前在某一个设备登录过
                //在用户进行登录验证的时候，需要先验证token是否合法，然后去
redis查询是否存在token 不存在 代表不合法

 loginDomainRepository.redisTemplate.delete(RedisKey.TOKEN+token);
            }

 loginDomainRepository.redisTemplate.opsForValue().set(RedisKey.LOG
IN_USER_TOKEN + user.getId(),token);
            //7. 返回给前端token，存在cookie当中，下次请求的时候 从cookie
中获取token
            HttpServletResponse response =
loginParam.getResponse();
            Cookie cookie = new Cookie("t_token", token);
            cookie.setMaxAge(8*24*3600);
            cookie.setPath("/");
            response.addCookie(cookie);
            //8. 比如给用户 加积分，成就系统，任务系统
            //9. 需要记录日志，记录当前用户的登录行为  MQ+mongo进行日志记录
            //10.更新用户的最后登录时间
            if (!isNew){
                user.setLastLoginTime(System.currentTimeMillis());
                this.loginDomainRepository.createUserDomain(new
UserParam()).updateUser(user);
            }
            return CallResult.success();
        }catch (Exception e){
            e.printStackTrace();
            return
CallResult.fail(BusinessCodeEnum.LOGIN_WX_NOT_USER_INFO.getCode(),"
授权问题,无法获取用户信息");
        }
    }



}
```

```
package com.mszlu.xt.sso.domain.repository;
```

```
import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import
com.baomidou.mybatisplus.core.conditions.update.LambdaUpdateWrapper
;
import com.mszlu.xt.common.constants.RedisKey;
import com.mszlu.xt.common.wx.config.WxOpenConfig;
import com.mszlu.xt.sso.dao.UserMapper;
import com.mszlu.xt.sso.dao.data.User;
import com.mszlu.xt.sso.domain.LoginDomain;
import com.mszlu.xt.sso.domain.UserDomain;
import com.mszlu.xt.sso.model.params.LoginParam;
import com.mszlu.xt.sso.model.params.UserParam;
import me.chanjar.weixin.mp.api.WxMpService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.UUID;
import java.util.concurrent.TimeUnit;

@Component
public class LoginDomainRepository {

    @Autowired
    public WxMpService wxMpService;
    @Autowired
    private WxOpenConfig wxOpenConfig;
    @Autowired
    public StringRedisTemplate redisTemplate;
    @Autowired
    private UserDomainRepository userDomainRepository;


    public LoginDomain createDomain(LoginParam loginParam) {
        return new LoginDomain(this,loginParam);
    }

    public boolean checkState(String state) {
        Boolean isValid =
redisTemplate.hasKey(RedisKey.WX_STATE_KEY+state);
        return isValid != null && isValid;
    }
    public String buildQrUrl() {
        //        String csrfKey = wxOpenConfig.getCsrfKey();
//        String time = new DateTime().toString("yyyyMMddHHmmss");
//        csrfKey = csrfKey+"_"+time;
        String csrfKey = UUID.randomUUID().toString();

 redisTemplate.opsForValue().set(RedisKey.WX_STATE_KEY+csrfKey,"1",
60, TimeUnit.SECONDS);
        //csrf 跨站伪造攻击 http://xxxx/sso/login/wxLoginCallBack?
state=csrfKey 需要验证csrfKey是不是我们存储的,
        // 如果是就证明 此链接安全 不是伪造的
        //典型的场景 比如你登录了银行网站 这时候 你去访问了一个论坛，点击论坛
里面的一个图片 ，但是你的账户少钱了
        //将csrfKey放入redis当中，并设置有效期
        //会用到一个第三方的工具
        String url =
wxMpService.buildQrConnectUrl(wxOpenConfig.getRedirectUrl(),
wxOpenConfig.getScope(), csrfKey);
        return url;
    }



    public UserDomain createUserDomain(UserParam userParam) {
        return userDomainRepository.createDomain(userParam);
    }
}
```

loginType:

