

第6章 移动端开发-体检预约、手机快速登录

学习目标：

- 了解体检预约流程业务
- 能够基于阿里云短信服务实现短信发送
- 掌握体检预约的实现过程【重中之重】
- 掌握预约成功页面展示的实现过程
- 了解移动端手机快速登录需求
- 掌握手机快速登录实现过程

1. 回顾体检预约流程需求

用户可以通过如下操作流程进行体检预约：

- 1、在移动端首页点击体检预约，页面跳转到套餐列表页面
- 2、在套餐列表页面点击要预约的套餐，页面跳转到套餐详情页面
- 3、在套餐详情页面点击立即预约，页面跳转到预约页面，使用页面静态化技术
- 4、在预约页面录入体检人信息，包括手机号，点击发送验证码
- 5、在预约页面录入收到的手机短信验证码，点击提交预约，完成体检预约

效果如下图：



点击【提交预约】完成预约。

1. health_web 系统管理员录入检查项，检查组管理，套餐管理，预约设置
2. health_mobile, 手机/微信用户 套餐列表->套餐详情->预约页面

2. 短信发送

【目标】

能够基于阿里云短信服务实现短信发送

【路径】

1. 短信服务介绍
2. 注册阿里云账号
3. 设置短信签名
4. 设置短信模板
5. 设置access keys
6. 短信服务API

7. 发送短信

【讲解】

2.1. 短信服务介绍

目前市面上有很多第三方提供的短信服务，这些第三方短信服务会和各个运营商（移动、联通、电信）对接，我们只需要注册成为会员并且按照提供的开发文档进行调用就可以发送短信。需要说明的是这些短信服务都是收费的服务。

本项目短信发送我们选择的是阿里云提供的短信服务。

短信服务（Short Message Service）是阿里云为用户提供的一种通信服务的能力，支持快速发送短信验证码、短信通知等。三网合一专属通道，与工信部携号转网平台实时互联。电信级运维保障，实时监控自动切换，到达率高达99%。短信服务API提供短信发送、发送状态查询、短信批量发送等能力，在短信服务控制台上添加签名、模板并通过审核之后，可以调用短信服务API完成短信发送等操作。

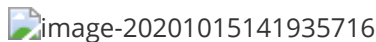
2.2. 注册阿里云账号

阿里云官网：<https://www.aliyun.com/>

点击官网首页免费注册跳转到如下注册页面：



注册后，并进入实名认证



2.3. 设置短信签名

注册成功后，点击登录按钮进行登录。登录后进入短信服务管理页面，选择国内消息菜单：

点击产品分类->云计算基础->云通信->短信服务



进入【管理控制台】



开通【短信服务】功能



【签名】：

选择签名管理



点击添加签名按钮：



目前个人用户只能申请适用场景为验证码的签名，通用需要企业认证。

2.4. 设置短信模板

在国内消息菜单页面中，点击模板管理标签页：



点击添加模板按钮：



我的模板内容是：传智健康 验证码\${code}，您正进行传智健康系统的身份验证，打死不告诉别人！

其中\${code}为动态参数，需要我们后续在代码中控制。

2.5. 设置access keys

在发送短信时需要进行身份认证，只有认证通过才能发送短信。本小节就是要设置用于发送短信时进行身份认证的key和密钥。鼠标放在页面右上角当前用户头像上，会出现下拉菜单：



点击accesskeys：



点击“开始使用子用户AccessKey”按钮，指定用户权限，而不是分配所有权限。

第一步：新建用户

输入登录名称和显示名称，点击【确认】



接收短信，防止信息泄露，输入验证码即可。



新建用户成功



第二步：授权

选择用户，添加权限



搜索“SMS”，表示短信服务，选择权限，点击“开始创建”。



第三步：创建AccessKeyID

点击创建的用户，进入到详情页面。



创建成功，其中AccessKeyID为访问短信服务时使用的ID，AccessKeySecret为密钥。



注意：需要马上保存AccessKeyID和AccessKeySecret，因为处于安全考虑，这个只显示1次，一旦退出页面就不再显示了。

点击“查看用户详情”，可以在用户详情页面下禁用刚刚创建的AccessKey：



在短信服务中，点击“国内消息设置”。可以设置每日和每月短信发送上限：



由于短信服务是收费服务，所以还需要进行充值才能发送短信：

在费用中，点击“充值”充2元就可以了



2.6. 短信服务API

点击帮助文档



找到短信服务中的“短信发送API”



将代码可以拷贝到工程中测试：

需要修改：

1：accessKeyId和accessKeySecret

```
1 final String accessKeyId = "LTAI4tZAmy7xxxxx"; // 你的accessKeyId, 参考本文档步骤2
2 final String accessKeySecret = "snM1i338WCE0hd1ws6tdbyxxxxxxx"; // 你的
   accessKeySecret, 参考本文档步骤2
```

2：手机号

```
1 request.setPhoneNumbers("1326921xxxx");
```

3：签名和模板

```
1 request.setSignName("传智健康");
2 // 必填: 短信模板-可在短信控制台中找到, 发送国际/港澳台消息时, 请使用国际/港澳台短信模版
3 request.setTemplateCode("SMS_16569xxxx");
```

4：发送的验证码及参数number（根据模板短信内容）

```
1 String params = "111111";
2 request.setTemplateParam("{\"number\": \""+params+"\""});
```

2.7. 发送短信

2.7.1. 导入maven坐标

在health_common中导入坐标

```
1 <dependency>
2   <groupId>com.aliyun</groupId>
3   <artifactId>aliyun-java-sdk-core</artifactId>
4   <version>3.3.1</version>
5 </dependency>
6 <dependency>
7   <groupId>com.aliyun</groupId>
8   <artifactId>aliyun-java-sdk-dysmsapi</artifactId>
9   <version>1.0.0</version>
10 </dependency>
```

