浙大城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

Android 移动端实现步骤

课程编号：C01098

课程名称：移动互联网应用开发实践

学　　号：31901056

姓　　名：姚斯安

专业班级：计算 1902

所在学院：计算学院

报告日期：2022 年 7 月 4 日

一、activity_main.xml 布局设计

VideoView 实现视频播放，ImageView 作为没有计划开始时存在的桌面，ViewFlipper 实现图片轮播，AutoScrollTextView 作为自编组件实现公告。

```
<VideoView
    android:id="@+id/videoView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:ignore="MissingConstraints" />

<ImageView
    android:id="@+id/imageView7"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="centerCrop"
    android:src="@drawable/caomei"
    tools:ignore="MissingConstraints" />

<ViewFlipper
    android:id="@+id/flipper"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:inAnimation="@anim/left_in"
    android:outAnimation="@anim/right_out"
    tools:ignore="MissingConstraints">
</ViewFlipper>

<com.example.boepicturescreen.AutoScrollTextView
    android:id="@+id/tv_text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="40dp"
    app:layout_constraintLeft_toLeftOf="parent"
    android:layout_marginRight="40dp"
    app:layout_constraintRight_toRightOf="parent"
    android:text=""
    android:inputType="text"
    tools:ignore="MissingConstraints" />
```

二、rabbitmq 消息队列连接

```
/**
 * 连接设置
 */
private void setupConnectionFactory() {
    factory =  new ConnectionFactory();
    factory.setHost(hostName);//服务器ip
    factory.setPort(portNum);//rabbitmq端口，默认5672
    factory.setUsername(userName);
    factory.setPassword(passWord);
}
```

```
//rabbitmq 连接设置
private String hostName = "121.199.49.79";//服务器ip
private int portNum = 5672 ;//rabbitmq端口，默认5672
private String userName = "control" ;
private String passWord = "control" ;
ConnectionFactory factory;
```

```java
/**
 * 收消息（从发布者那边订阅消息）
 */
private void basicConsume(final Handler handler,String infoQueue){

    try {
        //连接
        Connection connection = factory.newConnection() ;
        //通道
        final Channel channel = connection.createChannel() ;
        //实现Consumer的最简单方法是将便捷类DefaultConsumer子类化。可以在basicConsume 调用上传递此子类的对象以设置订阅：
        channel.basicConsume(infoQueue , false ,  new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
                super.handleDelivery(consumerTag, envelope, properties, body);

                String msg = new String(body) ;
                long deliveryTag = envelope.getDeliveryTag() ;
                channel.basicAck(deliveryTag , false);
                //从message池中获取msg对象更高效
                Message uimsg = handler.obtainMessage();

                //除去bom头
                if(msg.startsWith("\ufeff")){
                    msg = msg.substring(1);
                }

                Bundle bundle = new Bundle();
                bundle.putString("msg", msg);
                uimsg.setData(bundle);
                handler.sendMessage(uimsg);
            }
        });
    } catch (IOException e) {
        e.printStackTrace();
    } catch (TimeoutException e) {
        e.printStackTrace();
    }
```

