保密等级
公开

Q/DX

青岛鼎信通讯股份有限公司技术文档

Q/DX D121.100-2022

手环信息推送平台V1.0技术文档

V1.0

2022－12－08 发布　　2022－12－08

青岛鼎信通讯股份有限公司　　发布

目　　次

1 范围 2
2 引用文件 2
3 文档概述 2
4 模块说明 2
4.1 APP 端-依赖注入 3
4.1.1 自定义注解 3
4.1.2 反射注解 4
4.1.3 注解使用 4
4.1.4 控件初始化流程 5
4.2 APP 端-网络框架 6
4.2.1 泛型使用 6
4.2.2 封装 OkHttp 6
4.2.3 网络框架使用 6
4.2.4 网络请求执行流程 7
4.3 APP 端-组件间通信 7
4.3.1 订阅发布模式 7
4.3.2 封装 EventBus 8
4.3.3 订阅发布使用 8
4.3.4 组件通信流程 9
4.4 APP 端-远程升级 9
4.4.1 封装 OkHttp 9
4.4.2 FileProvider 9
4.4.3 远程升级使用 10
4.4.4 远程升级执行流程 10
4.5 APP 端-人脸检测 11
4.5.1 OpenCV 11
4.5.2 人脸检测使用 11
4.5.3 人脸检测调用流程 12
4.6 APP 端-人脸识别 12
4.6.1 人脸识别使用 12
4.6.2 人脸识别使用 13
4.7 APP 端-WebView 14
4.7.1 AgentWeb 15
4.7.2 AgentWeb 使用 15
4.7.3 H5 页面调用流程 15
4.8 APP 端-MQTT 消息接收 16
4.8.1 EMQX 16

4.8.2 EMQX 手环端使用 17
4.8.3 EMQX 手环端使用 18
4.9 服务器端-手环消息类型分类 18
4.9.1 手环消息类型分类说明 18
4.9.2 手滑消息类型分类维护 19
4.10 服务器端-手环规则配置 19
4.10.1 手环消息规则说明 19
4.10.2 手环消息规则 19
4.11 服务器端-MQTT 消息推送 20
4.11.1 EMQX 消息推送逻辑 20
4.11.2 EMQX 服务器端使用 20
4.12 服务器端-H5 页面 20
4.12.1 Thymeleaf 模板引擎 20
4.12.2 H5 功能使用 21
5 使用说明 21

前　　言

本文档定义了工程技术本部自动化部搭建运行在手环上的的系统框架，该框架分为前后端两部分，前端APP部分可分为多个模块，包括但不限于依赖注入、网络框架、事件总线、远程升级、人脸检测、人脸识别、WebView、MQTT消息接收等模块，后端服务器部分包含手环消息类型分类、手环规则配置、MQTT消息推送、H5页面等模块。

本文档适用于工程技术本部自动化部手环信息推送平台V1.0的维护指导。

本文档由青岛鼎信通讯股份有限公司工程技术本部自动化部软件室起草。

手环信息推送平台 V1.0 技术文档

1 范围

本文档定义了工程技术本部自动化部搭建，前端运行在手环端，服务器端以SpringBoot框架未基础，在linux服务器中，前端框架可分为多个模块，包括但不限于依赖注入、网络框架、事件总线、远程升级、人脸检测、人脸识别、WebView、MQTT消息接收等模块，后端服务器部分包含手环消息类型分类、手环规则配置、MQTT消息推送、H5页面等模块。

本文档适用于工程技术本部自动化部下的手环信息推送平台项目。

2 引用文件

本文档根据工作实际情况进行整理，并参考已发布文档《Android项目框架通用功能技术文档V1.0》，进行整理。

3 文档概述

文档主要包含以下几方面：

1）模块说明

2）使用原则

4 模块说明

模块之间的分布及关联关系，如图1所示：

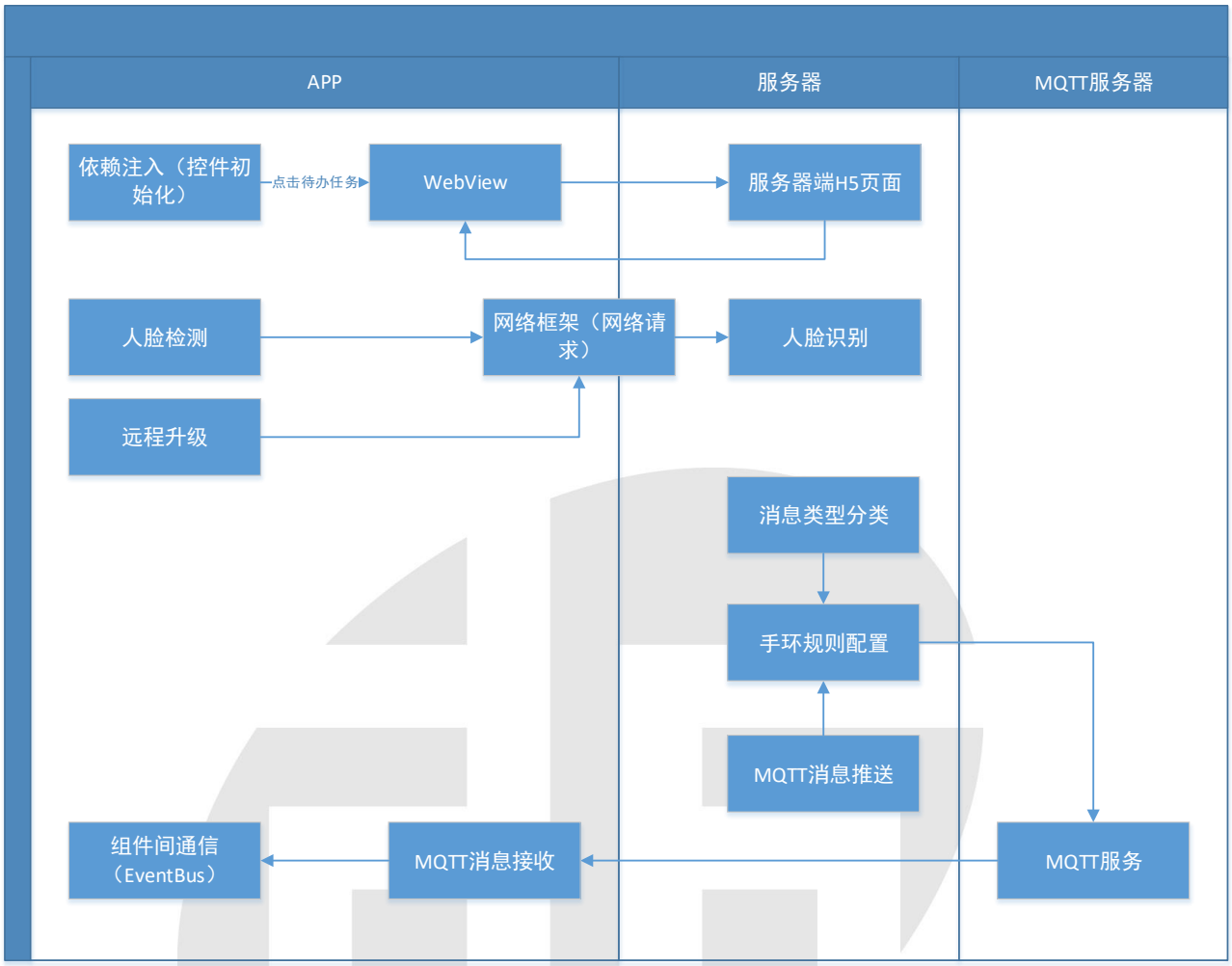

图1 模块之间的分布及关联关系

4.1 APP 端-依赖注入

通过依赖注入，规范编码，简化编程，减少代码冗余度，依赖注入主要用到了 Java 的注解及反射技术实现。

4.1.1 自定义注解

Java可自定义注解，自定义注解主要设置元注解@Target和@Retention，注解通过设置@Target作用于包、类、接口、实例、方法、注解等类型上；通过设置@Retention，设置该注解的生命周期，注解的生命周期划分为3类，分别是：RetentionPolicy.SOURCE（源码注解）、RetentionPolicy.CLASS（编译时注解）、RetentionPolicy.RUNTIME（运行时注解），如果需要在运行时去获取注解信息，则需设置为RetentionPolicy.RUNTIME。