2.7.2. 封装工具类

在health_common中添加工具类

1: 签名



2: 模板code



在health_common中，封装SMSUtils.java

传递验证码和手机号

```
1 package com.itheima.health.utils;
2
3 import com.aliyuncs.DefaultAcscClient;
4 import com.aliyuncs.IAcscClient;
5 import com.aliyuncs.dysmsapi.model.v20170525.SendSmsRequest;
6 import com.aliyuncs.dysmsapi.model.v20170525.SendSmsResponse;
7 import com.aliyuncs.exceptions.ClientException;
8 import com.aliyuncs.http.MethodType;
9 import com.aliyuncs.profile.DefaultProfile;
10 import com.aliyuncs.profile.IClientProfile;
11
12 /**
13  * 短信发送工具类
14  */
15 public class SMSUtils {
16     public static final String VALIDATE_CODE = "SMS_189616640"; // 发送短信验证码 验证码签名，改成你的
17     public static final String ORDER_NOTICE = "SMS_159771588"; // 体检预约成功通知，通知类的模板（需要通用的签名）
18     private static final String SIGN_NAEM = "黑马程序员"; // 短信的签名，属于验证码签名，改成你的
19     private static final String PARAMETER_NAME = "code"; // 短信模板内容中的参数名，改成你的 看模板内容中的${}
20     private static final String ACCESS_KEY = "LTAI4GERJj7v71F3FKjw3z2A"; // 你的AccessKey ID，改成你的
21     private static final String SECRET_KEY = "dIVZnHGdUTYbqOKMlxZ7R7jXVcnPoz"; // 你的AccessKey Secret，改成你的
22
23     public static void main(String[] args) throws ClientException {
```

```

24     SMSUtils.sendShortMessage(VALIDATE_CODE, "13652431027", "666666");
25 }
26
27 /**
28  * 发送短信
29  * @param templateCode 验证码的模板code
30  * @param phoneNumbers 接收的手机号码
31  * @param param 验证码
32  * @throws ClientException
33  */
34 public static void sendShortMessage(String templateCode, String
phoneNumbers, String param) throws ClientException{
35     // 设置超时时间-可自行调整
36     System.setProperty("sun.net.client.defaultConnectTimeout", "10000");
37     System.setProperty("sun.net.client.defaultReadTimeout", "10000");
38     // 初始化ascClient需要的几个参数
39     final String product = "Dysmsapi";// 短信API产品名称（短信产品名固定，无
需修改）
40     final String domain = "dysmsapi.aliyuncs.com";// 短信API产品域名（接口
地址固定，无需修改）
41     // 替换成你的AK
42     //final String accessKeyId = "LTAIak3CfAehK7cE";// 你的accessKeyId,参
考本文档步骤2
43     //final String accessKeySecret = "zsykwhTIFa48f8fFdU06GOKjHWHe14";//
你的accessKeySecret，参考本文档步骤2
44     // 初始化ascClient,暂时不支持多region（请勿修改）
45     IClientProfile profile = DefaultProfile.getProfile("cn-hangzhou",
ACCESS_KEY, SECRET_KEY);
46     DefaultProfile.addEndpoint("cn-hangzhou", "cn-hangzhou", product,
domain);
47     IAcsClient acsClient = new DefaultAcsClient(profile);
48     // 组装请求对象
49     SendSmsRequest request = new SendSmsRequest();
50     // 使用post提交
51     request.setMethod(MethodType.POST);
52     // 必填:待发送手机号。支持以逗号分隔的形式进行批量调用，批量上限为1000个手机号
码,批量调用相对于单条调用及时性稍有延迟,验证码类型的短信推荐使用单条调用的方式
53     request.setPhoneNumbers(phoneNumbers);
54     // 必填:短信签名-可在短信控制台中找到
55     request.setSignName(SIGN_NAME);
56     // 必填:短信模板-可在短信控制台中找到
57     request.setTemplateCode(templateCode);
58     // 可选:模板中的变量替换JSON串,如模板内容为"亲爱的${name},您的验证码为
${code}"时,此处的值为
59     // 友情提示:如果JSON中需要带换行符,请参照标准的JSON协议对换行符的要求,比如短信
内容中包含\r\n的情况在JSON中需要表示成\\r\\n,否则会导致JSON在服务端解析失败
60     //request.setTemplateParam("{\"code\":\""+param+"\"}");
61     request.setTemplateParam(String.format(
{"%s\":\"%s\"}", PARAMETER_NAME, param));
62     // 可选-上行短信扩展码(扩展码字段控制在7位或以下，无特殊需求用户请忽略此字段)
63     // request.setSmsUpExtendCode("90997");
64     // 可选:outId为提供给业务方扩展字段，最终在短信回执消息中将此值带回给调用者
65     // request.setOutId("yourOutId");
66     // 请求失败这里会抛ClientException异常
67     SendSmsResponse sendSmsResponse = acsClient.getAcsResponse(request);
68     if (sendSmsResponse.getCode() != null &&
sendSmsResponse.getCode().equals("OK")) {
69         // 请求成功

```

```

70         System.out.println("请求成功");
71     }else{
72         System.out.println(sendSmsResponse.getMessage());
73     }
74 }
75 }
76

```

2.7.3. 测试短信发送

```

1  package com.itheima.health.main;
2
3  import com.itheima.health.utils.SMSUtils;
4
5
6  public class SMSMain {
7      public static void main(String[] args) throws Exception {
8          SMSUtils.sendShortMessage(SMSUtils.VALIDATE_CODE,"你的手机号
9      码","666666");
10     }
11 }

```

测试：查看手机



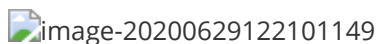
【小结】

阿里云使用步骤

1. 注册, 登录
2. 搜索 短信服务, 开通短信服务
3. 进入短信控制台, 认证
4. 申请 签名, 模版
5. 充钱(2元, 0.045元/条), 创建access keys和secretKey(开发者资质认证)
6. 添加依赖, 拷贝工具类到项目(修改签名, accessKey, secretKey, 模板code)
7. SMSUtil存入小抄
8. 测试

注意事项

工具类里面(需要改 模版code,签名, access keys, 验证码code>number)



3.体检预约【最重要】

【目标】

实现体检预约

需求：



1. 在套餐详情页面(setmeal_detail.html)点击立即预约, 页面跳转到预约页面(orderInfo.html)

2. 在预约页面(orderInfo.html)录入体检人信息, 包括手机号, 点击发送验证码
3. 在预约页面录入收到的手机短信验证码, 点击提交预约, 完成体检预约

【路径】

在/pages/orderInfo.html

套餐信息显示

orderInfo.html 加载时, 获取套餐的id, 发送请求从后台获取套餐信息, 绑定到页面中展示

发送验证码

前端: "/validateCode/send4Order.do?telephone=" + telephone

- ValidateCodeController 提供send4Order方法,接收telephone手机号码
- 从redis取出, 如果有值, 发送过了。
- 如果没值:
 - 生成验证码
 - 调用SMSUtils.发送
 - 存入redis, 加入有效时间, 过期失效

提交体检预约

提交过来的orderInfo对象,

OrderController, 添加submit的方法, 使用Map<String,String> 接收orderInfo

验证码校验 web

1. 通过手机号码获取redis中的验证码
2. 有值
 - 获取前端传过来的验证码
 - 比较redis中的验证码与前端的验证是否相同
 - 不同时, 返回验证码不正确
 - 相同, 删除key, 处理预约后续操作, 设置预约类型 (health_mobile, 写死为微信预约) 调用服务, 返回结果给页面
3. 没有值
 - 返回 重新获取验证码

预约业务实现步骤 service

1. 通过预约日期查询预约设置信息
 - 不存在, 报错, 所选日期不能预约, 请选择其它日期
 - 存在 判断是否约满 reservations>=number
 - 约满, 报错: 所选日期预约已满, 请选择其它日期
2. 通过手机号码查询会员是否存在?