创建线程连接队列，获取前端发来的消息后匹配队列具体处理消息

```java
//初始化队列名
rabbitQueueArray.add("screenshot");//发送截屏请求的队列
rabbitQueueArray.add("volume");//发送控制音量请求的队列
rabbitQueueArray.add("brightness");//发送控制屏幕亮度请求的队列
rabbitQueueArray.add("restart");//发送重启请求的队列
rabbitQueueArray.add("timing");//发送定时开关机请求的队列
rabbitQueueArray.add("scheme");//发送播放计划请求的队列
rabbitQueueArray.add("announce");//发送公告请求的队列
rabbitQueueArray.add("video");//发送播放视频的队列
rabbitQueueArray.add("upgrade");//系统升级

//运行rabbitmq,获取并处理消息
for(int i = 0; i < rabbitQueueArray.size(); i++){
    int finalI = i;
    new Thread(new Runnable() {
        @Override
        public void run() {
            Looper.prepare();
            rabbitmqGetInfo(rabbitQueueArray.get(finalI));
            Looper.loop();
        }
    }).start();
}
```

```java
//消息队列获取信息，参数为某个消息队列的queue
private void rabbitmqGetInfo(String infoqueue){
    //rabbitmq 连接设置（测试ok）
    setupConnectionFactory();
    //用于从线程中获取数据
    final Handler incomingMessageHandler = handleMessage(msg) -> {
            String message = msg.getData().getString("msg");

            //String转JSONObject
            JSONObject result = null;
            try {
                result = new JSONObject(message);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            Log.i(TAG,"result:"+result);
            Log.i(TAG, "msg:"+message);

            JSONObject finalResult = result;
            new Thread(new Runnable() {
                @Override
                public void run() { mainFunction(message, finalResult); }
            }).start();
    };
    //开启消费者线程
    new Thread(new Runnable() {
        @Override
        public void run() { basicConsume(incomingMessageHandler,infoqueue); }
    }).start();
}
```

```java
//处理消息
private void mainFunction(String msg, JSONObject object){
    if(msg != null && object != null){
        Log.i(TAG,"rabbitMsg != null && rabbitJson != null!!!");
        try {
            if(object.getString("category").equals("screenshot")){//如果接收到截屏请求
                //发送图片给后端
                //截屏，测试ok
                final Bitmap[] screen = {null};
                new Thread(new Runnable() {
                    @Override
                    public void run() { screen[0] = Screenshots(); }
                }).start();

                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
```

三、具体功能实现

1、截屏功能

Screenshots 函数获取屏幕截图并将截图保存到 data 路径下的一个文件夹，保存为 screen.jpg

```
/**                                                                    ⚠ 98  ⚠ 6  ✓ 109  ^
 * 屏幕截图
 */
private Bitmap Screenshots(){
    String screenname = "screen.jpg";
    View view = getWindow().getDecorView();
    view.setDrawingCacheEnabled(true);
    view.buildDrawingCache();
    Bitmap bmp = view.getDrawingCache();
    Log.i(TAG, "bmp: "+bmp);
    if (bmp != null)
    {
        try {
            Log.i(TAG, "BaseContext: "+getBaseContext().getFilesDir());
            String TargetPath = "/data/data/com.example.boepicturescreen/files" + "/myimages";
            if (!FileUtils.fileIsExist(TargetPath)) {
                Log.d("Save Bitmap", "TargetPath isn't exist");
            }else{
                File file = new File(TargetPath,screenname);
                FileOutputStream fos = new FileOutputStream(file);
                bmp.compress(Bitmap.CompressFormat.PNG, 100, fos);
                fos.flush();
                fos.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //一定记得设置回来，清空缓存，不然总是截第一张照片
    view.setDrawingCacheEnabled(false);
```

消息队列接收到截屏请求后调用 Screenshots 函数截屏，之后开一个线程，创建 okhttp 客户端，调用后端接口将该图片上传给 web 端。

```java
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            OkHttpClient client = new OkHttpClient();//创建http客户端
            File file = new File("/data/data/com.example.boepicturescreen/files/myimages/screen.jpg");//被上传的文件，需要注意权限
            MultipartBody.Builder requestBody = new MultipartBody.Builder().setType(MultipartBody.FORM);//通过表单上传
            RequestBody fileBody = RequestBody.create(MediaType.parse("image/*"),file);//上传的文件以及类型
            requestBody.addFormDataPart("photo",file.getName(),fileBody);//参数，1、请求key；2、文件名称；3、filebody
            Log.i(TAG,"address:"+ipv4address+screenshotupload);
            Request request = new Request.Builder()
                    .url(ipv4address+screenshotupload)
                    .post(requestBody.build())
                    .build();//创造http请求
            client.newBuilder().readTimeout(60000, TimeUnit.MILLISECONDS).build().newCall(request).enqueue(new Callback() {
                @Override
                public void onFailure(@NonNull Call call, @NonNull IOException e) {
                    Log.d("文件上传","失败!");
                    e.printStackTrace();
                }

                @Override
                public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {
                    if(response.isSuccessful()){
                        try {
                            JSONObject jsonObject = new JSONObject(response.body().string());
                            JSONObject x = jsonObject.getJSONObject("data");
                              imgUrl = x.getString("murl");
                            Log.d("返回体",jsonObject.toString());

                            Log.d("文件上传成功",jsonObject.getString("rspCode"));
                              Log.d("服务器上的文件",imgUrl);
                        }catch (Exception e){
                            e.printStackTrace();
                        }
                    }else {
                        Log.d("文件上传",response.message()+"error:body "+response.body().string());
                    }
```