源码注解，一般在编程时用于提示程序员，例如定义一个接口类，另外一个类继承该接口，那么该类需要实现所继承接口类的所有接口，接口上的@Override注解，是一种标记类型注解，用于提示被标注的方法实现或重写了父类方法。

编译时注解，在编译源码时起作用，比如编译时要额外向注解的某个类、方法或者实例中，进行一些预处理操作时，会用到该注解。例如Android中的@RequiresApi注解，用于消除高版本API在低版本安卓SDK中运行时报错。

运行时注解，在程序加载到内存中时仍然存在，通过这种特点，我们可对使用到该类型的注解进行譬如实例化、事件监听等操作，比如butterknife注解框架，可使用注解对控件进行实例化；

自定义注解如下：

```
@Target(ElementType.Filed)
@Retention(RetentionPolicy.RUNTIME)
public @interface ViewBind{
int value();
}
```

4.1.2 反射注解

注解添加到对应位置后，在运行时则需要对标注该注解的内容进行动态处理，因此引出Java的另外一个机制，反射，Java反射机制在运行时可获取且可调用到任意一个类或者对象的方法及属性，如图2所示：

图2 Class类内部主要信息

安卓控件的反射注解如下：

```
//第一步：通过对象获取类
Class<?> handlerClass=object.getClass();
//第二步：通过类获取Field对象
Field[] fields=handleClass.getDecalreedFields();
//第三步：遍历标准特定注解的Field
ViewBind viewBind=field.getAnnotion(ViewBind .class);
//第四步：实例化该注解的Filed
Method findViewById=handlerClass.getMethod("findViewById",int.class);
View view=(View)findViewById.invoke(object,viewBind.value());
field.set(object,view);
```

4.1.3 注解使用

依赖注入在框架中用于初始化 Activity 控件，管理控件点击事件；

控件初始化注解使用方式，如下：

```
@ViewBind(R.id.widget_name)
private Button widgetName;
```

控件点击事件使用方式，如下：

```
@OnClick({R.id.widget_name})
private void OnClick(View v){
Switch(v.getId()){
Case R.id.widget_name:
//自定义点击处理事件
break;
}
}
```

4.1.4 控件初始化流程

Activity页面的控件使用依赖注入进行控件初始化执行流程，如图3所示：

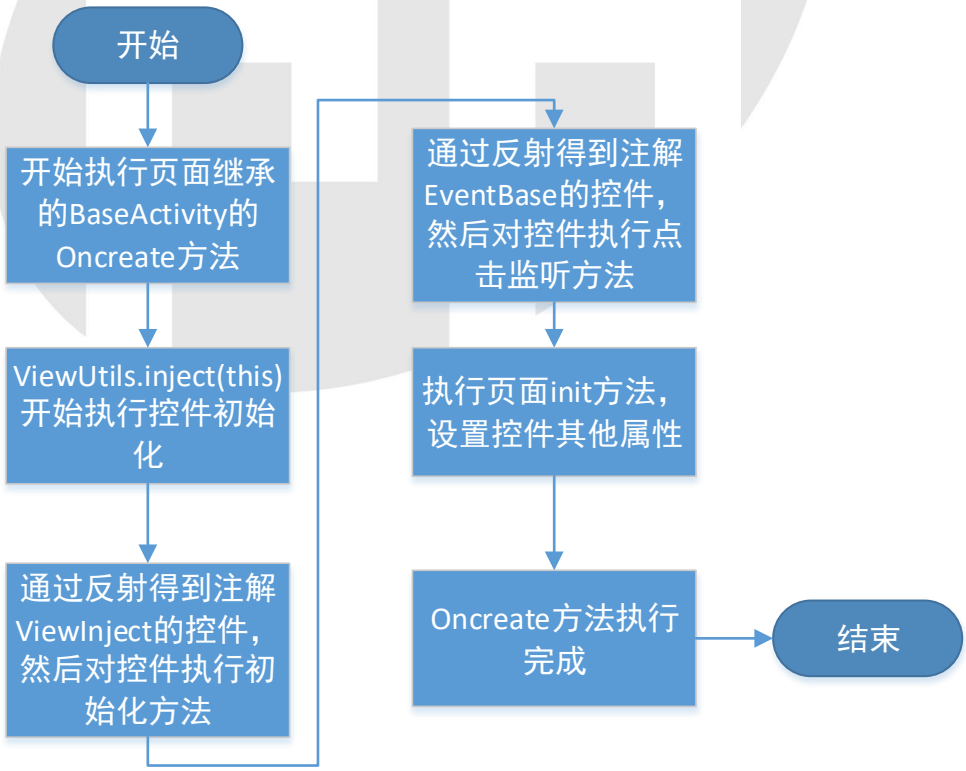

图3 控件初始化流程

4.2 APP 端-网络框架

通过封装网络请求，定义前后端数据交互格式，利用泛型，对外提供统一调用接口，简化代码。

4.2.1 泛型使用

泛型的本质时参数化类型，将被操作数据类型指定为一个参数，被操作数据类型在使用时被决定，这种被操作数据类型可以在类、接口、方法中使用，他们分别被称为泛型类、泛型接口、泛型方法。

本框架中，由于APP发送请求到服务器，服务器返回数据，由于APP端不同页面获取的数据可能不同，APP端会定义多种实体类来接收服务器端返回的数据，面对不确定的数据类型，需要使用泛型来提高代码重用率。

泛型类的定义如下：

```
public class MyClass<T extend BaseBean>{
private T tClass;
}
```

泛型接口的定义如下：

```
public interface MyClass<T>{
void show(T t);
}
```

泛型方法的定义如下：

```
public <T> T myMethod(T t){
return t;
}
```

4.2.2 封装 OkHttp

网络请求框架有很多，比如Retrofit、OkHttp、XUtils等等，其中OkHttp相比其他解决方案，安卓底层源码已经使用了OkHttp,因此其是一个相对成熟的解决方案。

封装OkHttp的过程中，OkHttp的使用，采用建造模式，通过设置请求地址及请求参数，选择同步或异步的方式，发送请求到服务器端，服务器返回数据后，Okhttp接收数据，并通过fastjson解析数据，得到数据对象。

4.2.3 网络框架使用

网络请求使用方式，如下：

```
new LoginRequest(context, new NetWorkListener<TopSysBaseUser>() {
    @Override
    public void onSuccess(TopSysBaseUser topSysBaseUser){
       //请求成功，处理事件
    }
    @Override
```

```
        public void onFailure(String errorString) {
            //请求失败，处理事件
         }
        @Override
        public void netWorkError() {
          //网络失败，处理事件
        }
    });
```