- 存在才有可能重复预约

判断是否重复预约 通过member_id, orderDate, setmeal_id 查询订单表

有数据就是 重复预约, 则报错, 已经预约过了, 不能重复预约

- 不存在是不可能重复预约

添加新会员

3. 添加订单
4. 通过日期更新已预约人数

【讲解】

3.1. 前端代码



1: 在详情页面 (/pages/setmeal_detail.html) 点击体检预约

```
1 <div class="box-button">
2   <a @click="toOrderInfo()" class="order-btn">立即预约</a>
3 </div>
```

2: toOrderInfo()方法:

```
1 toOrderInfo(){
2   window.location.href = "orderInfo.html?id=" + id;
3 }
```

3: 在预约页面 (/pages/orderInfo.html) 进行调整

3.1.1. 展示预约的套餐信息（已完成）

第一步: 从请求路径中获取当前套餐的id

```
1 <script>
2   var id = getUrlParam("id");
3 </script>
```

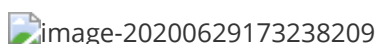
第二步: 定义模型数据setmeal, 用于套餐数据展示

```
1 <script>
2   var vue = new Vue({
3     el: '#app',
4     data: {
5       setmeal: {}, //套餐信息
6       orderInfo: {
7         setmealId: id, // 用于传递套餐id
8         sex: '1' // 用于默认显示性别男
9       } //预约信息
10    }
11  });
12 </script>
```

第三步: 显示套餐信息



第四步: 在VUE的钩子函数中发送ajax请求, 根据id查询套餐信息



3.1.2. 手机号校验

第一步：在orderInfo.html页面导入的healthmobile.js文件中已经定义了校验手机号的方法

```
1 <script src="../../plugins/healthmobile.js"></script>
```

healthmobile.js:

```
1 /**
2  * 手机号校验
3  * 1--以1为开头;
4  * 2--第二位可为3,4,5,7,8,中的任意一位;
5  * 3--最后以0-9的9个整数结尾。
6  */
7 function checkTelephone(telephone) {
8     var reg=/^[1][3,4,5,7,8][0-9]{9}$/;
9     if (!reg.test(telephone)) {
10         return false;
11     } else {
12         return true;
13     }
14 }
```

第二步：为发送验证码按钮绑定事件sendValidateCode()

```
1 <div class="input-row">
2     <label>手机号</label>
3     <input v-model="orderInfo.telephone" type="text" class="input-clear"
4     placeholder="请输入手机号">
5     <input style="font-size: x-small;" id="validateCodeButton"
6     @click="sendValidateCode()" type="button" value="发送验证码">
7 </div>
8 <div class="input-row">
9     <label>验证码</label>
10    <input v-model="orderInfo.validateCode" type="text" class="input-clear"
11    placeholder="请输入验证码">
12 </div>
```

sendValidateCode()方法:

对手机号进行校验

```
1 //发送验证码
2 sendValidateCode(){
3     //获取用户输入的手机号
4     var telephone = this.orderInfo.telephone;
5     //校验手机号输入是否正确
6     if (!checkTelephone(telephone)) {
7         this.$message.error('请输入正确的手机号');
8         return false;
9     }
10 }
```

3.1.3. 30秒倒计时效果

第一步：前面在sendValidateCode方法中进行了手机号校验，如果校验通过，需要显示30秒倒计时效果

```
1 //发送验证码
2 sendValidateCode(){
3     //获取用户输入的手机号
4     var telephone = this.orderInfo.telephone;
5     //校验手机号输入是否正确
6     if (!checkTelephone(telephone)) {
7         this.$message.error('请输入正确的手机号');
8         return false;
9     }
10    validateCodeButton = $("#validateCodeButton")[0];
11    clock = window.setInterval(doLoop, 1000); //一秒执行一次
12 },
```

第二步：其中，validateCodeButton和clock是在healthmobile.js文件中定义的属性。

doLoop是在healthmobile.js文件中定义的方法

```
1 var clock = ''; //定时器对象，用于页面30秒倒计时效果
2 var nums = 30;
3 var validateCodeButton;
4 //基于定时器实现30秒倒计时效果
5 function doLoop() {
6     validateCodeButton.disabled = true; //将按钮置为不可点击
7     nums--;
8     if (nums > 0) {
9         validateCodeButton.value = nums + '秒后重新获取';
10    } else {
11        clearInterval(clock); //清除js定时器
12        validateCodeButton.disabled = false;
13        validateCodeButton.value = '重新获取验证码';
14        nums = 30; //重置时间
15    }
16 }
```

3.1.4. 发送ajax请求

第一步：发送ajax请求

```
1 //发送验证码
2 sendValidateCode(){
3     //获取用户输入的手机号
4     var telephone = this.orderInfo.telephone;
5     //校验手机号输入是否正确
6     if (!checkTelephone(telephone)) {
7         this.$message.error('请输入正确的手机号');
8         return false;
9     }
10    validateCodeButton = $("#validateCodeButton")[0];
11    clock = window.setInterval(doLoop, 1000); //一秒执行一次
12    axios.post("/validateCode/send4Order.do?telephone=" +
13        telephone).then((res) => {
14        this.$message({
15            message: res.data.message,
```

```

15         type: res.data.flag?"success":"error"
16     });
17 });
18 },