```
package com.mszlu.xt.sso.model.enums;

import java.util.HashMap;
import java.util.Map;

/**
 * @author Jarno
 */
public enum LoginType {
    /**
     * look name
     */
```

```java
    WX(0,"wx 登录");

    private static final Map<Integer, LoginType> CODE_MAP = new HashMap<>(3);

    static{
        for(LoginType topicType: values()){
            CODE_MAP.put(topicType.getCode(), topicType);
        }
    }

    /**
     * 根据code获取枚举值
     * @param code
     * @return
     */
    public static LoginType valueOfCode(int code){
        return CODE_MAP.get(code);
    }

    private int code;
    private String msg;

    LoginType(int code, String msg) {
        this.code = code;
        this.msg = msg;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }
}
```

4. User 和数据库表t_user进行一一对应

```java
package com.mszlu.xt.sso.dao.data;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import lombok.Data;

@Data
public class User {
    @TableId(type = IdType.ASSIGN_ID)
    private Long id;
    private String nickname;
    @TableField("u_sex")
    private Integer sex;
    @TableField("u_city")
    private String city;
    @TableField("u_province")
    private String province;
    @TableField("u_country")
    private String country;
    @TableField("u_head_image_url")
    private String headImageUrl;
    @TableField("u_open_id")
    private String openid;
    @TableField("u_login_type")
    private Integer loginType;
    @TableField("u_register_time")
    private Long registerTime;
    @TableField("u_last_login_time")
    private Long lastLoginTime;
    @TableField("u_union_id")
    private String unionId;
    @TableField("u_mobile")
    private String mobile;
    @TableField("u_area")
    private String area;
    @TableField("u_grade")
    private String grade;
    @TableField("u_school")
    private String school;
    @TableField("u_name")
    private String name;

}
```

5. 其中涉及到User表操作的行为，交由UserDomain来负责

```
package com.mszlu.xt.sso.domain;

import com.mszlu.xt.sso.dao.data.User;
import com.mszlu.xt.sso.domain.repository.UserDomainRepository;
import com.mszlu.xt.sso.model.params.UserParam;

/**
 * 用户领域，权责明确，用户的事情应该教给用户的domain来去实现
 */
public class UserDomain {

    private UserDomainRepository userDomainRepository;
    private UserParam userParam;

    public UserDomain(UserDomainRepository userDomainRepository,
UserParam userParam) {
        this.userDomainRepository = userDomainRepository;
        this.userParam = userParam;
    }

    public void updateUser(User user) {
        userDomainRepository.updateUser(user);
    }

    public void saveUser(User user) {
        userDomainRepository.saveUser(user);
    }

    public User findUserByUnionId(String unionId) {
        return userDomainRepository.findUserByUnionId(unionId);
    }
}
```

```
package com.mszlu.xt.sso.domain.repository;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import
com.baomidou.mybatisplus.core.conditions.update.LambdaUpdateWrapper
;
import com.mszlu.xt.sso.dao.UserMapper;
import com.mszlu.xt.sso.dao.data.User;
import com.mszlu.xt.sso.domain.UserDomain;
import com.mszlu.xt.sso.model.params.UserParam;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

@Component
public class UserDomainRepository {

    @Resource
    private UserMapper userMapper;

    public UserDomain createDomain(UserParam userParam) {
        return new UserDomain(this,userParam);
    }

    public User findUserByUnionId(String unionId) {
        LambdaQueryWrapper<User> queryWrapper = new
LambdaQueryWrapper<>();
        //limit 1是对应的一个优化，查到数据就不在检索了
        queryWrapper.eq(User::getUnionId,unionId).last("limit 1");
        return userMapper.selectOne(queryWrapper);
    }

    public void saveUser(User user) {
        userMapper.insert(user);
    }

    public void updateUser(User user) {
        LambdaUpdateWrapper<User> updateWrapper = new
LambdaUpdateWrapper<>();

 updateWrapper.set(User::getLastLoginTime,user.getLastLoginTime());
        updateWrapper.eq(User::getId,user.getId());
        userMapper.update(null, updateWrapper);
    }
}
```

6. 对应的UserMapper，使用mybatisplus，简化开发，单表查询无需写sql