4.2.4 网络请求执行流程

当APP发起网络请求时，网络请求执行流程，如图4所示：

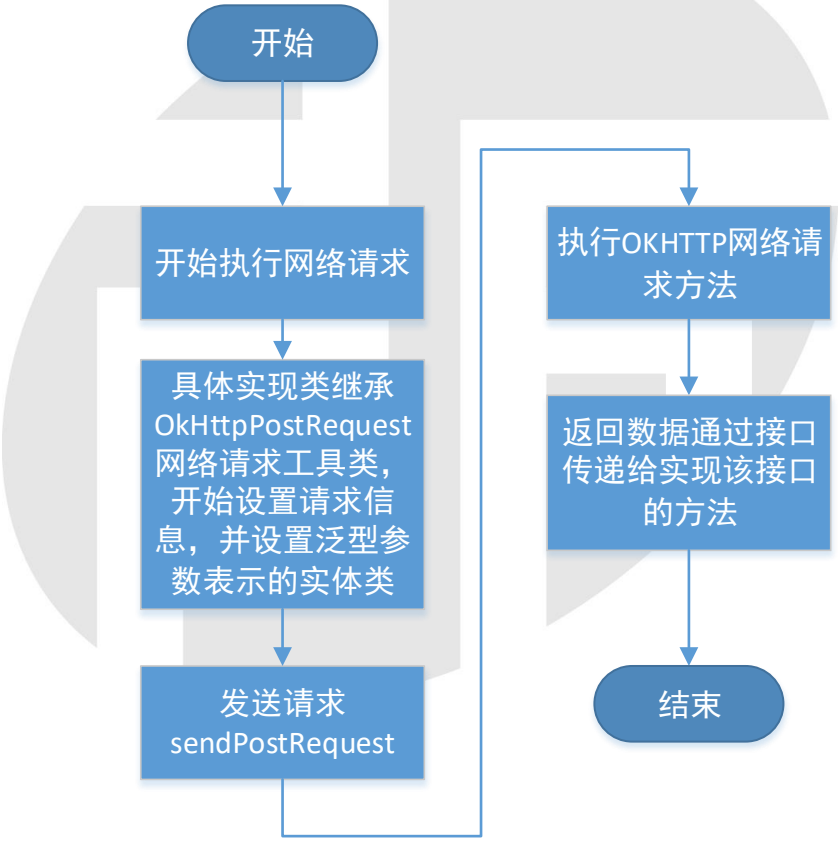

图4 网络请求执行流程

4.3 APP 端-组件间通信

由于页面之间需要数据交互，而Intent无法满足多场景需求，如：不跳转页面的情况下，向上一页面传输数据等，这就需要使用组件间通信，满足任意页面之间通信的要求。

4.3.1 订阅发布模式

订阅发布模式通过让多个观察者对象同事注册监听某个被观察对象，当被观察对象发生变化，通知所有观察者对象，然后再做相应处理，这使得观察者与被观察对象解耦，如图5所示，注册订阅某消息类型，发布者发布消息，消息调度中心向订阅者主动推送消息内容。

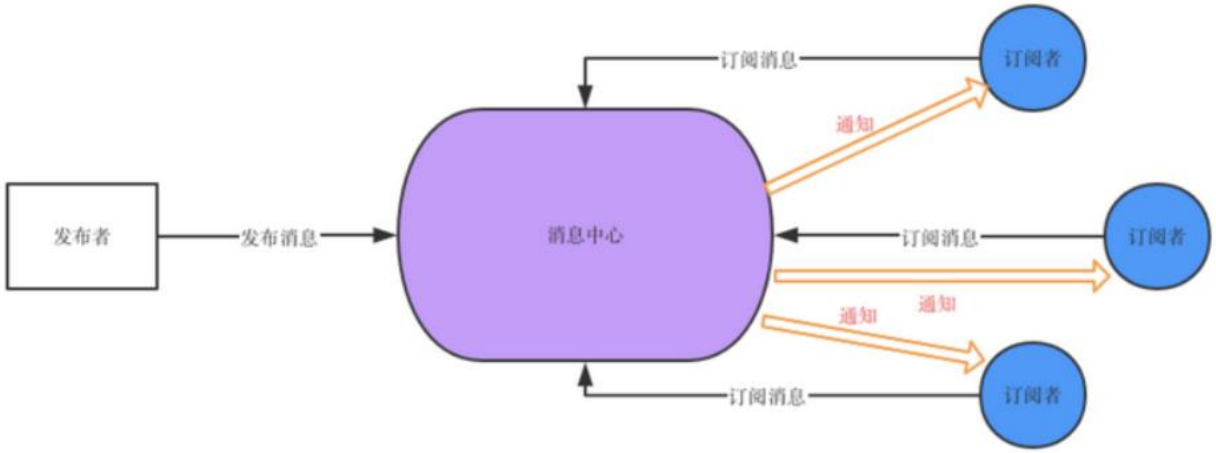

图5 消息发布订阅机制

4.3.2 封装 EventBus

EventBus利用订阅发布模式对项目进行解耦，利用少量代码，实现多组件之间的通信，如图6所示，订阅页面注册EventBus，然后在页面通过注解@Subscribe标注消息接收方法，当发送者通过EventBus发送消息时，消息接收方法会根据消息类型接收消息内容，然后对消息做进一步处理。

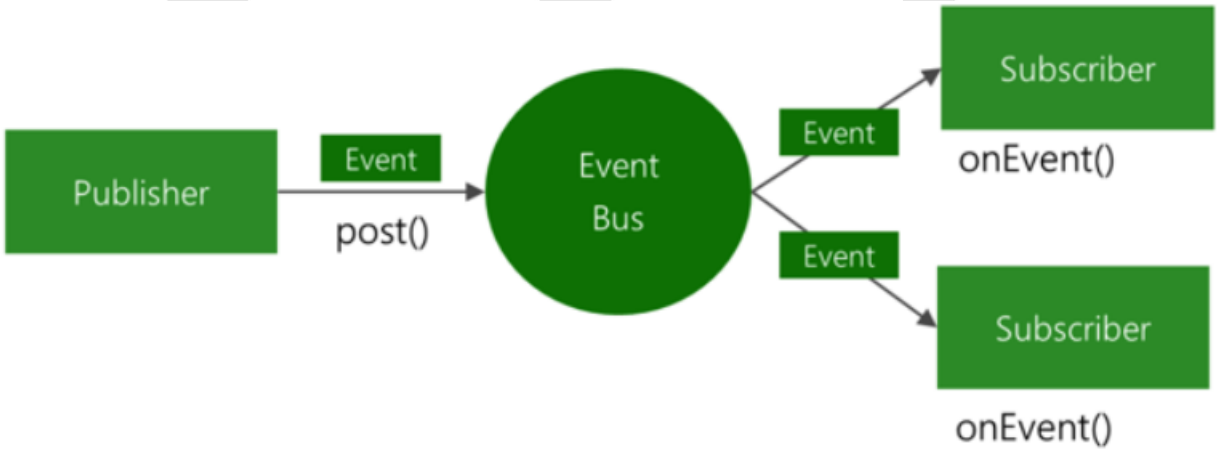

图6 EventBus工作机制

4.3.3 订阅发布使用

订阅发布使用方式，如下：

```
//第一步：订阅服务
EventBus.getDefault().register(this);
//第二步：发布消息
EventBusManage eventBusManage = new EventBusManage(Constants.FIRST);
EventBus.getDefault().post(eventBusManage);
```

```
//第三步：订阅者接收消息
@Subscribe
public void onEventMainThread(EventBusManage eventBusManage) {
//根据类型，处理消息
}
```