```

在health_common中的MessageConstant中添加提示信息的常量

image-20200629175048497

第二步：创建ValidateCodeController，提供方法发送短信验证码，并将验证码保存到redis

```

1  package com.itheima.health.controller;
2
3  import com.aliyuncs.exceptions.ClientException;
4  import com.itheima.health.constant.MessageConstant;
5  import com.itheima.health.constant.RedisMessageConstant;
6  import com.itheima.health.entity.Result;
7  import com.itheima.health.utils.SMSUtils;
8  import com.itheima.health.utils.ValidateCodeUtils;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13 import redis.clients.jedis.Jedis;
14 import redis.clients.jedis.JedisPool;
15
16 /**
17  * Description: No Description
18  * User: Eric
19  */
20 @RestController
21 @RequestMapping("/validateCode")
22 public class ValidateCodeController {
23
24     @Autowired
25     private JedisPool jedisPool;
26
27     /**
28      * 发送手机验证码
29      *
30      * @param telephone
31      * @return
32      */
33     @PostMapping("/send4Order")
34     public Result send4Order(String telephone) {
35         // 生成验证码之前要检查一下是否发送过了，通过redis获取key为手机号码，看是否存
36         在
37         Jedis jedis = jedisPool.getResource();
38         String key = RedisMessageConstant.SENDTYPE_ORDER + "_" + telephone;
39         // redis中的验证码
40         String codeInRedis = jedis.get(key);
41         if (null == codeInRedis) {
42             // 不存在，没发送，生成验证码，调用SMSUtils.发送验证码，把验证码存入
43             redis(5,10,15分钟有效期)，value:验证码，key:手机号码
44             Integer code = ValidateCodeUtils.generateValidateCode(6);
45             try {

```

```

44         // 发送验证码
45         SMSUtils.sendShortMessage(SMSUtils.VALIDATE_CODE, telephone,
code + "");
46         // 存入redis, 有效时间为15分钟
47         jedis.setex(key, 15*60, code + "");
48         // 返回成功
49         return new Result(true,
MessageConstant.SEND_VALIDATECODE_SUCCESS);
50     } catch (ClientException e) {
51         e.printStackTrace();
52         // 发送失败
53         return new Result(false,
MessageConstant.SEND_VALIDATECODE_FAIL);
54     }
55 }
56 // - 存在: 验证码已经发送了, 请注意查收
57 return new Result(false, MessageConstant.SENT_VALIDATECODE);
58 }
59
60 }
61

```

3.1.5. 日历展示

页面中使用DatePicker控件来展示日历。根据需求, 最多可以提前一个月进行体检预约, 所以日历控件只展示未来一个月的日期

第一步: 引入dataPicker.js

```
1 <script src="../../plugins/datapicker/datePicker.js"></script>
```

第二步: 定义体检日期

通过样式: .picktime, 对应input组件中的class="picktime"

```

1 <div class="date">
2     <label>体检日期</label>
3     <i class="icon-date" class="picktime"></i>
4     <input v-model="orderInfo.orderDate" type="text" class="picktime"
readonly>
5 </div>

```

第三步: 定义日期控件

.picktime表示通过样式查找输入框。

```

1 <script>
2     //日期控件
3     var calendar = new datePicker();
4     calendar.init({
5         'trigger': '.picktime', /*按钮选择器, 用于触发弹出插件*/
6         'type': 'date', /*模式: date日期; datetime日期时间; time时间; ym年月; */
7         'minDate': getSpecifiedDate(new Date(), 1), /*最小日期*/
8         'maxDate': getSpecifiedDate(new Date(), 30), /*最大日期*/
9         'onSubmit': function() { /*确认时触发事件*/
10             //var theselectData = calendar.value;

```

```

11     },
12     'onClose': function() { /*取消时触发事件*/ }
13   });
14 </script>

```

其中getSpecifiedDate方法定义在healthmobile.js文件中

```

1  //获得指定日期后指定天数的日期
2  function getSpecifiedDate(date,days) {
3      date.setDate(date.getDate() + days); //获取指定天之后的日期
4      var year = date.getFullYear();
5      var month = date.getMonth() + 1;
6      var day = date.getDate();
7      return (year + "-" + month + "-" + day);
8  }

```



3.1.6. 提交预约请求（身份证校验）

为提交预约按钮绑定事件

第一步：定义“体检预约”

```

1  <div class="box-button">
2      <button @click="submitOrder()" type="button" class="btn order-btn">提交预
    约</button>
3  </div>

```

第二步：submitOrder()方法

```

1  //提交预约
2  submitOrder(){
3      //校验身份证号格式
4      if(!checkIdCard(this.orderInfo.idCard)){
5          this.$message.error('身份证号码输入错误，请重新输入');
6          return ;
7      }
8      axios.post("/order/submit.do",this.orderInfo).then((response) => {
9          if(response.data.flag){
10             //预约成功，跳转到预约成功页面
11             window.location.href="orderSuccess.html?orderId=" +
response.data.data.id;
12         }else{
13             //预约失败，提示预约失败信息
14             this.$message.error(response.data.message);
15         }
16     });
17 }

```

第三步：其中checkIdCard方法是在healthmobile.js文件中定义的，用来验证身份证的js

```

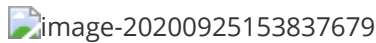
1  /**
2   * 身份证号码校验
3   * 身份证号码为15位或者18位，15位时全为数字，18位前17位为数字，最后一位是校验位，可能为
   * 数字或字符X
4   */
5  function checkIdCard(idCard){
6      var reg = /^(^d{15}$)|(^d{18}$)|(^d{17}(\d|x|x)$)/;
7      if(reg.test(idCard)){
8          return true;
9      }else{
10         return false;
11     }
12 }

```

3.2. 后台代码

3.2.0 异常处理

复制health_web的controller中的HealthExceptionHandler到health_mobile中的controller下



3.2.1. Controller

在health_mobile工程中创建OrderMobileController并提供submit方法

```

1  package com.itheima.health.controller;
2
3  import com.alibaba.dubbo.config.annotation.Reference;
4  import com.itheima.health.constant.MessageConstant;
5  import com.itheima.health.constant.RedisMessageConstant;
6  import com.itheima.health.entity.Result;
7  import com.itheima.health.pojo.Order;
8  import com.itheima.health.service.OrderService;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.util.StringUtils;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.RequestBody;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.bind.annotation.RestController;
15 import redis.clients.jedis.Jedis;
16 import redis.clients.jedis.JedisPool;
17
18 import java.util.Map;
19
20 /**
21  * Description: No Description
22  * User: Eric
23  */
24 @RestController
25 @RequestMapping("/order")
26 public class OrderMobileController {
27
28     @Autowired
29     private JedisPool jedisPool;
30

