第11章 刷脸登录

- 理解刷脸登录的需求
- 理解刷脸登录的开发流程
- 实现刷脸登录功能

1 浅谈人工智能

1.1 人工智能的概述

人工智能（Artificial Intelligence），英文缩写为AI。它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学

人工智能是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等。人工智能从诞生以来，理论和技术日益成熟，应用领域也不断扩大，可以设想，未来人工智能带来的科技产品，将会是人类智慧的“容器”。人工智能可以对人的意识、思维的信息过程的模拟。人工智能不是人的智能，但能像人那样思考、也可能超过人的智能。

1.2 人工智能的应用领域

随着智能家电、穿戴设备、智能机器人等产物的出现和普及，人工智能技术已经进入到生活的各个领域，引发越来越多的关注。

1.3 基于人工智能的刷脸登录介绍

刷脸登录是基于人工智能、生物识别、3D传感、大数据风控技术，最新实现的登录形式。用户在无需输入用户名密码的前提下，凭借"刷脸"完成登录过程。实现刷脸登录的核心是人脸处理，在人脸处理中有两个概念：

- 人脸检测：检测图中的人脸，并为人脸标记出边框。检测出人脸后，可对人脸进行分析，获得眼、口、鼻轮廓等72个关键点定位准确识别多种人脸属性，如性别，年龄，表情等信息
- 人脸识别（对比）：通过提取人脸的特征，计算两张人脸的相似度，从而判断是否同一个人，并给出相似度评分。

作为中小型企业，可以采取世面上流行的人工智能产品快速的实现刷脸登录需求。目前比较流行人脸检测产品如下（我们的课程中使用百度云AI来完成人脸登录功能）：

- Face++
- 腾讯优图
- 科大讯飞
- 百度云AI

2 百度云AI概述

2.1 概述

百度人脸识别基于深度学习的人脸识别方案，准确识别图片中的人脸信息，提供如下功能：

- **人脸检测**：精准定位图中人脸，获得眼、口、鼻等72个关键点位置，分析性别、年龄、表情等多种人脸属性
- **人脸对比**：对比两张人脸的相似度，并给出相似度评分，从而判断是否同一个人

- **人脸搜索**：针对一张人脸照片，在指定人脸集合中搜索，找出最相似的一张脸或多张人脸，并给出相似度分值
- **活体检测**：提供离线/在线方式的活体检测能力，判断操作用户是否为真人，有效抵御照片、视频、模具等作弊攻击
- **视频流人脸采集**：设备端离线实时监测视频流中的人脸，同时支持处理静态图片或者视频流，输出人脸图片并进行图片质量控制

2.2 百度云AI的开发步骤

1. 注册账号创建应用
2. 搭建工程导入依赖
3. 人脸注册
4. 人脸识别

2.3 百度云AI的注册与认证

（1）注册百度云帐号

打开百度云平台：https://login.bce.baidu.com/reg.html?tpl=bceplat&from=portal进行账号注册

（2）激活人脸识别，并创建应用

找到产品-人工智能-人脸识别激活应用，并注册应用

应用创建完成之后，进入刚刚创建的应用获取开发所需的AppID，API Key，Secret Key。

3 百度云API的入门

3.1 搭建环境

创建工程并导入依赖：

```
<dependency>
    <groupId>com.baidu.aip</groupId>
    <artifactId>java-sdk</artifactId>
    <version>4.8.0</version>
</dependency>
```

3.2 人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

```
    //人脸注册
    @Test
    public void testFaceRegister() throws Exception {
        //传入可选参数调用接口
        HashMap<String, String> options = new HashMap<String, String>();
        options.put("quality_control", "NORMAL");
        options.put("liveness_control", "LOW");
        String imageType = "BASE64";
        String groupId = "itcast";
        String userId = "1000";
```

```
        //构造base64图片字符串
        String path = "C:\\Users\\ThinkPad\\Desktop\\ihrm\\day11\\资源\\照片\\001.png";
        byte[] bytes = Files.readAllBytes(Paths.get(path));
        String image = Base64Util.encode(bytes);

        // 人脸注册
        JSONObject res = client.addUser(image, imageType, groupId, userId, options);
        System.out.println(res.toString(2));
    }
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	String		用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	String		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	String		用户资料，长度限制256B
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

3.3 人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