4.3.4 组件通信流程

各页面间传值时，可使用Event进行数据传递，如接收mqtt消息并更新待办页面流程，如下图7所示：

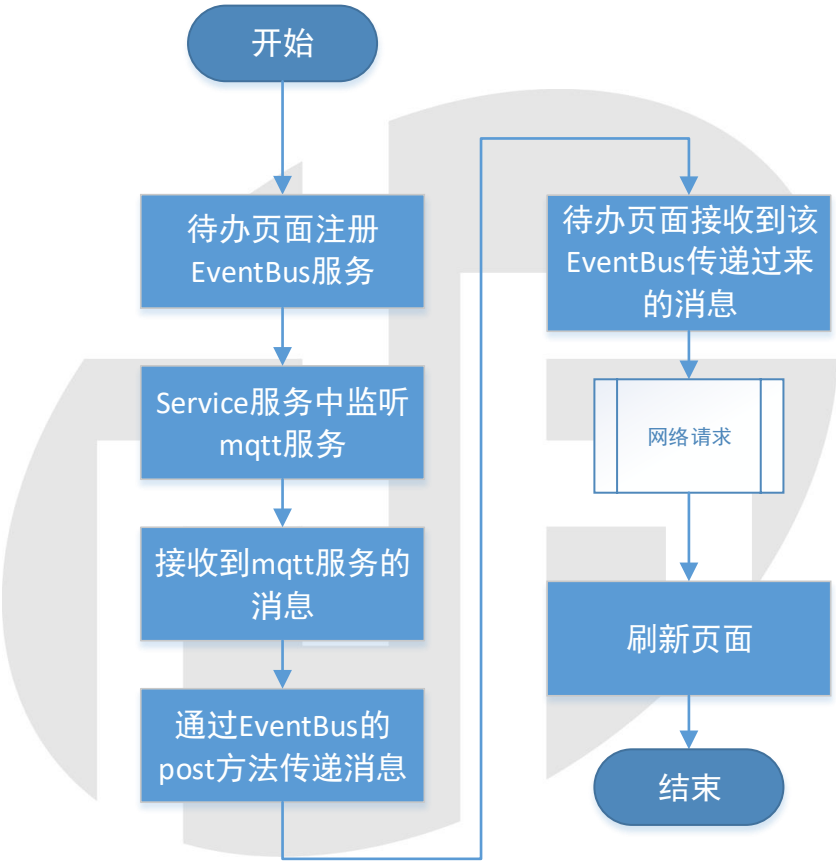

图7 网络请求执行流程

4.4 APP 端-远程升级

由于业务功能增加或更改，都需要对APP进行远程升级，因此可通过APP自动检查及手动检查更新等方式，检测服务器APP版本与现运行版本是否一致，如不一致，则需进行升级。

4.4.1 封装 OkHttp

由于需要从远程服务器下载文件到APP运行设备中，OkHttp提供文件下载功能，因此封装OkHttp即可，通过OkHttpClient类发起下载请求，然后通过OnResponse回调方法处理，将数据写入到文件输出流，数据传输完成后，存储文件到本地。

4.4.2 FileProvider

Android 7.0之后，Android官方提高了程序访问私有目录的安全性，如果要存取文件，需使用FileProvider设置可访问的目录。

FileProvider使用方式，如下：

第一步：xml文件中设置provider

第二步：resource文件中设置可共享的目录

4.4.3 远程升级使用

本模块根据开闭原则，封装成升级模块，开发者只需调用即可。

远程升级调用方式，如下：

第一步：获取远程服务器APP版本，使用网络框架的调用方法即可；

第二步：对比当前版本；

第三步：弹出升级对话框进行升级；

第四步：下载APP

```
AppUtils.downLoadApp(new DownLoadListener() {
        @Override
        public void onSuccess() {
            //APP下载完成，处理事件
        }
        @Override
        public void onDown(String progress) {
            //APP下载中，处理事件
        }
        @Override
        public void onFailure(String errorString) {
            //APP下载失败，处理事件
    }
    }, context);
```

4.4.4 远程升级执行流程

远程升级的详细执行流程，如图8所示：

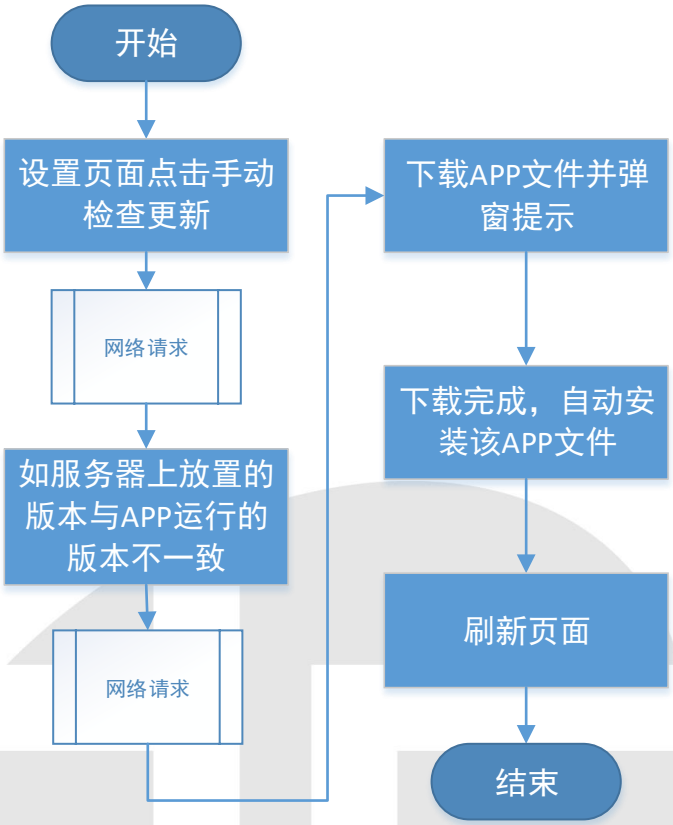

图8 远程升级执行流程

4.5 APP 端-人脸检测

移动端人脸检测是目前非常常见的图像处理方法，比如刷脸登录、刷脸支付等等，各种应用场景层出不穷，因此本框架也开发封装了人脸检测功能。本模块利用JNI及Opencv图像检测技术实现摄像头图像数据的人脸检测功能。Opencv通过训练人脸数据得到Adaboost级联分类器，Android通过jni调用分类器，从而计算当前图像是否有人脸数据。

4.5.1 OpenCV

OpenCV是一个跨平台的计算机视觉与机器学习软件库，由于其采用C和C++语言，其轻量且高效，并提供多语言的对外接口。

OpenCV提供Android端的版本，因此可下载相关文件，将其集成到Android项目中，然后通过Java的JNI技术，调用相关函数即可。

集成时，需将相关文件复制到项目目录下，通过CmakeLists.txt设置OpenCV文件依赖，完成后，新建cpp文件，并引入相关包即可。

安卓摄像头获取图片，传入cpp文件中，cpp文件通过一系列处理，包含灰度化等，将图像数据传入detectMultiScale，得到人脸位置，然后将人脸位置数据传回Android前端，Android前端通过绘图方法，将人脸范围标注到界面上。