```

```

31 @Reference
32 private OrderService orderService;
33
34 /**
35  * 提交 预约
36  * @param paramMap
37  * @return
38  */
39 @PostMapping("/submit")
40 public Result submit(@RequestBody Map<String,String> paramMap){
41     // 验证码校验
42     Jedis jedis = jedisPool.getResource();
43     // 手机号码
44     String telephone = paramMap.get("telephone");
45     // 验证码的 key
46     String key = RedisMessageConstant.SENDTYPE_ORDER + ":" + telephone;
47     // redis中的验证码
48     String codeInRedis = jedis.get(key);
49     if(StringUtils.isEmpty(codeInRedis)){
50         //没值 重新发送
51         return new Result(false, "请重新获取验证码!");
52     }
53     // 前端传的验证码
54     String validateCode = paramMap.get("validateCode");
55     if(!codeInRedis.equals(validateCode)){
56         return new Result(false, MessageConstant.VALIDATECODE_ERROR);
57     }
58     // 移除redis中的验证码, 防止重复提交,
59     jedis.del(key); // 测试时可注释掉
60     // 设置预约的类型
61     paramMap.put("orderType", Order.ORDERTYPE_WEIXIN);
62     // 预约成功页面展示时需要用到id
63     Order order = orderService.submitOrder(paramMap);
64     return new Result(true, MessageConstant.ORDER_SUCCESS, order);
65 }
66 }
67

```

3.2.2. 服务接口

在health_interface工程中创建体检预约服务接口OrderService并提供预约方法

```

1 package com.itheima.health.service;
2
3 import com.itheima.health.exception.HealthException;
4 import com.itheima.health.pojo.Order;
5
6 import java.util.Map;
7
8 /**
9  * Description: No Description
10  * User: Eric
11  */
12 public interface OrderService {
13     /**
14      * 提交预约
15      * @param orderInfo

```



```

16     */
17     Order submitOrder(Map<String, String> orderInfo) throws HealthException;
18 }
19

```

3.2.3. 服务实现类

在health_service工程中创建体检预约服务实现类OrderServiceImpl并实现体检预约方法。

体检预约方法处理逻辑比较复杂，需要进行如下业务处理：

【路径】

- 1、检查用户所选择的预约日期是否已经提前进行了预约设置，如果没有设置则无法进行预约
- 2、检查用户所选择的预约日期是否已经约满，如果已经约满则无法预约
- 3、检查用户是否重复预约（同一个用户在同一天预约了同一个套餐），如果是重复预约则无法完成再次预约
- 4、检查当前用户是否为会员，如果是会员则直接完成预约，如果不是会员则自动完成注册并进行预约
- 5、预约成功，更新当日的已预约人数

实现代码如下：

```

1     /**
2      * 预约订单
3      * @param orderInfo
4      * @return
5      */
6     @Override
7     @Transactional
8     public Order submit(Map<String, String> orderInfo) throws
HealthException {
9         //1. 通过日期查询预约设置信息
10        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
11        // 预约日期 前端来
12        Date orderDate = null;
13        try {
14            orderDate = sdf.parse(orderInfo.get("orderDate"));
15        } catch (ParseException e) {
16            //e.printStackTrace();
17            throw new HealthException("日期格式不正确，请选择正确的日期");
18        }
19        OrderSetting orderSetting =
orderSettingDao.findByOrderDate(orderDate);
20        //如果不存在则报错 throw new healthException
21        if(null == orderSetting){
22            throw new HealthException("所选日期不能预约，请选择其它日期");
23        }
24        //存在：判断预约已满，满了则报错 throw new healthException
25        if(orderSetting.getReservations() >= orderSetting.getNumber()){
26            throw new HealthException("选日期预约已满，请选择其它日期");
27        }
28        //2. 判断是否重复预约
29        String telephone = orderInfo.get("telephone");
30        //通过手机号查询会员信息
31        Member member = memberDao.findByTelephone(telephone);

```

```

32 //存在才需要判断是否重复预约
33 Order order = new Order();
34 order.setOrderDate(orderDate);
35 order.setSetmealId(Integer.valueOf(orderInfo.get("setmealId")));
36 if(null != member) {
37     // 查询t_order, 条件orderDate=? and setmeal_id=?,member=?
38     order.setMemberId(member.getId());
39     //判断是否重复预约
40     List<Order> orderList = orderDao.findByCondition(order);
41     if(null != orderList && orderList.size() > 0){
42         throw new HealthException("该套餐已经预约过了, 请不要重复预约");
43     }
44 }else {
45     //不存在
46     member = new Member();
47     // name 从前端来
48     member.setName(orderInfo.get("name"));
49     // sex 从前端来
50     member.setSex(orderInfo.get("sex"));
51     // idCard 从前端来
52     member.setIdCard(orderInfo.get("idCard"));
53     // phoneNumber 从前端来
54     member.setPhoneNumber(telephone);
55     // regTime 系统时间
56     member.setRegTime(new Date());
57     // password 可以不填, 也可生成一个初始密码
58     member.setPassword("12345678");
59     // remark 自动注册
60     member.setRemark("由预约而注册上来的");
61     // 添加会员
62     memberDao.add(member);
63     order.setMemberId(member.getId());
64 }
65 //3. 可预约
66 // 预约类型
67 order.setOrderType(orderInfo.get("orderType"));
68 // 预约状态
69 order.setOrderStatus(Order.ORDERSTATUS_NO);
70 //添加t_order 预约信息
71 orderDao.add(order);
72 //4. 更新已预约人数, 更新成功则返回1, 数据没有变更则返回0
73 int affectedCount =
orderSettingDao.editReservationsByOrderDate(orderSetting);
74 if(affectedCount == 0){
75     throw new HealthException(MessageConstant.ORDER_FULL);
76 }
77 //5. 返回新添加的订单对象
78 return order;
79 }

```


导入DateUtils放置到health_common中



- 使用将一个字符串转换成日期。


3.2.4. Dao接口

3.2.4.1. OrderSettingDao.java

 image-20200925154047188

3.2.4.2. MemberDao 与 OrderDao

复制资料中的MemberDao.java 和OrderDao.java到 health_dao下


 image-20200629174441080

3.2.5. Mapper映射文件


3.2.5.1. OrderSettingDao.xml


```
1  <!--更新已预约人数-->
2  <update id="editReservationsByOrderDate" parameterType="date">
3      update t_ordersetting set reservations = reservations+1 where orderDate =
4      #{orderDate} and number>reservations
5  </update>
```