2、控制音量

消息队列接收到控制音量请求后将音量值从 json 数据中提取出来，调用 audioManager 函数将当前应用程序音量调为该值

```java
}else if(object.getString("category").equals("volume")){//如果接收到控制音量请求
    //控制音量（测试ok）
    int tempVolume = 90;//暂时变量，后序接收到信息修改音量变化情况
    JSONObject newjsonobject = object.getJSONObject("content");
    tempVolume = newjsonobject.getInt("volume");
    int finalTempVolume = tempVolume;
    new Thread(new Runnable() {
        @Override
        public void run() {
            audioManager(finalTempVolume);//tempVolume消息发过来的设定的音量值，比如50、100
        }
    }).start();
```

```
/**
 * 控制音量
 */
private void audioManager(int tempVolume){
    //音量控制,初始化定义
    AudioManager audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
    //最大音量
    int maxVolume = audioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
    //当前音量
    int currentVolume = audioManager.getStreamVolume(AudioManager.STREAM_MUSIC);
//      //一步步长控制音量的增减
//      if(updateaudio.equals("RAISE")) {//升高
//          audioManager.adjustStreamVolume(AudioManager.STREAM_MUSIC, AudioManager.ADJUST_RAISE, AudioManager.FLAG_SHOW_UI);
//      }else if(updateaudio.equals("LOWER")){//降低
//          audioManager.adjustStreamVolume (AudioManager.STREAM_MUSIC, AudioManager.ADJUST_LOWER , AudioManager.FLAG_SHOW_UI);
//      }else{//保持不变
//          audioManager.adjustStreamVolume (AudioManager.STREAM_MUSIC, AudioManager.ADJUST_SAME , AudioManager.FLAG_SHOW_UI);
//      }
    //直接控制音量的多少
    double newtempVolume;
    newtempVolume = (((double)tempVolume)/100)*maxVolume;//不同类型的音量，Android规定了不同的范围，STREAM_MUSIC最大值maxVolume为15，这里
    tempVolume = (int)Math.round(newtempVolume);//先转为double获取相对值然后通过round转为相对该系统的绝对值
    if(tempVolume == 0){
        audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 0, AudioManager.FLAG_SHOW_UI);
    }else{
        audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, tempVolume, AudioManager.FLAG_SHOW_UI); //tempVolume:音量绝对值
    }
}
```

3、控制屏幕亮度

方法跟控制音量类似，用 window.setAttributes 函数将当前窗口屏幕亮度改为传过来的特定值

```
/**
 *调节当前屏幕亮度
 */
public static void SetSystemLight(int lightnumber, Activity activity){
    Window window = activity.getWindow();//对当前窗口进行设置
    WindowManager.LayoutParams layoutparams = window.getAttributes();//获取窗口属性为后面亮度做铺垫作用
    layoutparams.screenBrightness = Float.valueOf(lightnumber) * (1f / 255f);//用窗口管理（自定义的）layoutparams获取亮度值，android亮度值处于在0-255之间的整形数值
    window.setAttributes(layoutparams);//设置当前窗口屏幕亮度
}

}
```

4、重启 app

Android 的 Service 实现后台定时检测并重启应用，Service(服务)是一个一种可以在后台执行长时间运行操作而没有用户界面的应用组件。服务可由其他应用组件启动（如 Activity），服务一旦被启动将在后台一直运行，即使启动服务的组件（Activity）已销毁也不受影响。此外,组件可以绑定到服务,以与之进行交互,甚至是执行进程间通信 (IPC)。例如，服务可以处理网络事务、播放音乐，执行文件 I/O 或与内容提供程序交互，而所有这一切均可在后台进行。