4.5.2 人脸检测使用

人脸检测调用方式，如下：

```
int  width = bitmap.getWidth();
```

```
int  height = bitmap.getHeight();
int[] pix = new int[width * height];
bitmap.getPixels(pix, 0, width, 0, 0, width, height);
//通过下面一行代码调用so文件，返回人脸检测结果
int[] resultInt = FaceJNI.faceDetect(width, height, pix);
```

4.5.3 人脸检测调用流程

人脸检测调用流程，如图9所示：

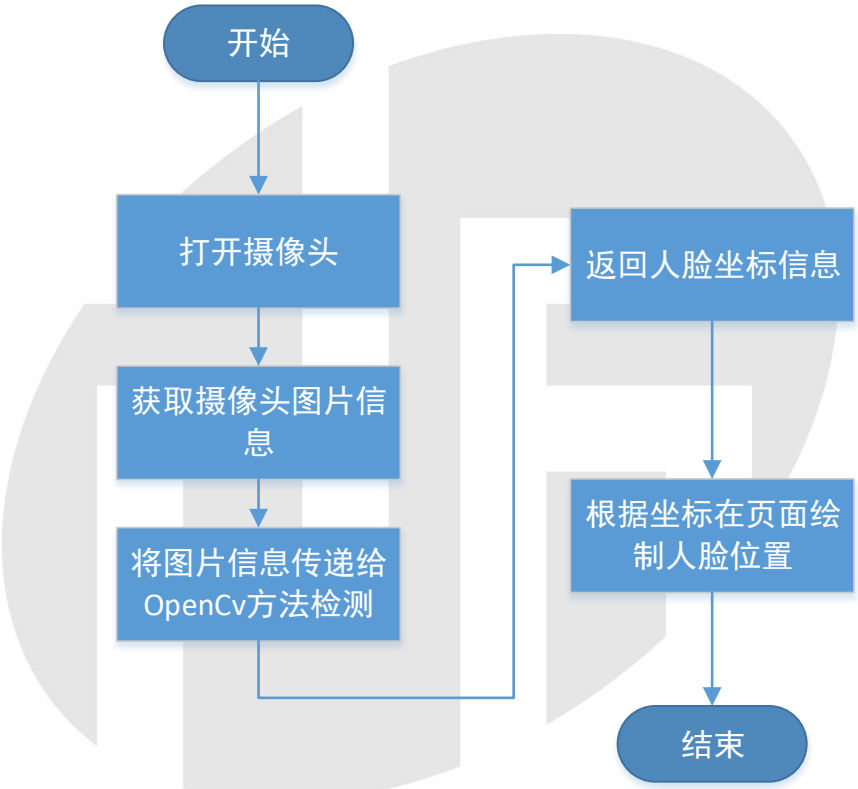

图9 人脸检测调用流程

4.6 APP 端-人脸识别

人脸识别是将已注册的人脸，与传入的待识别人脸进行特征值对比，通过对比结果得到是否为同一人，OpenCV提供多种人脸识别方法，包含FisherFaceRecognizer(基于LDA降维)，LBPHFaceRecognizer(基于LBPH特征)，其中LBPHFaceRecognizer可以自行训练更新模型，但是要收集大量样本训练，FisherFaceRecognizer对人脸角度变化十分敏感，因此采用基于深度学习的人脸识别模型FaceRecognizerSF。

4.6.1 人脸识别使用

人脸识别模块采用通过深度学习训练的人脸识别文件，人脸识别因需对图像进行计算对比，而且放在服务器端方便管理，因此采用APP端经过人脸检测，将检测到人脸的图像数据通过网络发送到服务器端，服务器端接收到图像数据后，对图像重新进行处理，处理后，检测后人脸对齐然后提取得到人脸特征数据，

然后分别与数据库中的人脸特征数据进行匹配，得到大于阈值，且相似度最高的人脸数据，返回该人脸对应的用户信息，APP得到用户信息后，对事件进行处理，人脸识别完成。

```
//第一步:人脸检测
//第二步：通过网络框架，上传人脸图像，然后服务器返回人脸识别结果
new OkHttpUploadFileRequest(NetworkManage.FACE_RECOGNITION,TopSysBaseUser.class,
BASE_PATH + "/" + "face.jpg", new UploadFileListener<TopSysBaseUser>() {
    @Override
    public void onSuccess(TopSysBaseUser topSysBaseUser) {
        //得到人脸识别结果，处理事件
    }
    @Override
    public void onFailure(String errorString) {
        //失败，处理事件
    }
    },context);
```