3.2.5.2. MemberDao.xml和OrderDao.xml

 image-20200629174549516

修改包名，添加health包路径，另外把所有的resultType、parameterType中的全限定名去掉包名，只留下类名即可

 image-20200629174711353

 image-20200629174803554

测试：如果出现以下异常

 img

说明t_order表的id没有设置自增长，可以设置：

 img

创建表时设置主键自增长（主键必须是整型才可以自增长）：

```
1  CREATE TABLE `t_order` (
2
3      `id` int(11) NOT NULL AUTO_INCREMENT,
4
5      `member_id` int(11) DEFAULT NULL COMMENT '会员id',
6
7      `orderDate` date DEFAULT NULL COMMENT '预约日期',
8
9      `orderType` varchar(8) DEFAULT NULL COMMENT '预约类型 电话预约/微信预约',
10
11     `orderStatus` varchar(8) DEFAULT NULL COMMENT '预约状态（是否到诊）',
12
13     `setmeal_id` int(11) DEFAULT NULL COMMENT '套餐id',
14 )
```

```

15     PRIMARY KEY (`id`),
16
17     KEY `key_member_id` (`member_id`),
18
19     KEY `key_setmeal_id` (`setmeal_id`),
20
21     CONSTRAINT `key_member_id` FOREIGN KEY (`member_id`) REFERENCES `t_member`
    (`id`),
22
23     CONSTRAINT `key_setmeal_id` FOREIGN KEY (`setmeal_id`) REFERENCES
    `t_setmeal` (`id`)
24
25 ) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;

```

修改表时设置主键自增长：

```
1 ALTER TABLE t_order CHANGE id id INT AUTO_INCREMENT;
```

修改表时删除主键自增长：

```
1 ALTER TABLE t_order CHANGE id id INT;
```

或者：选择表-->更改表-->修改表时设置主键自增长：



【小结】

一：验证码

1. 调用阿里服务, 发送成功后, 验证码存到Redis(存5, 10, 15分钟), 防止重复发送, 防止重复提交 (删除key)
2. 用户点击了预约, 需要把用户输入的验证码和redis里面存的验证码进行校验

二：预约业务

1. 判断当前的日期是否可以预约 (t_ordersetting表)
2. 判断当前的日期预约是否已满 (t_ordersetting表)
3. 判断是否 是会员 (t_member表)
 - 如果是会员, 避免重复预约 (t_order表)
 - 不是会员, 自动注册成会员, t_member 表插入一条记录 (t_member表)
4. 进行预约
 - 向t_order 表插入一条记录 (t_order表)
 - 向t_ordersetting更新reservations+1 (t_ordersetting表) 记得做防超卖 (number>reservations)
 - 事务控制, 接口异常的声明
5. 测试时要达到代码覆盖率100%

4. 预约成功页面展示

【目标】

前面已经完成了体检预约，预约成功后页面会跳转到成功提示页面（orderSuccess.html）并展示预约的相关信息（体检人、体检套餐、体检时间等）。

orderSuccess.html



跳转到orderSuccess.html，传递预约成功的订单id。

```
1 axios.post("/order/submit.do",this.orderInfo).then((response) => {
2     if(response.data.flag){
3         //预约成功，跳转到预约成功页面
4         window.location.href="orderSuccess.html?orderId=" +
response.data.data.id;
5     }else{
6         //预约失败，提示预约失败信息
7         this.$message.error(response.data.message);
8     }
9 });
```

【路径】

前台代码编写

1. 在/pages/orderSuccess.html

体检人: {{orderInfo.member}} t_member 体检套餐: {{orderInfo.setmeal}} t_setmeal 体检日期: {{orderInfo.orderDate}} t_order 预约类型: {{orderInfo.orderType}} t_order

```
1 完成需求:
2 1. 页面输出订单相关人的信息
3 2. 使用订单id，查询订单详细信息。存放到orderInfo的模型中。
```

页面加载时发送请求，提交订单的id，获取结果绑定orderInfo {member:, setmeal:, orderDate:, orderType:}

后台代码编写：

- 1.类OrderController.java
- 2.类OrderService.java
- 3.类OrderServiceImpl.java
- 4.类OrderDao.java
- 5.配置文件OrderDao.xml

```
1 select m.name member,s.name setmeal,o.orderDate, o.orderType From t_member m,
   t_setmeal s, t_order o
2 where s.id=o.setmeal_id and m.id=o.member_id and o.id=#{orderId}
```

【讲解】

4.1. 页面调整

提供orderSuccess.html页面，展示预约成功后相关信息

第一步：页面输出订单相关人的信息

```
1 <div class="info-title">
2     <span class="name">体检预约成功</span>
3 </div>
4 <div class="notice-item">
5     <div class="item-title">预约信息</div>
6     <div class="item-content">
7         <p>体检人: {{orderInfo.member}}</p>
8         <p>体检套餐: {{orderInfo.setmeal}}</p>
9         <p>体检日期: {{orderInfo.orderDate}}</p>
10        <p>预约类型: {{orderInfo.orderType}}</p>
11    </div>
12 </div>
```

第二步：使用订单id，查询订单详细信息。存放到orderInfo的变量中。

```
1 <script>
2     var vue = new Vue({
3         el: '#app',
4         data: {
5             orderInfo: {}
6         },
7         mounted() {
8             axios.get("/order/findById.do?id=" + id).then((res) => {
9                 if(res.data.flag){
10                     var orderInfo = res.data.data;
11                     // 去掉时分秒
12                     orderInfo.orderDate=orderInfo.orderDate.substring(0,10);
13                     this.orderInfo = orderInfo;
14                 }else{
15                     alert(res.data.message);
16                 }
17             });
18         }
19     });
20 </script>
```

4.2. 后台代码

4.2.1. Controller

在OrderController中提供findById方法，根据预约id查询预约相关信息

```
1 /**
2  * 预约成功展示
3  */
4 @GetMapping("/findById")
5 public Result findById(int id){
6     // 调用服务通过id查询订单信息
7     Map<String,Object> orderInfo = orderService.findOrderDetailById(id);
8     return new Result(true, MessageConstant.QUERY_ORDER_SUCCESS,orderInfo);
9 }
```

4.2.2. 服务接口

在OrderService服务接口中扩展findOrderDetailById方法

```
1  /**
2   * 通过订单id查询预约信息
3   * @param id
4   * @return
5   */
6  Map<String, Object> findOrderDetailById(int id);
```