```
package com.mszlu.xt.sso.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mszlu.xt.sso.dao.data.User;

public interface UserMapper extends BaseMapper<User> {
}
```

7. JWT工具类

```
package com.mszlu.common.utils;

import com.alibaba.fastjson.JSON;
import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Value;


import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;

public class JwtUtil {



//    @Value("${jwt.secret}")
//    private static String key;
    /**
     * 用户登录成功后生成Jwt
     * 使用Hs256算法  私匙使用用户密码
     *
     * @param ttlMillis jwt过期时间
     * @return
     */
    public static String createJWT(long ttlMillis, Long
userId,String key) {
        //指定签名的时候使用的签名算法，也就是header那部分，jjwt已经将这部分
内容封装好了。
        SignatureAlgorithm signatureAlgorithm =
SignatureAlgorithm.HS256;

        //生成JWT的时间
        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);

        //创建payload的私有声明（根据特定的业务需要添加，如果要拿这个做验证，
一般是需要和jwt的接收方提前沟通好验证方式的）
        Map<String, Object> claims = new HashMap<String, Object>();
        claims.put("id", userId);

        //生成签名的时候使用的秘钥secret,这个方法本地封装了的，一般可以从本地
配置文件中读取，切记这个秘钥不能外露哦。它就是你服务端的私钥，在任何场景都不应该
流露出去。一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了。

        //生成签发人
        Map<String, Object> value = new HashMap<String, Object>();
        claims.put("userId", userId);
        //下面就是在为payload添加各种标准声明和私有声明了
        //这里其实就是new一个JwtBuilder，设置jwt的body
        JwtBuilder builder = Jwts.builder()
                //如果有私有声明，一定要先设置这个自己创建的私有的声明，这个
是给builder的claim赋值，一旦写在标准的声明赋值之后，就是覆盖了那些标准的声明的
                .setClaims(claims)
                //设置jti(JWT ID)：是JWT的唯一标识，根据业务需要，这个可以
设置为一个不重复的值，主要用来作为一次性token,从而回避重放攻击。
                .setId(UUID.randomUUID().toString())
                //iat: jwt的签发时间
                .setIssuedAt(now)
                //代表这个JWT的主体，即它的所有人，这个是一个json格式的字符
串，可以存放什么userid，roldid之类的，作为什么用户的唯一标志。
                .setSubject(JSON.toJSONString(value))
                //设置签名使用的签名算法和签名使用的秘钥
                .signWith(signatureAlgorithm, key);
        if (ttlMillis >= 0) {
            long expMillis = nowMillis + ttlMillis;
            Date exp = new Date(expMillis);
            //设置过期时间
            builder.setExpiration(exp);
        }
        return builder.compact();
    }


    /**
     * Token的解密
     * @param token 加密后的token
     * @return
     */
    public static Claims parseJWT(String token, String key) {
        //签名秘钥，和生成的签名的秘钥一模一样

        //得到DefaultJwtParser
        Claims claims = Jwts.parser()
                //设置签名的秘钥
                .setSigningKey(key)
                //设置需要解析的jwt
                .parseClaimsJws(token).getBody();
        return claims;
    }