4.6.2 人脸识别使用

人脸识别过程，如图10所示：

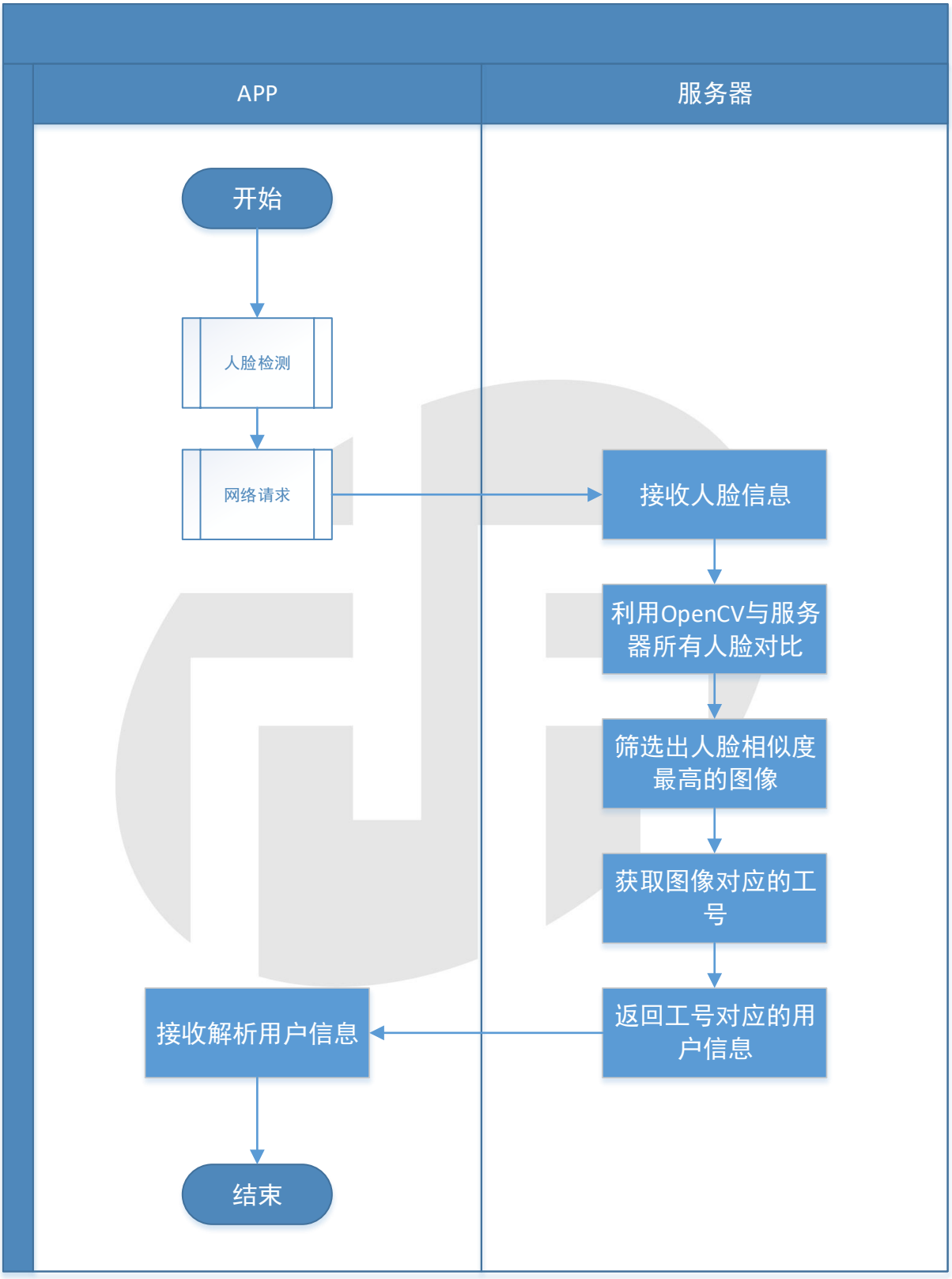

图10 人脸识别流程

4.7 APP 端-WebView

为满足多样化需求，业务页面往往需要动态变化，而Android原生界面每次改动，都需要对程序进行升级，比较繁琐，而通过前端调用后端的H5页面，则完美解决了这一问题，通过后台更新html页面中的代码，前端通过webview调用显示该页面即可，但是html与Android原生不同，Android原生可直接调用sdk提供的方法，html如需要调用Android方法，则需要html与Android通过指定格式才能进行通信交互。

4.7.1 AgentWeb

WebView是Android提供的用于展示web页面的控件，提供与JS交互的方法。首先需设置setJavaScriptEnabled（true）和，通过WebView设置与JS内具体的方法名，即可通过Android调用JS方法；通过在Android方法加上@JavascriptInterface，即可通过JS调用该方法。

由于WebView功能较单一，许多第三方进行了功能封装，其中的第三方框架AgentWeb，支持文件上传、下载、简化JS通信、带有进度条、加强了Web安全，因此本模块选择AgentWeb作为JS与Android通信的工具。

4.7.2 AgentWeb 使用

```
agentWeb=AgentWeb.with(this).setAgentWebParent(llParent,new
LinearLayout.LayoutParams(-1, -1)).useDefaultIndicator().createAgentWeb().ready()
.go("file:///android_asset/javascript.html");
//设置允许Android调用JS
agentWeb.getAgentWebSettings().getWebSettings().setJavaScriptEnabled(true);
//将图片调整到适合webview的大小
agentWeb.getAgentWebSettings().getWebSettings().setUseWideViewPort(true);
// 缩放至屏幕的大小
agentWeb.getAgentWebSettings().getWebSettings().setLoadWithOverviewMode(true);
//将带有该标志的JS事件传递给Android
agentWeb.getJsInterfaceHolder().addJavaObject("androidWebView", this);
```

4.7.3 H5 页面调用流程

查看待办任务详情程序处理流程，如图11所示：

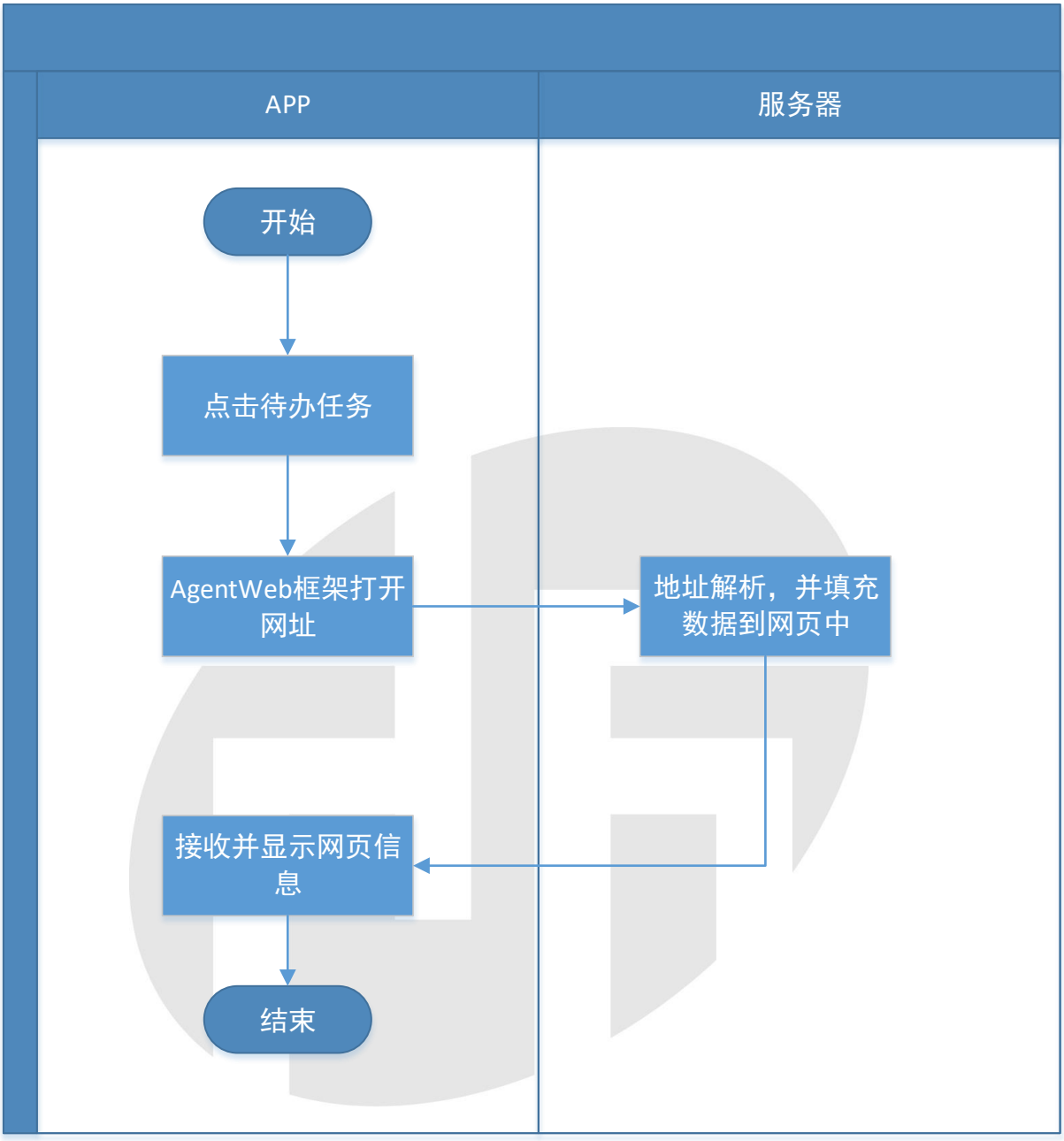

图11 查看待办任务详情程序处理流程

4.8 APP 端-MQTT 消息接收

为实现消息推送与接收，故使用消息中间件，在APP端集成MQTT模块，用于消息接收。

4.8.1 EMQX

EMQX是基于Erlang语言开发的MQTT消息服务器，用于支持各种接入标准MQTT协议的设备，实现设备端与服务器端的消息传递，从而实现数据交互。

EMQX服务器安装后，服务器端连接MEQX服务器后，可通过接口发送消息到APP设备端，APP设备端通过连接EMQX服务器，通过EMQX提供的方法接收消息，其工作原理如图12所示：