RestartAppService 服务类继承自 Service：

```java
 * 重启app服务
 */
public class RestartAppService extends Service {
    private static final String TAG = "RestartAppService";
    private static final long RESTART_DELAY = 2000; // 多少时间后重启检测(2秒),收到指令2秒后重启
    private MyBinder mBinder;

    // 此对象用于绑定的service与调用者之间的通信
    public class MyBinder extends Binder {

        /**
         * 获取service实例
         * @return
         */
        public RestartAppService getService() { return RestartAppService.this; }

        /**
         * 启动app重启任务
         */
        public void startRestartTask(final Context context) {
            Toast.makeText(context, "restart check", Toast.LENGTH_SHORT).show();
            Log.e(TAG,"restart app check");
            TimerTask task = () -> {
                    //定时时间到
                    String curtime = DateUtils.getCurrent();
                    //处理时段内
                    Log.e(TAG,curtime);
                    Log.e(TAG,"this time,begin restart"); // restart
                    Intent intent = getPackageManager().getLaunchIntentForPackage(
                            getApplication().getPackageName());
                    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
                    startActivity(intent);
                    System.exit(0);
            };

            Timer timer = new Timer();
            timer.schedule(task, RESTART_DELAY);
        }
    }
```

之后在 AndroidManifest.xml 中声明：

```xml
<!-- 重启app权限 -->
<service android:name=".RestartAppService"
    android:enabled="true"
    android:exported="true"/>
```

android:exported　　表示是否允许除了当前程序之外的其他程序访问这个服务

android:enabled　　表示是否启用这个服务

最后在 Application 或 Activity 中完成 Service 的绑定和启动服务：

```
/**
 * 重启
 */
private RestartAppService myService;
private ServiceConnection connService = new ServiceConnection() {
    /**
     * 与服务器端交互的接口方法 绑定服务的时候被回调，在这个方法获取绑定Service传递过来的IBinder对象，
     * 通过这个IBinder对象，实现宿主和Service的交互。
     */
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        RestartAppService.MyBinder mBinder = (RestartAppService.MyBinder) service;
        myService = mBinder.getService();
        mBinder.startRestartTask(getApplicationContext());
    }
    @Override
    public void onServiceDisconnected(ComponentName name) { myService = null; }
};
private void appRestart(){
    //立即重启（2秒后重启）测试ok
    Context mContext = getApplicationContext();
    Log.i(TAG, "==app onCreate==");
    // 记录异常日志
    //创建绑定对象并绑定服务，用于定时重启app
    Intent intent = new Intent(this, RestartAppService.class);
    bindService(intent, connService, Context.BIND_AUTO_CREATE);
}
```

5、定时开关机

接收到定时开关机请求后将发生过来的时间格式化，使用 AlarmManager 搭配 Receiver 可以实现定时自动启动应用程序：

```
String now = DateUtils.getNow();
long subtime = DateUtils.timeSub(now,starttime);
if(subtime>0){
    Log.i(TAG, "now:"+now);
    Log.i(TAG, "starttime:"+starttime);
    Log.i(TAG, "subtime:"+subtime);

    AlarmManager am = (AlarmManager)this.getSystemService(Context.ALARM_SERVICE);
    Intent intent = new Intent(MYACTION);
    PendingIntent pi = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    am.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()+(subtime*1000), pi);//启动
}
```

使用 Calendar 日历与 Timer 延时搭配可以简单实现定时关机

```
/**
 * 定时关闭app
 */
private void timeToCloseApp(String closetime){
    long PERIOD_DAY = 24 * 60 * 60 * 1000;

    Calendar calendar = Calendar.getInstance();

    Date startdate = DateUtils.stringtoDate(closetime,"yyyy-MM-dd HH:mm:ss");
    int hourofday = startdate.getHours();
    int minute = startdate.getMinutes();
    int seconds = startdate.getSeconds();

    calendar.set(Calendar.HOUR_OF_DAY, hourofday);
    calendar.set(Calendar.MINUTE, minute);
    calendar.set(Calendar.SECOND, seconds);

    Date date = calendar.getTime();//第一次执行任务的时间

    if (date.before(new Date())) {
        date = this.addDay(date, 1);
    }
    Timer timer = new Timer();

    timer.schedule(() -> {
            //发布设备状态：离线
            setReturnAppState("off");

            closeapp();
    }, date, PERIOD_DAY);
```