    /**
     * 校验token
     * 在这里可以使用官方的校验，我这里校验的是token中携带的密码于数据库一致的话就校验通过
     * @return
     */
//    public static Boolean isVerify(String token, String key, Long userId) {
//        //签名秘钥，和生成的签名的秘钥一模一样
//    //Jwts.parser在执行parseClaimsJws(token)时如果token时间过期会抛出ExpiredJwtException异常
//        try {
//            //得到DefaultJwtParser
//            Claims claims = Jwts.parser()
//                    //设置签名的秘钥
//                    .setSigningKey(key)
//                    //设置需要解析的jwt
//                    .parseClaimsJws(token).getBody();
//            if (claims.get("id").equals(value)) {
//                return true;
//            }
//
//        }catch (ExpiredJwtException e){
//            e.printStackTrace();
//        }
//        return false;
//    }
    public static void main(String[] args) {
        String bbbb = JwtUtil.createJWT(1000 * 7 * 24 * 60 * 60, 1000L, "bbbb");
        System.out.println(bbbb);
        Claims bbbb1 = JwtUtil.parseJWT(bbbb, "bbbb");
        System.out.println(bbbb1.get("userId",Long.class));
    }

}
```

不要忘了添加，mybatisplus的配置

```
package com.mszlu.xt.sso.config;

import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@MapperScan("com.mszlu.xt.sso.dao")
public class MybatisConfig {

    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        interceptor.addInnerInterceptor(new PaginationInnerInterceptor());
        return interceptor;
    }
}
```

8. 测试
9. 前端启动 在目录下 npm run serve启动 访问 http://localhost:8080
10. nginx启动，并做配置

```
server {
        listen       80;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;
         location / {
              proxy_pass http://127.0.0.1:8080;
        }
        location /api/sso {
              proxy_pass http://127.0.0.1:8118;
            }
         location /api {
              proxy_pass http://127.0.0.1:8128;
            }
        location /lzadmin {

            proxy_pass http://127.0.0.1:8228;
        }
}
```

11. 在nginx目录 start nginx即可
12. 配置hosts文件，添加，因为微信登录要回调我们的应用，目前只有lzxtedu.com这个域名符合规则，所以需要改，以域名访问，由于配置了hosts，实际访问的还是本地的程序

```
127.0.0.1       lzxtedu.com
127.0.0.1       www.lzxtedu.com
127.0.0.1       localhost
```

13. 改为访问http://www.lzxtedu.com，这样下一步在获取用户信息后，写信息到cookie中的时候才不会出问题

14. 注意跨域问题

```
    @Override
      public void addCorsMappings(CorsRegistry registry) {

   registry.addMapping("/**").allowedOrigins("http://www.lzxtedu.com"
  );
      }
```

> *jwt讲解*

jwt 可以生成 一个加密的token，做为用户登录的令牌，当用户登录成功之后，发放给客户端。

请求需要登录的资源或者接口的时候，将token携带，后端验证token是否合法。

jwt 有三部分组成：A.B.C

A：Header，{"type":"JWT","alg":"HS256"} 固定

B：playload，存放信息，比如，用户id，过期时间等等，可以被解密，不能存放敏感信息

C：签证，A和B加上秘钥 加密而成，只要秘钥不丢失，可以认为是安全的。

jwt 验证，主要就是验证C部分 是否合法。

2.4.3 获取用户信息

> *登录完成需要获取用户信息，才能完成前端的页面展示，需要拿上token来后端进行授权，上一步，token直接写到了cookie中，也就是从cookie拿到token在后端做认证*

1. 编写拦截器，进行token认证

```
package com.mszlu.xt.sso.handler;

import com.alibaba.fastjson.JSON;
import com.mszlu.xt.common.login.UserThreadLocal;
import com.mszlu.xt.common.model.BusinessCodeEnum;
import com.mszlu.xt.common.model.CallResult;
import com.mszlu.xt.sso.service.TokenService;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@Component
@Slf4j
public class LoginInterceptor implements HandlerInterceptor {

    @Autowired
    private TokenService tokenService;

    //1. 实现登录拦截器，需要登录才能访问的接口，都会被拦截
    //2. 要从cookie中拿到对应的token
    //3. 根据token去做对应的认证，认证通过，拿到userId
    //4. 通过ThreadLocal将userId放入其中，后续的接口都可以通过threadLocal方便的拿到用户id