4.2.3. 服务实现类

在OrderServiceImpl服务实现类中实现findOrderDetailById方法

```
1  /**
2   * 通过订单id查询预约信息
3   * @param id
4   * @return
5   */
6  @Override
7  public Map<String, Object> findOrderDetailById(int id) {
8      return orderDao.findById4Detail(id);
9  }
```

4.2.4. Dao接口

在OrderDao接口中扩展findById4Detail方法

```
1  Map findById4Detail(Integer id);
```

4.2.5. Mapper映射文件

在OrderDao.xml映射文件中提供SQL语句

```
1  <!--根据预约id查询预约信息，包括体检人信息、套餐信息-->
2  <select id="findById4Detail" parameterType="int" resultType="map">
3      select m.name member ,s.name setmeal,o.orderDate orderDate,o.orderType
4      orderType
5      from
6      t_order o,
7      t_member m,
8      t_setmeal s
9      where o.member_id=m.id and o.setmeal_id=s.id and o.id=#{id}
10 </select>
```

m.name member ,s.name setmeal,o.orderDate orderDate,o.orderType orderType

对应：

页面内容

```
1 <p>体检人: {{orderInfo.member}}</p>
2 <p>体检套餐: {{orderInfo.setmeal}}</p>
3 <p>体检日期: {{orderInfo.orderDate}}</p>
4 <p>预约类型: {{orderInfo.orderType}}</p>
5
```

【小结】

前端中需要什么样的数据，来源于哪些表中的字段，找出表与表之间的关系 写出sql语句

再分析前端的数据格式，后端返回前端需要的数据格式。

mybatis可以返回map数据类型，通过给列名取别名方式满足前端数据格式需求

报表步骤

1. 找出要展示的数据所在的表 t_member t_setmeal t_order
2. 找出条件所在的表 t_order.id
3. 找出数据表之间的表关系，如果没有则找中间表
4. 找出条件表之间的关系，如果没有则找中间表
5. 找出数据与条件表之间的关系，如果没有则找中间表

5. 手机快速登录

【需求分析】

手机快速登录功能，就是通过短信验证码的方式进行登录。这种方式相对于用户名密码登录方式，用户不需要记忆自己的密码，只需要通过输入手机号并获取验证码就可以完成登录，是目前比较流行的登录方式。



【目标】

实现手机快速登录

【路径】

(1) 发送验证码

1. 获得用户输入的手机号码
2. 先判断 是否已经发送过了。否则生成动态验证码(4或者6位)
3. 使用阿里云发送短信验证码
4. 把验证码存到redis(存10分钟), 防止用户重复发送验证码，登陆时验证前端输入的是否与redis的一致

(2) 登录

1. 获得用户输入的信息(Map)
2. 取出redis里面的验证码和用户输入的验证码进行校验
3. 如果校验通过
 - 判断是否是会员, 不是会员, 自动注册为会员
 - 保存用户的登录状态, 这里只存手机号码(OPEN auth 2.0 或者自己手动签发token , 我们这里可以使用Cookie存储用户_手机号码)

【讲解】

5.1. 前台代码

登录页面为/pages/login.html

5.1.1. 发送验证码

(1) 为获取验证码按钮绑定事件，并在事件对应的处理函数中校验手机号，如果手机号输入正确则显示30秒倒计时效果并发送ajax请求，发送短信验证码

```
1 <div class="input-row">
2   <label>手机号</label>
3   <div class="loginInput">
4     <input v-model="loginInfo.telephone" id='account' type="text"
placeholder="请输入手机号">
5     <input id="validateCodeButton" @click="sendValidateCode()"
type="button" style="font-size: 12px" value="获取验证码">
6   </div>
7 </div>
```

(2) 调用sendValidateCode()方法

```
1 //发送验证码
2 sendValidateCode(){
3   var telephone = this.loginInfo.telephone;
4   if (!checkTelephone(telephone)) {
5     this.$message.error('请输入正确的手机号');
6     return false;
7   }
8   validateCodeButton = $("#validateCodeButton")[0];
9   clock = window.setInterval(doLoop, 1000); //一秒执行一次
10  axios.post("/validateCode/send4Login.do?telephone=" +
telephone).then((res) => {
11    this.$message({
12      message: res.data.message,
13      type: res.data.flag?"success":"error"
14    })
15  });
16 }
```

(3) 在ValidateCodeController中提供send4Login方法，调用短信服务发送验证码并将验证码保存到redis

注意：存放到redis的可以值：手机号+002（RedisMessageConstant.SENDTYPE_LOGIN）

```
1 /**
2  * 发送登陆手机验证码
3  *
4  * @param telephone
5  * @return
6  */
7 @PostMapping("/send4Login")
8 public Result send4Login(String telephone) {
9   //- 生成验证码之前要检查一下是否发送过了，通过redis获取key为手机号码，看是否存在
```

```

10     Jedis jedis = jedisPool.getResource();
11     String key = RedisMessageConstant.SENDTYPE_LOGIN + "_" + telephone;
12     // redis中的验证码
13     String codeInRedis = jedis.get(key);
14     if (null == codeInRedis) {
15         //- 不存在，没发送，生成验证码，调用SMSUtils.发送验证码，把验证码存入
redis(5,10,15分钟有效期)，value:验证码，key:手机号码
16         Integer code = ValidateCodeUtils.generateValidateCode(6);
17         try {
18             // 发送验证码
19             SMSUtils.sendShortMessage(SMSUtils.VALIDATE_CODE, telephone,
code + "");
20             // 存入redis，有效时间为15分钟
21             jedis.setex(key, 15*60, code + "");
22             // 返回成功
23             return new Result(true,
MessageConstant.SEND_VALIDATECODE_SUCCESS);
24         } catch (ClientException e) {
25             e.printStackTrace();
26             // 发送失败
27             return new Result(false,
MessageConstant.SEND_VALIDATECODE_FAIL);
28         }
29     }
30     //- 存在：验证码已经发送了，请注意查收
31     return new Result(false, MessageConstant.SEND_VALIDATECODE);
32 }

```

5.1.2. 提交登录请求

(1) 为登录按钮绑定事件

```

1 | <div class="btn yes-btn"><a @click="login()" href="#">登录</a></div>

```

(2) 点击登录，login方法：

```

1  //登录
2  login(){
3      var telephone = this.loginInfo.telephone;
4      if (!checkTelephone(telephone)) {
5          this.$message.error('请输入正确的手机号');
6          return false;
7      }
8      axios.post("/login/check.do",this.loginInfo).then((response) => {
9          if(response.data.flag){
10             //登录成功，跳转到index.html
11             window.location.href="index.html";
12          }else{
13             //失败，提示失败信息
14             this.$message.error(response.data.message);
15          }
16      });
17  }