6、播放轮播图

接收到播放请求后处理发送过来的图片 url、轮播开始时间、结束时间等一系列数据，创建线程在播放之前下载 url 中的图片，使用 Timer 定时器类实现在规定时间开始与关闭轮播。

```
//创建线程下载处理图片
new Thread(new Runnable() {
    @Override
    public void run() {
        Looper.prepare();
        for (int i = 0;i<programImagesUrl.size();i++){
            asyncGet(programImagesUrl.get(i));
        }
        Looper.loop();
    }
}).start();
String now = DateUtils.getNow();
long subtimestart = DateUtils.timeSub(now,starttime);
long subtimeend = DateUtils.timeSub(now,endtime);

Log.i(TAG,"now:"+now);

if(subtimestart<subtimeend){
    if(subtimestart>0){//如果还没到开始时间
        Timer mTimer = new Timer();
        mTimer.schedule(() -> {
                try {
                    timeToStartProgram(delay);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
        },subtimestart*1000);
        mTimer.schedule(() -> {
                timeTostopProgram();
        },subtimeend*1000);
```

```
    }else {
        if(subtimeend>0){//如果过了开始时间但是还没到结束时间
            Timer mTimer = new Timer();
            mTimer.schedule(() -> {
                    try {
                        timeToStartProgram(delay);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
            },0);//立即开始
            mTimer.schedule(() -> {
                    timeTostopProgram();
            },subtimeend*1000);
        }
    }
}else{
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(MainActivity.this,"播放失败",Toast.LENGTH_SHORT).show();
        }
    });
    Log.d("播放失败","原因：结束时间早于开始时间");
}
```

开始轮播时将桌面 imageview 隐藏，由于视频组件优先级高于轮播，因此不隐藏视频组件，以便于两个计划同时出现时视频在上面播放，播放玩后隐藏视频组件继续播放轮播。

```
//轮播函数
private void timeToStartProgram(int delay) throws InterruptedException {
    Log.i(TAG, "program start!!!!!!!!!!");
    isPlayingFlipper = true;

    //发布设备状态：播放
    setReturnAppState("palying");

    //隐藏桌面
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            ImageView imageView = (ImageView) findViewById(R.id.imageView7);
            imageView.setVisibility(View.INVISIBLE);
        }
    });

    //实现图片轮播
    ViewFlipper flipper = findViewById(R.id.flipper);
    //显示轮播
    runOnUiThread(new Runnable() {
        @Override
        public void run() { flipper.setVisibility(View.VISIBLE); }
    });

    //动态导入添加子View
    for(int i = 0;i < imageViews.size();i++){
        int finalI = i;
        runOnUiThread(new Runnable() {
            @Override
            public void run() { flipper.addView(imageViews.get(finalI)); }
        });
    }
    flipper.setFlipInterval(delay*1000+1000);//设定轮播延时，延时比传来的延时数据+1秒才正常
    flipper.startFlipping();
}
```

关闭轮播时查看是否在播放视频，如果不在播放视频则显示桌面

```
private void timeTostopProgram(){
    Log.i(TAG, "program stop!!!!!!!!!!");
    isPlayingFlipper = false;

    //隐藏轮播
    ViewFlipper flipper = findViewById(R.id.flipper);
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            flipper.stopFlipping();
            flipper.setVisibility(View.INVISIBLE);

            if(isPlayingmp4 = false){
                //显示桌面
                ImageView imageView = (ImageView) findViewById(R.id.imageView7);
                imageView.setVisibility(View.VISIBLE);
            }
        }
    });

    //发布设备状态：空闲
    setReturnAppState("relax");
```