    //ThreadLocal ： 线程隔离的，多个线程之间，存放在threadLocal中的变量，不会被其他线程所获取和更改
    // 一个请求 就是一个线程，一个请求 controller，service，domain，dao代码
    //请求完成之后，threadLocal就会随着线程销毁
    //相比redis的好处，1. 省内存  3. redis获取信息 需要进行网络连接（开销极大）



    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        log.info("-------------------login interceptor start-----------------------");
        log.info("request uri:{}",request.getRequestURI());
        log.info("request method:{}",request.getMethod());
        log.info("-------------------login interceptor end-----------------------");

        Cookie[] cookies = request.getCookies();
        if (cookies == null){
            returnJson(response);
            return false;
        }
        String token = null;
        for (Cookie cookie : cookies) {
            String name = cookie.getName();
            if ("t_token".equals(name)){
                token = cookie.getValue();
            }
        }

        if (StringUtils.isBlank(token)){
            returnJson(response);
            return false;
        }
        Long userId = tokenService.checkToken(token);
        if (userId == null){
            returnJson(response);
            return false;
        }
        UserThreadLocal.put(userId);
        return true;
    }

    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex) throws
Exception {
        //用完threadLocal之后 其中的数据 删除，以防出现内存泄漏问题
        //threadLocal 内存泄漏  面试问题
        UserThreadLocal.remove();
    }

    private void returnJson(HttpServletResponse response){
        PrintWriter writer = null;
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json; charset=utf-8");
        try {
            writer = response.getWriter();
            CallResult callResult =
CallResult.fail(BusinessCodeEnum.NO_LOGIN.getCode(),"您的登录已失效，
请重新登录");
            writer.print(JSON.toJSONString(callResult));
        } catch (IOException e){
            e.printStackTrace();
        } finally {
            if(writer != null){
                writer.close();
            }
        }
    }
}
```

2. 写接口UserApi

```
package com.mszlu.xt.sso.api;
import com.mszlu.common.model.CallResult;
import com.mszlu.xt.model.params.user.UserParam;
import com.mszlu.xt.sso.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.ClassPathResource;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.imageio.ImageIO;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.awt.image.BufferedImage;
import java.util.Date;

@RestController
@RequestMapping("user")
public class UserApi {

    @Autowired
    private UserService userService;


    @RequestMapping("userInfo")
    public CallResult userInfo(){
        UserParam userParam = new UserParam();
        return userService.userInfo(userParam);
    }

}
```

3. UserService实现

```
package com.mszlu.xt.sso.service;

import com.mszlu.common.model.CallResult;
import com.mszlu.xt.model.params.user.UserParam;

public interface UserService {
    CallResult userInfo(UserParam userParam);
}



package com.mszlu.xt.sso.service.impl;

import com.mszlu.common.model.CallResult;
import com.mszlu.common.service.AbstractTemplateAction;
import com.mszlu.xt.model.params.user.UserParam;
import com.mszlu.xt.sso.domain.UserDomain;
import com.mszlu.xt.sso.domain.repository.UserDomainRepository;
import com.mszlu.xt.sso.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl extends AbstractService implements
UserService {

    @Autowired
    private UserDomainRepository userDomainRepository;


    @Override
    public CallResult userInfo(UserParam userParam) {
        UserDomain userDomain =
userDomainRepository.createDomain(userParam);
```

```
        return this.serviceTemplate.executeQuery(new
AbstractTemplateAction<Object>() {
            @Override
            public CallResult<Object> doAction() {
                return userDomain.userInfo();
            }
        });
    }
}
```

4. domain实现