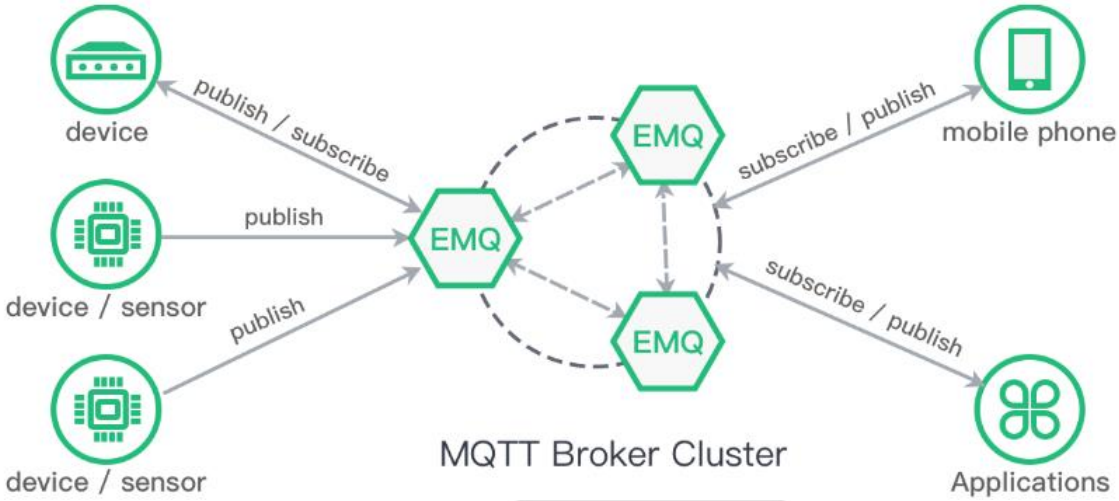

图12 EMQX工作原理

4.8.2 EMQX 手环端使用

EMQX手环端连接调用方式，如下：

```
private void doClientConnection() {
    try {
        disconnect();
        if (!MqttAndroidClient.isConnected()) {
            MqttAndroidClient.connect(options, null, iMqttActionListener);
        }
    } catch (Exception e) {
        Log.i(TAG,"mqtt连接异常" + e.getMessage());
    }
}
```

EMQX手环端接收消息的方式，如下：

```
private MqttCallbackExtended mqttCallbackExtended = new MqttCallbackExtended() {
  @Override
  public void connectionLost(Throwable cause){
            Log.i(TAG, "连接丢失 ");
            if(PreferenceUtils.getBoolean(PreferenceManage.IS_ONLINE,false))
            doClientConnection();
  }
  @Override
  public void messageArrived(String topic, MqttMessage message){
```

```
                // subscribe后得到的消息会执行到这里面
                _topic = topic;
                _qos = message.getQos() + "";
                _msg = new String(message.getPayload());
        }
        @Override
        public void deliveryComplete(IMqttDeliveryToken token) {
        }
        @Override
        public void connectComplete(boolean reconnect, String serverURI) {
                Log.i(TAG,"mqtt连接完成");
            }
        };
```