```
    //人脸更新
    @Test
    public void testFaceUpdate() throws Exception {
        //传入可选参数调用接口
        HashMap<String, String> options = new HashMap<String, String>();
        options.put("quality_control", "NORMAL");
        options.put("liveness_control", "LOW");
        String imageType = "BASE64";
        String groupId = "itcast";
        String userId = "1000";

        //构造base64图片字符串
        String path = "C:\\Users\\ThinkPad\\Desktop\\ihrm\\day11\\资源\\照片\\001.png";
        byte[] bytes = Files.readAllBytes(Paths.get(path));
        String image = Base64Util.encode(bytes);

        //人脸注册
        JSONObject res = client.updateUser(image, imageType, groupId, userId, options);
        System.out.println(res.toString(2));
    }
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	String		更新指定groupid下uid对应的信息
user_id	是	String		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	String		用户资料，长度限制256B
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

3.4 人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
    //人脸检测
    @Test
    public void testFaceDetect() throws IOException {
        String path = "C:\\Users\\ThinkPad\\Desktop\\ihrm\\day11\\资源\\照片\\002.png";
        byte[] bytes = Files.readAllBytes(Paths.get(path));
        String image = Base64Util.encode(bytes);
        String imageType = "BASE64";
        HashMap<String, String> subOptions = new HashMap<String, String>();
        subOptions.put("max_face_num", "10");
        //人脸检测
        JSONObject res = client.detect(image, imageType, subOptions);
        System.out.println(res.toString(2));
    }
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_field	否	String		包括**age,beauty,expression,faceshape,gender,glasses,landmark,race,quality,facetype信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	String	1	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。
face_type	否	String		人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+beauty	否	int64	美丑打分，范围0-100，越大表示越美。当**face_fields包含beauty时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含faceshape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。

字段	必选	类型	说明
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+race	否	array	人种 **face_field包含race时返回**
++type	否	string	**yellow**: 黄种人 **white**: 白种人 **black**:黑种人 **arabs**: **阿拉伯人**
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例
+++right_eye	否	double	右眼遮挡比例
+++nose	否	double	鼻子遮挡比例
+++mouth	否	double	嘴巴遮挡比例
+++left_cheek	否	double	左脸颊遮挡比例
+++right_cheek	否	double	右脸颊遮挡比例
+++chin	否	double	下巴遮挡比例
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 **原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,...**