7、公告

处理公告请求时先处理文字等信息，使用 Timer 定时器实现定时播放

```
        String now = DateUtils.getNow();
        long subtimestart = DateUtils.timeSub(now,starttime);
        long subtimefinish = DateUtils.timeSub(now,finishtime);
        Log.i(TAG,"subtimestart: "+subtimestart);
        Log.i(TAG,"subtimefinish: "+subtimefinish);
        if(subtimestart<subtimefinish){
            if(subtimestart>0){//如果还没到开始时间
                Timer mTimer = new Timer();
                String finalTextcontent = textcontent;
                mTimer.schedule(() -> {
                        timeToStartAnnounce(finalTextcontent,textsize,textcolor,backgroundcolor);
                },(subtimestart*1000));
                mTimer.schedule(() -> {
                        timeToFinishAnnounce();
                },(subtimefinish*1000));
            }else {
                if(subtimefinish>0){//如果过了开始时间但是还没到结束时间
                    Timer mTimer = new Timer();
                    String finalTextcontent1 = textcontent;
                    mTimer.schedule(() -> {
                            timeToStartAnnounce(finalTextcontent1,textsize,textcolor,backgroundcolor);
                    },0);
                    mTimer.schedule(() -> {
                            timeToFinishAnnounce();
                    },(subtimefinish*1000));
                }
            }
        }else{
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(MainActivity.this,"播放失败",Toast.LENGTH_SHORT).show();
                }
            });
            Log.d("播放失败","原因：结束时间早于开始时间");
```

实现公告的文字组件采用自编组件 AutoScrollTextView 实现文字的滚动效果，如果是

安卓自带的滚动效果则是需要文字长度大于当前页面才会滚动，无法实现几个字的滚动效果。

```
/**
 * 发布公告
 */
private void timeToStartAnnounce(String textcontent, int textsize, String textcolor, String textbackgroundcolor){
    //发布设备状态：播放
    setReturnAppState("playing");

    AutoScrollTextView tv = (AutoScrollTextView) findViewById(R.id.tv_text);
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            tv.setText(textcontent);
            tv.setTextSize(textsize);
            tv.setTextColor(Color.parseColor(textcolor));
            tv.setBackgroundColor(Color.parseColor(textbackgroundcolor));
            tv.init(getWindowManager(),textcolor);

            tv.startScroll();
            tv.setVisibility(View.VISIBLE);
        }
    });
}
private void timeToFinishAnnounce(){
    AutoScrollTextView tv = (AutoScrollTextView) findViewById(R.id.tv_text);
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            tv.stopScroll();
            tv.setVisibility(View.INVISIBLE);
        }
    });

    //发布设备状态：空闲
    setReturnAppState("relax");
}
```

```
public class AutoScrollTextView extends androidx.appcompat.widget.AppCompatTextView implements OnClickListener {
    public final static String TAG = AutoScrollTextView.class.getSimpleName();

    private float textLength = 0f;//文本长度
    private float viewWidth = 0f;
    private float step = 0f;//文字的横坐标
    private float y = 0f;//文字的纵坐标
    private float temp_view_plus_text_length = 0.0f;//用于计算的临时变量
    private float temp_view_plus_two_text_length = 0.0f;//用于计算的临时变量
    public boolean isStarting = false;//是否开始滚动
    private Paint paint = getPaint();//绘图样式
    private String text = "";//文本内容
```

8、播放视频

视频播放与轮播图实现逻辑基本一致，不同在于图片轮播是将图片下载到本地轮播，因为考虑到视频的体量远远大于图片，下载会消耗大量资源，所以视频是直接利用缓存播放网络视频。

```
    /**
     * 播放视频
     */
    private void play_mp4(String mp4url) throws IOException {
        //发布设备状态：播放
        setReturnAppState("playing");
        isPlayingmp4 = true;

        ViewFlipper flipper = findViewById(R.id.flipper);
        ImageView imageView = (ImageView) findViewById(R.id.imageView7);

        MediaController mediaController=new MediaController(this);
        VideoView videoView = (VideoView)this.findViewById(R.id.videoView);
```

```
runOnUiThread(new Runnable() {
    @Override
    public void run() {
        //隐藏轮播
        flipper.stopFlipping();
        flipper.setVisibility(View.INVISIBLE);

        //隐藏桌面
        imageView.setVisibility(View.INVISIBLE);

        String videoUrl = mp4url ;
        Log.i(TAG, "origin uri: "+videoUrl);
        Uri uri = Uri.parse(videoUrl);
        Log.i(TAG, "parsed uri: "+uri);

        videoView.setVideoURI(uri);
        videoView.requestFocus();

        mediaController.setMediaPlayer(videoView);
        videoView.setMediaController(mediaController);

        //显示视频
        videoView.setVisibility(View.VISIBLE);
        videoView.start();
    }
```

```
private void stop_mp4() throws IOException {
    VideoView videoView = (VideoView)this.findViewById(R.id.videoView);
    isPlayingmp4 = false;

    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            videoView.stopPlayback();

            //隐藏视频
            videoView.setVisibility(View.INVISIBLE);

            if(isPlayingFlipper == false){
                //显示桌面
                ImageView imageView = (ImageView) findViewById(R.id.imageView7);
                imageView.setVisibility(View.VISIBLE);
            }
        }
    });

    //发布设备状态：空闲
    setReturnAppState("relax");
```