```

5.2. 后台代码

5.2.1. Controller

在health_mobile工程中创建LoginController并提供check方法进行登录检查，处理逻辑为：

【路径】

- 1、校验用户输入的短信验证码是否正确，如果验证码错误则登录失败
- 2、如果验证码正确，则判断当前用户是否为会员，如果不是会员则自动完成会员注册
- 3、向客户端写入Cookie，内容为用户手机号

```
1 package com.itheima.health.controller;
2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.itheima.health.constant.MessageConstant;
5 import com.itheima.health.constant.RedisMessageConstant;
6 import com.itheima.health.entity.Result;
7 import com.itheima.health.pojo.Member;
8 import com.itheima.health.service.MemberService;
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.web.bind.annotation.PostMapping;
11 import org.springframework.web.bind.annotation.RequestBody;
12 import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RestController;
14 import redis.clients.jedis.Jedis;
15 import redis.clients.jedis.JedisPool;
16
17 import javax.servlet.http.Cookie;
18 import javax.servlet.http.HttpServletResponse;
19 import java.util.Date;
20 import java.util.Map;
21
22 /**
23  * Description: No Description
24  * User: Eric
25  */
26 @RestController
27 @RequestMapping("/login")
28 public class LoginController {
29
30     @Autowired
31     private JedisPool jedisPool;
32
33     @Reference
34     private MemberService memberService;
35
36     @PostMapping("/check")
37     public Result checkMember(@RequestBody Map<String,String> loginInfo,
38                               HttpServletResponse res){
39         String telephone = loginInfo.get("telephone");
40         String validateCode = loginInfo.get("validateCode");
41         // 验证码的验证
42         String key = RedisMessageConstant.SENDTYPE_LOGIN + "_" + telephone;
43         Jedis jedis = jedisPool.getResource();
44         // 获取 redis中的验证码
```

```

44     String codeInRedis = jedis.get(key);
45     if(null == codeInRedis){
46         // 失效或没有发送
47         return new Result(false, "请点击发送验证码");
48     }
49     if(!codeInRedis.equals(validateCode)){
50         return new Result(false, "验证码不正确");
51     }
52     jedis.del(key); // 清除验证码, 已经使用过了
53
54     // 判断是否为会员
55     Member member = memberService.findByTelephone(telephone);
56     if(null == member){
57         // 会员不存在, 添加为新会员
58         member = new Member();
59         member.setRegTime(new Date());
60         member.setPhoneNumber(telephone);
61         member.setRemark("手机快速注册");
62         memberService.add(member);
63     }
64     // 跟踪记录的手机号码, 代表着会员
65     Cookie cookie = new Cookie("login_member_telephone", telephone);
66     cookie.setMaxAge(30*24*60*60); // 存1个月
67     cookie.setPath("/"); // 访问的路径 根路径下时 网站的所有路径 都会发送这个
68     res.addCookie(cookie);
69     return new Result(true, MessageConstant.LOGIN_SUCCESS);
70 }
71
72 }
73

```

5.2.2. 服务接口

在MemberService服务接口中提供findByTelephone和add方法

```

1  package com.itheima.health.service;
2
3  import com.itheima.health.pojo.Member;
4
5  /**
6   * Description: No Description
7   * User: Eric
8   */
9  public interface MemberService {
10     /**
11      * 通过手机号码查询会员信息
12      * @param telephone
13      * @return
14      */
15     Member findByTelephone(String telephone);
16
17     /**
18      * 添加会员
19      * @param member
20      */
21     void add(Member member);

```

```
22 }  
23
```

5.2.3. 服务实现类

在MemberServiceImpl服务实现类中实现findByTelephone和add方法

【路径】

1: 使用手机号查询会议

2: 新增会员

```
1 package com.itheima.health.service.impl;  
2  
3 import com.alibaba.dubbo.config.annotation.Service;  
4 import com.itheima.health.dao.MemberDao;  
5 import com.itheima.health.pojo.Member;  
6 import com.itheima.health.service.MemberService;  
7 import org.springframework.beans.factory.annotation.Autowired;  
8  
9 /**  
10  * Description: No Description  
11  * User: Eric  
12  */  
13 @Service(interfaceClass = MemberService.class)  
14 public class MemberServiceImpl implements MemberService {  
15  
16     @Autowired  
17     private MemberDao memberDao;  
18  
19     /**  
20      * 通过手机号码查询会员信息  
21      * @param telephone  
22      * @return  
23      */  
24     @Override  
25     public Member findByTelephone(String telephone) {  
26         return memberDao.findByTelephone(telephone);  
27     }  
28  
29     /**  
30      * 添加会员  
31      * @param member  
32      */  
33     @Override  
34     public void add(Member member) {  
35         memberDao.add(member);  
36     }  
37 }  
38
```

5.2.4. Dao接口（已完成）

在MemberDao接口中声明findByTelephone和add方法

```
1 public void add(Member member);
2 public Member findByTelephone(String telephone);
```

5.2.5. Mapper映射文件（已完成）

在MemberDao.xml映射文件中定义SQL语句

```
1 <!--新增会员-->
2 <insert id="add" parameterType="com.itheima.health.pojo.Member">
3     <selectKey resultType="java.lang.Integer" order="AFTER"
4     keyProperty="id">
5         SELECT LAST_INSERT_ID()
6     </selectKey>
7     insert into
8     t_member
9     (fileNumber,name,sex,idCard,phoneNumber,
10     regTime,password,email,birthday,remark)
11     values
12     ({fileNumber},{name},{sex},{idCard},{phoneNumber},
13     {regTime},{password},{email},{birthday},{remark})
14 </insert>
15 <!--根据手机号查询会员-->
16 <select id="findByTelephone" parameterType="string"
17     resultType="com.itheima.health.pojo.Member">
18     select * from t_member where phoneNumber = #{phoneNumber}
19 </select>
```

登录成功之后可以查看浏览器Cookie是否写入成功。



【小结】

1. 发送验证码 存入小抄

- 获得用户输入的手机号码
- 生成验证码
- 阿里云发送验证码
- 把验证码存到redis(5分钟)

2. 登录

- 获得用户输入的信息(Map)
- 取出redis里面存的验证码和用户输入的验证码进行比较 存入小抄
- 判断是否是会员
 - 不是会员, 自动注册为会员

3. Cookie跟踪