3.5 人脸查找

在指定人脸集合中，找到最相似的人脸

```
    //人脸搜索
    @Test
    public void testFaceSearch() throws IOException {
        String path = "D:\\223.png";
        byte[] bytes = Files.readAllBytes(Paths.get(path));
        String image = Base64Util.encode(bytes);
        String imageType = "BASE64";
        HashMap<String, String> options = new HashMap<String, String>();
        options.put("user_top_num", "1");
        //人脸搜索
        JSONObject res = client.search(image, imageType, "itcast", options);
        System.out.println(res.toString(2));
    }
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	String		从指定的group中进行查找 用逗号分隔，**上限20个**
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	String		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	String		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

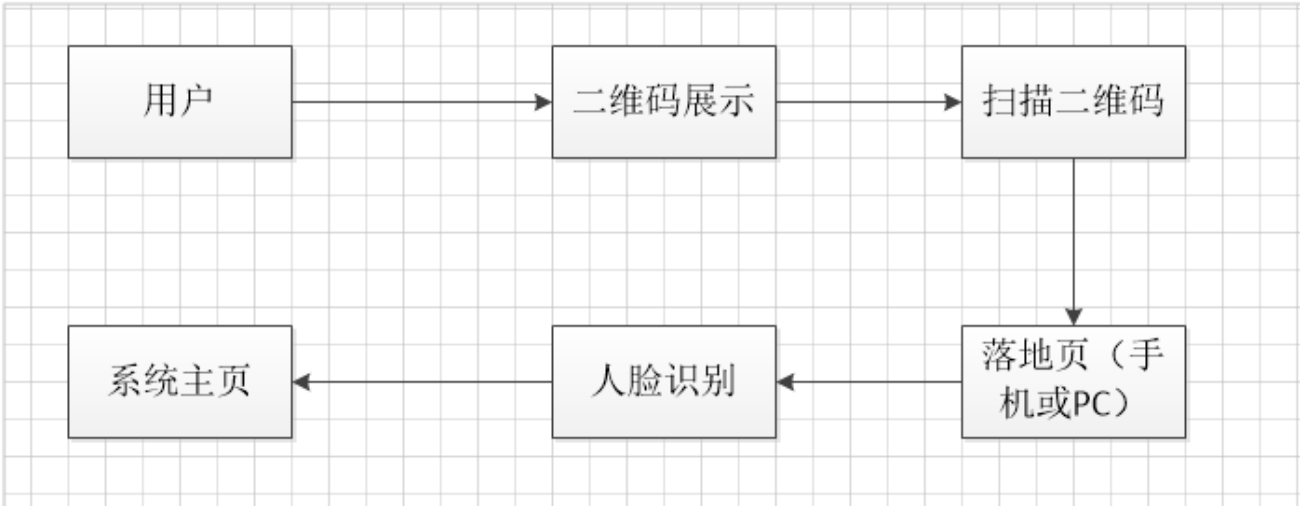
人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分，推荐阈值80分

4 刷脸登录实现

4.1 需求分析

为了用户登录的便捷，我们在系统中增加刷脸登录的功能，大致流程如下图：



- 用户在登录页面触发刷脸登录功能
- 在该页面中弹出一个二维码，此二维码是后台即时生成，包含特殊标志（但本质上是一个URL链接），后续登录流程将会使用此标志。用户对该二维码进行扫描，并在扫描端（手机或PC，注：此处不建议使用微信扫描）浏览器打开落地页。
- 打开落地页时，授权使用摄像头，并进行人脸识别，识别成功后，关闭落地页。
- 识别成功后，登录页面自动检测到成功标识，并获取相关信息，进入系统主页。
- 技术点
  - 二维码生成
  - 百度云AI
  - Redis
  - 前端摄像头调用

4.2 搭建环境

(1) 引入坐标

```
<!-- 百度云AI API-->
<dependency>
    <groupId>com.baidu.aip</groupId>
    <artifactId>java-sdk</artifactId>
    <version>4.8.0</version>
</dependency>
<!-- 二维码 -->
<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>core</artifactId>
    <version>3.2.1</version>
</dependency>
<dependency>
    <groupId>com.google.zxing</groupId>
```

```
    <artifactId>javase</artifactId>
    <version>3.2.1</version>
</dependency>
```

（2）添加配置

```
ai:
  appId: 15191935
  apiKey: cyWSHgas93Vtdmt42OwbW8pu
  secretKey: yf1GusMvvLBdOnyubfLubNyod9iEDEZW
  imageType: BASE64
  groupId: itcast
qr:
  url: https://localhost:8080/#/facelogin
```

（3）创建二维码工具类

配置二维码创建的工具类

```
@Component
public class QRCodeUtil {

    /**
     * 生成Base64 二维码
     */
    public String crateQRCode(String content) throws IOException {
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        try {
            QRCodeWriter writer = new QRCodeWriter();
            BitMatrix bitMatrix = writer.encode(content, BarcodeFormat.QR_CODE, 200,
200);
            BufferedImage bufferedImage =
MatrixToImageWriter.toBufferedImage(bitMatrix);
            ImageIO.write(bufferedImage, "png", os);
            //添加图片标识
            return new String("" +
Base64.encode(os.toByteArray()));
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            os.close();
        }
        return null;
    }

}
```

在QRCodeUtil类头添加 @Component 注解，使用时可通过 @Autowired 来自动装配。

（4）创建基本的工程结构

在系统微服务中构建基本的Controller代码

```
@RestController
@RequestMapping("/sys/faceLogin")
public class FaceLoginController {


    /**
     * 获取刷脸登录二维码
     *       返回值：QRCode对象（code，image）
     *
     */
    @RequestMapping(value = "/qrcode", method = RequestMethod.GET)
    public Result qrcode() throws Exception {
       return null;
    }