4.8.3 EMQX 手环端使用

EMQX消息发布接收流程，如图13所示：

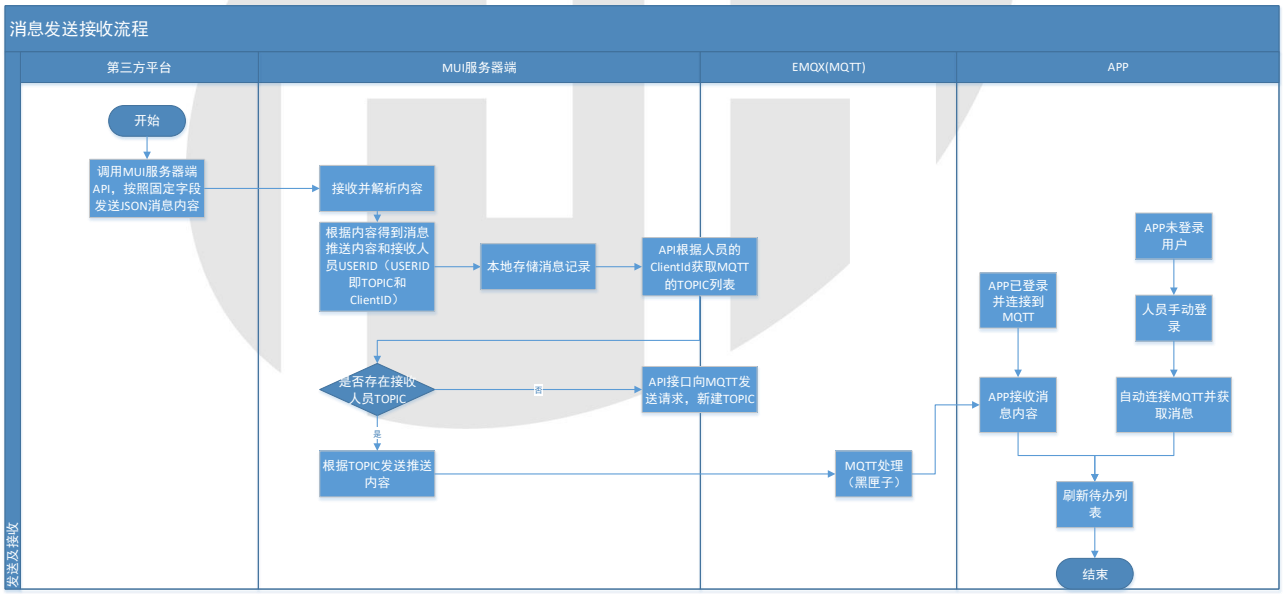

图13 EMQX消息发布接收流程

4.9 服务器端-手环消息类型分类

手环消息类型分类，用于区分当前消息类型，并通过类型设置消息展示内容。

4.9.1 手环消息类型分类说明

在FMS维护消息类型分类，可添加、编辑消息类型、消息模板、消息等级等信息；描述如下：

（1）消息类型：用于区分消息来源。

（2）消息模板：用于展示消息内容。

4.9.2 手滑消息类型分类维护

在FMS维护消息类型分类，如下图14所示：

图14 手环消息类型分类

4.10 服务器端-手环规则配置

手环消息规则配置，用于设置具体的消息推送规则，通过编辑消息类型、消息位置、关联消息接收人员，设置具体的消息推送规则。

4.10.1 手环消息规则说明

在FMS维护手环消息规则，可添加、编辑消息规则：

（1）消息消息名称：用于说明该规则用于推送什么信息。

（2）手环接收消息类型：用于设置本条消息规则接收的具体消息类型。

（3）手环接收消息位置：用于设置本条消息规则接收的指定位置的消息。

（4）手环消息接收人员：用于设置本条消息规则接收的相关人员。

4.10.2 手环消息规则

在FMS维护手环消息规则，如下图15所示：

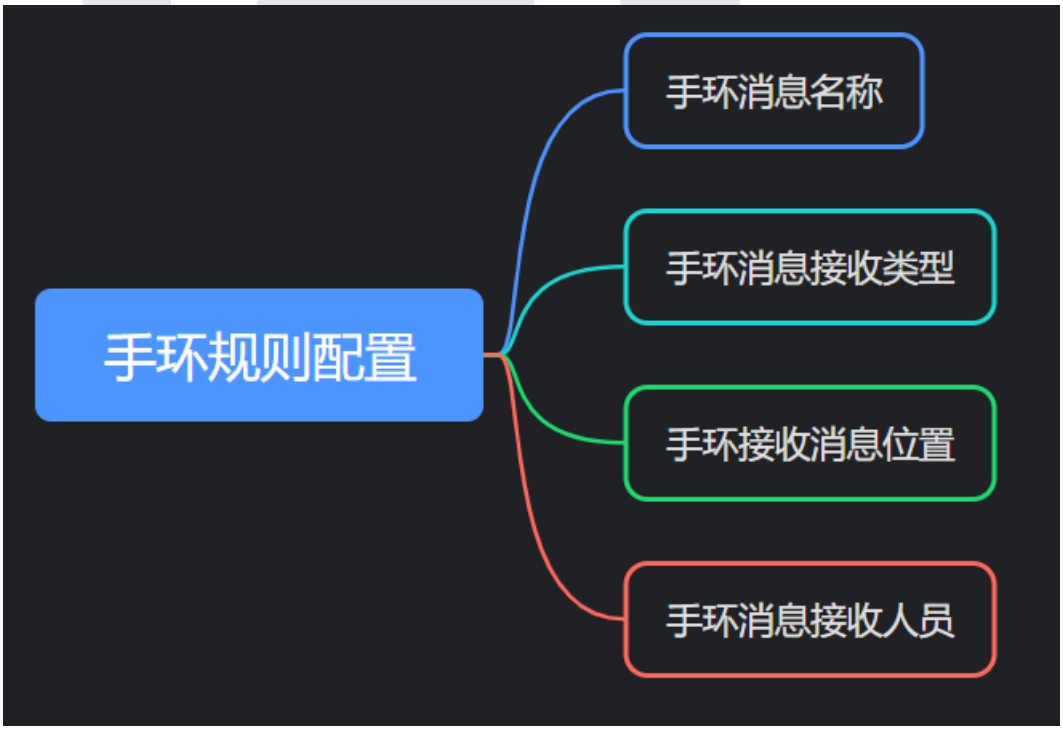

图15 手环规则配置

4.11 服务器端-MQTT 消息推送

为实现消息推送，在服务器端集成EMQX消息推送模块，用于消息推送。

4.11.1 EMQX 消息推送逻辑

服务器端使用SpringBoot框架，在pom中添加EMQX依赖，通过EMQX提供的API接口进行推送。配置如下：

（1）通过在FMS的手环消息类型分类，设置消息类型、消息模板、消息等级等信息。

（2）通过在FMS的手环消息规则配置管理，设置消息推送类型、位置、接收人员、关联页面URL地址等，配置消息推送规则。

（3）当接收到消息后，通过查找消息规则配置管理，然后通过EMQX提送的接口，通过HTTP的方式推送给相关人员。

4.11.2 EMQX 服务器端使用

EMQX服务器端推送消息方式，如下：

```
String subscriptions_result = HttpUtils.sendBasicAuthGet(mqtt_apiUrl + "subscriptions/" + userid, null);//获取客户端订阅信息
MqttSubscriptionsEntity mqttSubscriptionsEntity = JSONObject.parseObject(subscriptions_result, MqttSubscriptionsEntity.class);
if (mqttSubscriptionsEntity.getData().size() > 0) {//说明客户端已订阅信息
    MqttPublishEntity mqttPublishEntity = new MqttPublishEntity();
    mqttPublishEntity.setTopic(userid);//设置接收消息的订阅者
    mqttPublishEntity.setQos(0);//设置订阅消息等级
mqttPublishEntity.setPayload(JSONObject.toJSONString(topHisMqttMessage))
String publish_result = HttpUtils.sendBasicAuthPost(mqtt_apiUrl + "mqtt/publish", JSONObject.toJSONString(mqttPublishEntity));//接口发送消息
```

4.12 服务器端-H5 页面

服务器接收到消息后，使用theymleaf模板引擎，根据模板文件及数据进行页面渲染，展示消息详情。

4.12.1 Thymeleaf 模板引擎

Thymeleaf是一种模板引擎，是解决动态赋值给前端页面的一种解决方案，其原来如下图16所示，控制层将模板及数据输出，通过Thymeleaf模板引擎处理，将最终的html数据给到前端进行展示。

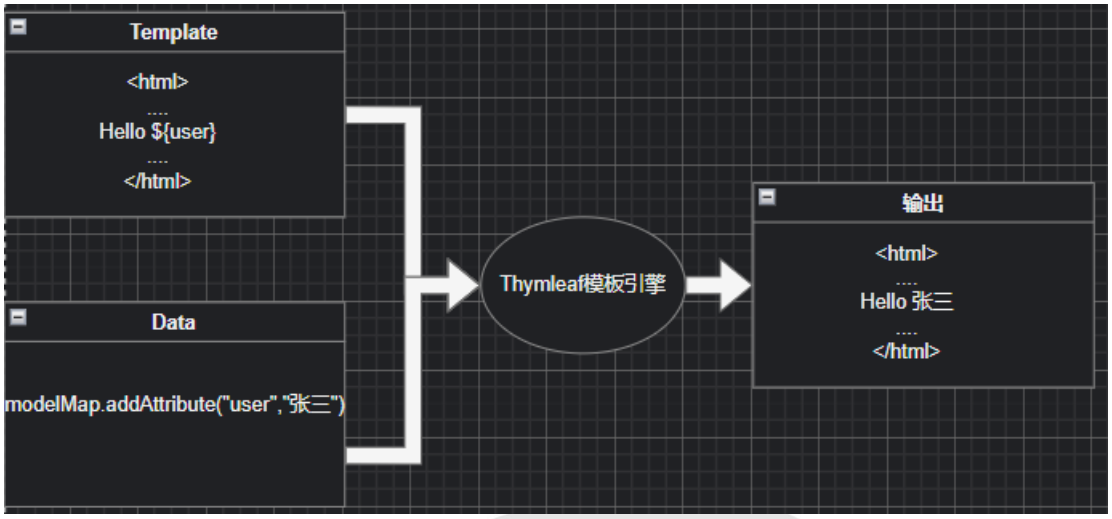

图16 Thymleaf工作原理

4.12.2 H5 功能使用

集成Thymeleaf引擎后，在指定文件夹下编写HTML消息展示文件，并根据Thymeleaf提供的语法关联数据，使用方式如下：

（1）控制层传递数据及模板地址：

```
modelMap.put("topHisMqttMessage", topHisMqttMessage);
modelMap.put("loginUserid", loginUserid);
return "common/commonhandle";
```

（2）HTML模板获取数据：

```
var topHisMqttMessage = [[${topHisMqttMessage}]];
var loginUserid=[[${loginUserid}]];
```

5 使用说明

手环信息推送平台V1.0，是根据封装模块化的思想搭建而成，是个人项目经验及技术积累的产物，因此某些模块及封装可能还有更为简便的处理方式，在使用过程中，要根据业务更好地实现需求和代码逻辑也是封装模块衡量标准之一。

版本记录

版本编号/修改状态	拟制人/修改人	审核人	批准人	备注
V1.0	赵富贵	张路	张路	