```
 public CallResult<Object> userInfo() {
        Long userId = UserThreadLocal.get();
     //这个地方 可以考虑做一个缓存，将用户的信息临时存储起来，用于在访问过程中
的user信息的提取，后续做优化
        User user = this.userDomainRepository.findUserById(userId);
     //userModel是为了和dao层的数据库实体对象区分，model为和view交互的数据
模型
        UserModel userModel = new UserModel();
        userModel.setCity(user.getCity());
        userModel.setCountry(user.getCountry());
        userModel.setHeadImageUrl(user.getHeadImageUrl());
        userModel.setNickname(user.getNickname());
        userModel.setSex(user.getSex());
        userModel.setProvince(user.getProvince());
        userModel.setMobile(user.getMobile());
        userModel.setArea(user.getArea());
        userModel.setGrade(user.getGrade());
        userModel.setName(user.getName());
        userModel.setSchool(user.getSchool());
        return CallResult.success(userModel);
    }
```

```
package com.mszlu.xt.model;

import lombok.Data;

/**
 * @author Jarno
 */
```

```
@Data
public class UserModel {

    private String nickname;
    private Integer sex;
    private String city;
    private String province;
    private String country;
    private String headImageUrl;
    private String mobile;
    private String name;
    private String inviteUrl;
    private String school;
    private String area;
    private String grade;
//    private List<BillModel> billModelList;
}
```

5. 添加mvc的配置，加入拦截器需要拦截的路径

```
package com.mszlu.xt.sso.config;

import com.mszlu.xt.sso.handler.LoginInterceptor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Slf4j
@Configuration
public class InterceptorConfig implements WebMvcConfigurer {
    @Autowired
    private LoginInterceptor loginInterceptor;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
```

```
 registry.addMapping("/**").allowedOrigins("http://www.lzxtedu.com"
);
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(loginInterceptor)
                .addPathPatterns("/topic/*")
                .addPathPatterns("/subject/*")
                .addPathPatterns("/course/*")
                .addPathPatterns("/order/*")
                .addPathPatterns("/user/*")
                .addPathPatterns("/i/*")
                .excludePathPatterns("/course/courseList")
                .excludePathPatterns("/subject/listSubjectNew")
                .excludePathPatterns("/course/subjectInfo")
                .excludePathPatterns("/order/notify")
                .excludePathPatterns("/case/*")
                .excludePathPatterns("/wechat/*")
                .excludePathPatterns("/login/wxLoginCallBack")
                .excludePathPatterns("/i/u/*");
//                .excludePathPatterns("/course/courseList");;
        System.out.println("拦截器");
    }

}
```

6. 测试 扫码登录，跳转页面，用户状态变为已登录

2.4.4 公众号实现微信登录网站(手机端)

2.4.4.1 需求

上面实现了PC端的扫码微信登录，我们的应用程序主要的推广渠道为微信公众号，故也需要可以实现从微信公众号打开网站并进行登录，以及移动端的支付等，这里先实现登录功能。

文档地址: https://developers.weixin.qq.com/doc/offiaccount/OA_Web_Apps/Wechat_webpage_authorization.html

https://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=sandbox/login&token=1972267840&lang=zh_CN

注意使用上方的测试账号进行测试，完成流程即可，实际工作中是有正式账号的

2.4.4.2 编码

1. LoginApi添加授权登录以及微信回调的接口

```
 //测试 需要网络，回调的时候，微信方 是通过外网进行的一个访问，内网穿透
    //将本地一个端口，通过内网穿透的工具 ，暴露到外网链接
    //微信有安全配置 必须得是配置的域名才行
    //http://a4tuaki.nat.ipyingshe.com/api/sso/login/authorize
    @RequestMapping("authorize")
    public String authorize(){
        String redirectUrl = this.loginService.authorize();
        return "redirect:"+ redirectUrl;
    }