    /**
     * 检查二维码：登录页面轮询调用此方法，根据唯一标识code判断用户登录情况
     *       查询二维码扫描状态
     *           返回值：FaceLoginResult
     *               state ：-1，0，1 （userId和token）
     */
    @RequestMapping(value = "/qrcode/{code}", method = RequestMethod.GET)
    public Result qrcodeCeck(@PathVariable(name = "code") String code) throws Exception
{
        return null;
    }

    /**
     * 人脸登录：根据落地页随机拍摄的面部头像进行登录
     *           根据拍摄的图片调用百度云AI进行检索查找
     */
    @RequestMapping(value = "/{code}", method = RequestMethod.POST)
    public Result loginByFace(@PathVariable(name = "code") String code,
@RequestParam(name = "file") MultipartFile attachment) throws Exception {
        return null;
    }


    /**
     * 图像检测，判断图片中是否存在面部头像
     */
    @RequestMapping(value = "/checkFace", method = RequestMethod.POST)
    public Result checkFace(@RequestParam(name = "file") MultipartFile attachment)
throws Exception {
        return null;
    }

}
```

在系统微服务中构建基本的Service代码

```
@Service
public class FaceLoginService {

    @Value("${qr.url}")
    private String url;

    //创建二维码
    public QRCode getQRCode() throws Exception {
        return null;
    }

    //根据唯一标识，查询用户是否登录成功
    public FaceLoginResult checkQRCode(String code) {
        return null;
    }

    //扫描二维码之后，使用拍摄照片进行登录
    public String loginByFace(String code, MultipartFile attachment) throws Exception {
        return null;
    }

    //构造缓存key
    private String getCacheKey(String code) {
        return "qrcode_" + code;
    }
}
```

4.3 二维码生成

```
@Component
public class QRCodeUtil {

    /**
     * 生成Base64 二维码
     */
    public String crateQRCode(String content) throws IOException {
        System.out.println(content);
        ByteArrayOutputStream os = new ByteArrayOutputStream();
        try {
            QRCodeWriter writer = new QRCodeWriter();
            BitMatrix bitMatrix = writer.encode(content, BarcodeFormat.QR_CODE, 200,
200);
            BufferedImage bufferedImage =
MatrixToImageWriter.toBufferedImage(bitMatrix);
            ImageIO.write(bufferedImage, "png", os);
            //添加图片标识
            return new String("" +
Base64.encode(os.toByteArray()));
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            os.close();
```

```
        }
        return null;
    }

}
```

在QRCodeUtil类头添加 @Component 注解，使用时可通过 @Autowired 来自动装配。

4.4 封装API

对于百度云AI SDK我们进行一些简单的封装，便于使用时，减少代码冗余。

```
package com.ihrm.system.utils;

import com.baidu.aip.face.AipFace;
import org.json.JSONArray;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;
import java.util.HashMap;

@Component
public class BaiduAiUtil {

    @Value("${ai.appId}")
    private String APP_ID;
    @Value("${ai.apiKey}")
    private String API_KEY;
    @Value("${ai.secretKey}")
    private String SECRET_KEY;
    @Value("${ai.imageType}")
    private String IMAGE_TYPE;
    @Value("${ai.groupId}")
    private String groupId;

    private AipFace client;

    private HashMap<String, String> options = new HashMap<String, String>();

    public BaiduAiUtil() {
        options.put("quality_control", "NORMAL");
        options.put("liveness_control", "LOW");
    }

    @PostConstruct
    public void init() {
        client = new AipFace(APP_ID, API_KEY, SECRET_KEY);
    }

    /**
```

```
     *  人脸注册 ：将用户照片存入人脸库中
     */
    public Boolean faceRegister(String userId, String image) {
        // 人脸注册
        JSONObject res = client.addUser(image, IMAGE_TYPE, groupId, userId, options);
        Integer errorCode = res.getInt("error_code");
        return errorCode == 0 ? true : false;
    }

    /**
     *  人脸更新 ：更新人脸库中的用户照片
     */
    public Boolean faceUpdate(String userId, String image) {
        // 人脸更新
        JSONObject res = client.updateUser(image, IMAGE_TYPE, groupId, userId,
options);
        Integer errorCode = res.getInt("error_code");
        return errorCode == 0 ? true : false;
    }

    /**
     * 人脸检测：判断上传图片中是否具有面部头像
     */
    public Boolean faceCheck(String image) {
        JSONObject res = client.detect(image, IMAGE_TYPE, options);
        if (res.has("error_code") && res.getInt("error_code") == 0) {
            JSONObject resultObject = res.getJSONObject("result");
            Integer faceNum = resultObject.getInt("face_num");
            return faceNum == 1?true:false;
        }else{
            return false;
        }
    }

    /**
     *  人脸查找：查找人脸库中最相似的人脸并返回数据
     *          处理：用户的匹配得分（score）大于80分，即可认为是同一个用户
     */
    public String faceSearch(String image) {
        JSONObject res = client.search(image, IMAGE_TYPE, groupId, options);
        if (res.has("error_code") && res.getInt("error_code") == 0) {
            JSONObject result = res.getJSONObject("result");
            JSONArray userList = result.getJSONArray("user_list");
            if (userList.length() > 0) {
                JSONObject user = userList.getJSONObject(0);
                double score = user.getDouble("score");
                if(score > 80) {
                    return user.getString("user_id");
                }
            }
        }
        return null;
    }
```

```
}
```

- 在构造方法中，实例化client。通过client，可以调用SDK中包含的各种API。
- APP_ID, API_KEY, SECRET_KEY在文中第一段中所述位置获取，如没有正确配置，会直接导致API调用失败。
- 根据官方文档所示，我们大致创建了faceRegister()、faceUpdate()、faceCheck()、faceSearch()四个方法。
  - 人脸注册 faceRegister(groupId, userId, image)
  - groupId:用于人脸库区分人群标识，自定义即可，人脸库会根据提交的groupId，将用户分组
  - userId:人脸库中的用户标识，同组不可重复，自定义即可（通常为系统中用户的唯一标识）
  - image:Base64 用户图片
  - 人脸更新 faceUpdate(groupId, userId, image)
  - 参数解释同人脸注册
  - 该方法用于发生变化时，更新人脸信息
  - 人脸检测 faceCheck(image)
  - image:Base64 用户图片
  - 该方法用于人脸注册、人脸更新和人脸登录前使用
  - 目前采用的方案是检测出人脸数大于0即可，如需深化需求，可按需扩展
  - 人脸登录 faceSearch(image)
  - image:Base64 用户图片
  - 该方法使用的是百度云AI 人脸搜索方法，目前采用的方式是匹配度最高的结果，即要登录的用户

同样的，在BaiduAiUtil类头添加 @Component 注解，使用时可通过 @Autowired 来自动装配。在API调用后返回值处理上，进行了简单的解析，如需深化解析，可按需扩展。

4.5 功能实现

完成刷脸登录一共需要我们解决如下5个问题：

- 人脸注册/人脸更新

  在刷脸登录之前，我们首先需要对系统中的用户进行人脸注册，将相关信息提交至人脸库，才可通过人脸识别的相关接口进行刷脸登录操作。当用户相貌变更较大时，可通过人脸更新进行人脸信息更换。

- 二维码生成

  获取验证码。通过工具生成相关信息后，如特殊标志，将特殊标志写入Redis缓存，并将标记值设为”-1“，我们认定值为”-1“，即为当前标记尚未使用。调用QRCodeUtil.crateQRCode()生成二维码。

- 二维码检测

  前端获取二维码后，对二维码进行展现，并且前台启动定时器，定时检测特殊标记状态值。当状态值为“1”时，表明登录成功。

- 人脸检测

  当用户扫码进入落地页，通过落地页打开摄像头，并且定时成像。将成像图片，通过接口提交给后端进行人脸检测。

- 人脸登录

  检测成功后，即进行人脸登录，人脸登录后，改变特殊标记状态值，成功为“1”，失败为“0”。当登录成功时，进行自动登录操作，将token和userId存入到redis中。

4.5.1 后端实现

（1）人脸注册/人脸更新：在刷脸登录之前，我们首先需要对系统中的用户进行人脸注册，将相关信息提交至人脸库，才可通过人脸识别的相关接口进行刷脸登录操作。当用户相貌变更较大时，可通过人脸更新进行人脸信息更换。