    //redirect_uri?code=CODE&state=STATE
    @GetMapping("wxGzhLoginCallBack")
    public String wxGzhLoginCallBack(HttpServletRequest request,
                                  HttpServletResponse response,
                                  String code,
                                  String state){
        //为了service层统一，所有的api层的参数处理 ，都放入loginParams中
        LoginParam loginParam = new LoginParam();
        loginParam.setCode(code);
        loginParam.setState(state);
        loginParam.setRequest(request);
        //后续 登录成功之后，要生成token，提供给前端，把token放入对应的
cookie
        // response.addCookie();
        loginParam.setResponse(response);
        CallResult callResult =
loginService.wxGzhLoginCallBack(loginParam);
        if (callResult.isSuccess()){
            return
"redirect:http://a4tuaki.nat.ipyingshe.com/course";
        }else{
            return "redirect:http://a4tuaki.nat.ipyingshe.com";
        }
    }
```

2. LoginServiceImpl中的实现：

   添加配置：

```
wx.open.config.mobile.redirectUrl=http://a4tuaki.nat.ipyingshe.com/api/sso/login/wxGzhLoginCallBack
#用于微信公众号登录和支付使用的
wx.pay.appId=wxe7f1baacb27b7d1a
wx.open.config.pay.secret=e5ee433b6225375dc468f15218f9bc26
```

```
 @Override
    public String authorize() {
        LoginDomain loginDomain =
loginDomainRepository.createDomain(new LoginParam());
        return loginDomain.buildGzhUrl();
    }

    @Override
    @Transactional
    public CallResult wxGzhLoginCallBack(LoginParam loginParam) {
        LoginDomain loginDomain =
loginDomainRepository.createDomain(loginParam);
        //带有事务的执行操作
        return this.serviceTemplate.execute(new
AbstractTemplateAction<Object>(){

            @Override
            public CallResult<Object> checkBiz() {
                //检查业务参数
                return loginDomain.checkWxLoginCallBackBiz();
            }

            @Override
            public CallResult<Object> doAction() {
                //写业务逻辑的
                return loginDomain.wxGzhLoginCallBack();
            }
        });
    }
```

3. LoginDomain添加wxGzhLoginCallBack方法实现（和网页登录一致）：

```
public String buildGzhUrl() {
        String url = this.loginDomainRepository.buildGzhUrl();
        return url;
    }

    public CallResult<Object> wxGzhLoginCallBack() {
        String code = loginParam.getCode();
        try {
            //2. 下次进行登录的时候，如果refreshToken存在，可以直接获取
accessToken，不需要用户重新授权
            String refreshToken =
loginDomainRepository.redisTemplate.opsForValue().get(RedisKey.GZH_
REFRESH_TOKEN);
            WxMpOAuth2AccessToken wxMpOAuth2AccessToken = null;
            if (refreshToken == null) {
                //1. 通过code获取到accessToken和refreshToken ,
                wxMpOAuth2AccessToken =
loginDomainRepository.wxMpServiceGzh.oauth2getAccessToken(code);
                refreshToken =
wxMpOAuth2AccessToken.getRefreshToken();
                // 需要保存refreshToken 保存在redis当中，过期时间 设置为
28天