```
    //人脸注册
    @RequestMapping(value = "/register/face", method = RequestMethod.POST)
    public Boolean registerFace(@RequestParam(name = "fid") String fid) throws
Exception {
        SysFile sysFile = fileService.findById(fid);
        String path = uploadPath + "/" + sysFile.getPath() + "/" +
sysFile.getUuidName();
        byte[] bytes = Files.readAllBytes(Paths.get(path));
        Boolean isSuc;
        String image = Base64Utils.encodeToString(bytes);
        isSuc = userService.checkFace(image);
        if (isSuc) {
            isSuc = baiduAiUtil.faceRegister("1", userId, image);
        }
        return isSuc;
    }

    //人脸更新
    @RequestMapping(value = "/update/face", method = RequestMethod.POST)
    public boolean updateFace(@RequestParam(name = "fid") String fid) throws Exception
{
        SysFile sysFile = fileService.findById(fid);
        String path = uploadPath + "/" + sysFile.getPath() + "/" +
sysFile.getUuidName();
        byte[] bytes = Files.readAllBytes(Paths.get(path));
        Boolean isSuc;
        String image = Base64Utils.encodeToString(bytes);
        isSuc = userService.checkFace(image);
        if (isSuc) {
            isSuc = baiduAiUtil.faceUpdate("1", userId, image);
        }
        return isSuc;
    }
```

（2）二维码生成：获取验证码。通过工具生成相关信息后，如特殊标志，将特殊标志写入Redis缓存，并将标记值设为”-1“，我们认定值为”-1“，即为当前标记尚未使用。调用QRCodeUtil.crateQRCode()生成二维码。

Controller：

```
    /**
     * 获取刷脸登录二维码
     */
    @RequestMapping(value = "/qrcode", method = RequestMethod.GET)
    public Result qrcode() throws Exception {
        return new Result(ResultCode.SUCCESS, faceLoginService.getQRCode());
    }
```

Service：

```
    public QRCode getQRCode() throws Exception {
        String code = idWorker.nextId() + "";
        FaceLoginResult result = new FaceLoginResult("-1");
        redisTemplate.boundValueOps(getCacheKey(code)).set(result, 30,
TimeUnit.MINUTES);
        String strFile = qrCodeUtil.crateQRCode(url + "?code=" + code);
        return new QRCode(code, strFile);
    }
```

（3）二维码检测：前端获取二维码后，对二维码进行展现，并且前台启动定时器，定时检测特殊标记状态值。当状态值为“1”时，表明登录成功。

Controller：

```
    /**
     * 检查二维码：登录页面轮询调用此方法，根据唯一标识code判断用户登录情况
     */
    @RequestMapping(value = "/qrcode/{code}", method = RequestMethod.GET)
    public Result qrcodeCeck(@PathVariable(name = "code") String code) throws Exception
{
        FaceLoginResult codeCheck = faceLoginService.checkQRCode(code);
        return new Result(ResultCode.SUCCESS, codeCheck);
    }
```

Service：

```
    public FaceLoginResult checkQRCode(String code) {
        String cacheKey = getCacheKey(code);
        FaceLoginResult result = (FaceLoginResult)
redisTemplate.opsForValue().get(cacheKey);
        return result;
    }
```

（4）人脸检测/人脸登录：当用户扫码进入落地页，通过落地页打开摄像头，并且定时成像。将成像图片，通过接口提交给后端进行人脸检测。

```
    /**
     * 图像检测，判断图片中是否存在面部头像
     */
    @RequestMapping(value = "/checkFace", method = RequestMethod.POST)
    public Result checkFace(@RequestParam(name = "file") MultipartFile attachment)
throws Exception {
        if (attachment == null || attachment.isEmpty()) {
            throw new CommonException();
        }
        Boolean aBoolean =
baiduAiUtil.faceCheck(Base64Utils.encodeToString(attachment.getBytes()));
        if(aBoolean) {
            return new Result(ResultCode.SUCCESS);
        }else{
```

```
            return new Result(ResultCode.FAIL);
        }
    }
```

（5）检测成功后，即进行人脸登录，人脸登录后，改变特殊标记状态值，成功为“1”，失败为“0”。当登录成功时，进行自动登录操作，将token和userId存入到redis中。

Controller：