 loginDomainRepository.redisTemplate.opsForValue().set(RedisKey.GZH
_REFRESH_TOKEN, refreshToken, 28, TimeUnit.DAYS);
            }else{
                wxMpOAuth2AccessToken =
loginDomainRepository.wxMpServiceGzh.oauth2refreshAccessToken(refre
shToken);
            }
            //3. 通过accessToken获取到了微信的用户信息（openid,
unionId）unionId在web端，公众号端，手机端唯一的
            WxMpUser wxMpUser =
loginDomainRepository.wxMpServiceGzh.oauth2getUserInfo(wxMpOAuth2Ac
cessToken, "zh_CN");
            //如果是正式账号 是没有此问题的，测试整合 unionId 不存在
            String unionId = wxMpUser.getUnionId();
            if (unionId == null){
                unionId = wxMpOAuth2AccessToken.getOpenId();
            }
            //4. 需要判断unionId在数据库中的user表中 是否存在 存在就更新最
后登录时间 不存在 注册
            User user =
this.loginDomainRepository.createUserDomain(new
UserParam()).findUserByUnionId(unionId);
            boolean isNew = false;
            if (user == null){
                //注册
                user = new User();
                Long currentTime = System.currentTimeMillis();
                user.setNickname(wxMpUser.getNickname());
                user.setHeadImageUrl(wxMpUser.getHeadImgUrl());
                user.setSex(wxMpUser.getSex());
                user.setOpenid(wxMpUser.getOpenId());
                user.setLoginType(LoginType.WX.getCode());
                user.setCountry(wxMpUser.getCountry());
                user.setCity(wxMpUser.getCity());
                user.setProvince(wxMpUser.getProvince());
                user.setRegisterTime(currentTime);
                user.setLastLoginTime(currentTime);
                user.setUnionId(unionId);
                user.setArea("");
                user.setMobile("");
                user.setGrade("");
                user.setName(wxMpUser.getNickname());
                user.setSchool("");
                this.loginDomainRepository.createUserDomain(new
UserParam()).saveUser(user);
                isNew = true;
            }

            //5. 使用jwt技术，生成token，需要把token存储起来
            String token = JwtUtil.createJWT(7 * 24 * 60 * 60 *
1000, user.getId(), secretKey);

 loginDomainRepository.redisTemplate.opsForValue().set(RedisKey.TOK
EN+token,String.valueOf(user.getId()),7,TimeUnit.DAYS);
            //6. 因为付费课程，所以账号只能在一端登录，如果用户在其他地方登
录，需要将当前的登录用户踢下线
            String oldToken =
loginDomainRepository.redisTemplate.opsForValue().get(RedisKey.LOGI
N_USER_TOKEN + user.getId());
            if (oldToken != null){
                //当前用户 之前在某一个设备登录过
                //在用户进行登录验证的时候，需要先验证token是否合法，然后去
redis查询是否存在token 不存在 代表不合法

 loginDomainRepository.redisTemplate.delete(RedisKey.TOKEN+token);
            }
 loginDomainRepository.redisTemplate.opsForValue().set(RedisKey.LOG
IN_USER_TOKEN + user.getId(),token);
            //7. 返回给前端token，存在cookie当中，下次请求的时候 从cookie
中获取token
            HttpServletResponse response =
loginParam.getResponse();
            Cookie cookie = new Cookie("t_token", token);
            cookie.setMaxAge(8*24*3600);
            cookie.setPath("/");
            response.addCookie(cookie);
            //8. 比如给用户 加积分，成就系统，任务系统
            //9. 需要记录日志，记录当前用户的登录行为  MQ+mongo进行日志记录
            //10.更新用户的最后登录时间
            if (!isNew){
                user.setLastLoginTime(System.currentTimeMillis());
                this.loginDomainRepository.createUserDomain(new
UserParam()).updateUser(user);
            }
            return CallResult.success();
        }catch (Exception e){
            e.printStackTrace();
            return
CallResult.fail(BusinessCodeEnum.LOGIN_WX_NOT_USER_INFO.getCode(),"
授权问题,无法获取用户信息");
        }
    }
```

```
public String buildGzhUrl() {
        String csrfKey = UUID.randomUUID().toString();

 redisTemplate.opsForValue().set(RedisKey.WX_STATE_KEY+csrfKey,"1",
60, TimeUnit.SECONDS);
        String url =
wxMpServiceGzh.oauth2buildAuthorizationUrl(wxOpenConfig.mobileRedir
ectUrl, WxConsts.OAuth2Scope.SNSAPI_USERINFO, csrfKey);
        return url;
    }
```

4. 测试

3. 前端

打开前端进行测试

4. 结语

学到这，具备了开发微信登录的能力，基于此，可以试着集成更多的第三方登录，比如微博，豆瓣等等的。