```
    @RequestMapping(value = "/{code}", method = RequestMethod.POST)
    public Result loginByFace(@PathVariable(name = "code") String code,
@RequestParam(name = "file") MultipartFile attachment) throws Exception {
        String userId = faceLoginService.loginByFace(code, attachment);
        if(userId == null) {
            return new Result(ResultCode.FAIL);
        }else{
            //构造返回数据
            return new Result(ResultCode.SUCCESS);
        }
    }
```

Service：

```
    public String loginByFace(String code, MultipartFile attachment) throws Exception {
        String userId =
baiduAiUtil.faceSearch(Base64Utils.encodeToString(attachment.getBytes()));
        FaceLoginResult result = new FaceLoginResult("1");
        if(userId != null) {
            User user = userDao.findById(userId).get();
            if(user != null) {
                Subject subject = SecurityUtils.getSubject();
                subject.login(new UsernamePasswordToken(user.getMobile(),
user.getPassword()));
                String token = subject.getSession().getId() + "";
                result = new FaceLoginResult("0",token,userId);
            }
        }
        redisTemplate.boundValueOps(getCacheKey(code)).set(result, 30,
TimeUnit.MINUTES);
        return userId;
    }
```

4.5.2 前端实现

前端主要实现的功能是，获取二维码并展示，然后后台轮询检测刷脸登录状态，并且实现落地页相关功能（摄像头调用、定时成像、发送人脸检测和发送人脸登录请求）

（1）二维码展现

```
// 二维码
    handlecode() {
      qrcode().then(res => {
```

```
        this.param.qrcode = res.data.file
        this.centerDialogVisible = true
        this.codeCheckInfo = res.data.code
        setInterval(() => {
          if (this.states === '-1') {
          codeCheck({ code: res.data.code }).then(res => {
            this.states = res.data.state
            this.token = res.data.token
            if (this.states === '0') {
            // 登录
            this.$store
              .dispatch('LoginByCode', res.data.token)
              .then(() => {
                this.$router.push({ path: '/' })
              })
              .catch(() => {
              })
            }
            if (this.states === '1') {
              // 关闭
              this.centerDialogVisible = false
            }
          })
        }
        }, 1000 * 10)
      })
    }
```

（2）落地页调用摄像头

```
handleClick() {
      let _this = this

      if (!this.vdstate) {
        return false
      }
      if (!_this.states) {
        // 注册拍照按钮的单击事件
      let video = this.$refs['vd']
      let canvas = this.$refs['cav']
      // let form = this.$refs["myForm"];
      let context = canvas.getContext('2d')
      // 绘制画面
      context.drawImage(video, 0, 0, 200, 200)
      let base64Data = canvas.toDataURL('image/jpg')

      // 封装blob对象
      let blob = this.dataURItoBlob(base64Data, 'camera.jpg') // base64 转图片file
      let formData = new FormData()
      formData.append('file', blob)

      this.imgUrl = base64Data
```

```
        checkFace(formData).then(res => {
          if (res.data.isSuc) {
            axios({
              method: 'post',
              url: '/api/frame/facelogin/' + this.$route.query.code,
              data: formData
            })
              .then(function(response) {
                console.log(response)
                _this.states = true
                _this.canvasShow = false
                _this.tipShow = true
                // _this.$message.success('验证通过' + '!')
              })
              .catch(function(error) {
                console.log(error)
              })
          } else {
            return false
          }
        })
        }

      },
      dataURItoBlob(base64Data) {
        var byteString
        if (base64Data.split(',')[0].indexOf('base64') >= 0)
          byteString = atob(base64Data.split(',')[1])
        else byteString = unescape(base64Data.split(',')[1])
        var mimeString = base64Data
          .split(',')[0]
          .split(':')[1]
          .split(';')[0]
        var ia = new Uint8Array(byteString.length)
        for (var i = 0; i < byteString.length; i++) {
          ia[i] = byteString.charCodeAt(i)
        }
        return new Blob([ia], { type: mimeString })
      }
    }
```

4.6 总结

通过上述的步骤，可以实现一个刷脸登录的功能，其核心在于百度云AI的使用。通过合理的使用百度云AI SDK提供的相关API，我们可以很轻松的实现刷脸登录功能。刷脸登录的业务流程有很多种，我们只是实现了一种借助二维码的方式，作为抛砖引玉。更多的流程和实现方式，在此不进行赘述。