9、自动更新

步骤 1、申明权限：由于自动更新需要访问网络，下载更新包，执行安装操作，所以需要申明以下权限：

```
<!-- 网络权限 -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<!-- 在SDCard中创建与删除文件权限 -->
<uses-permission
    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
    tools:ignore="ProtectedPermissions" />

<!-- 存储权限 -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!-- 安装APK权限 -->
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" />
```

步骤 2、设置 xml 文件设置安装包文件存储路径

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.and
    <!--安装包文件存储路径-->
    <external-files-path
        name="my_download"
        path="Download" />
    <external-path
        name="."
        path="." />
</paths>
```

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true" />
</network-security-config>
```

步骤 3、添加更新进度布局，里面就一个显示百分比的文本框，和一个进度条。

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/an
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:id="@+id/titleBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/txtStatus"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="状态"
            android:textSize="10sp"
            android:textStyle="normal" />

        <ProgressBar
            android:id="@+id/progress"
            style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/txtStatus" />
    </LinearLayout>
</LinearLayout>
```

步骤 4、把检查更新，下载 apk，安装 apk 等操作封装成一个类 AutoUpdater

```java
public class AutoUpdater {
    private static String TAG = AutoUpdater.class.getSimpleName();
    // 下载安装包的网络路径
    private String apkUrl = "";
    protected String checkUrl = "";

    // 保存 APK 的文件名
    private static final String saveFileName = "BOEAPK.apk";
    private static File apkFile;

    // 下载线程
    private Thread downLoadThread;
    private int progress;// 当前进度
    // 应用程序 Context
    private Context mContext;
    // 是否是最新的应用,默认为 false
    private boolean isNew = false;
    private boolean intercept = false;
    // 进度条与通知 UI 刷新的 handler 和 msg 常量
    private ProgressBar mProgress;
    private TextView txtStatus;
```

```
    private static final int DOWN_UPDATE = 1;
    private static final int DOWN_OVER = 2;
    private static final int SHOWDOWN = 3;

    public AutoUpdater(Context context) {
        mContext = context;
        apkFile = new
File(mContext.getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS),
saveFileName);

        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    FormBody.Builder parms = new FormBody.Builder();
                    OkHttpClient client = new OkHttpClient();
                    Request request = new Request.Builder()
                            .url("http://47.99.158.248:8080/devices/getap
k")
                            .post(parms.build())
                            .build();
                    Response response =
client.newCall(request).execute();
                    String returnok = response.body().string();
                    JSONObject ab = new JSONObject(returnok);
                    JSONObject cd = ab.getJSONObject("data");
                    apkUrl = cd.getString("apk_url");
                    Log.i(TAG, "apk_url: "+apkUrl);
                    checkUrl = cd.getString("json_url");
                    Log.i(TAG, "checkUrl: "+checkUrl);
                }catch (Exception e){
                    e.printStackTrace();
                }
            }
        }).start();
    }

    public void ShowUpdateDialog() {
//        AlertDialog.Builder builder = new
AlertDialog.Builder(mContext);
//        builder.setTitle("软件版本更新");
//        builder.setMessage("有最新的软件包，请下载并安装!");
//        builder.setPositiveButton("立即下载", new
DialogInterface.OnClickListener() {
```

```
//           @Override
//           public void onClick(DialogInterface dialog, int which) {
//               ShowDownloadDialog();
//           }
//       });
//       builder.setNegativeButton("以后再说", new
DialogInterface.OnClickListener() {
//           @Override
//           public void onClick(DialogInterface dialog, int which) {
//               dialog.dismiss();
//           }
//       });
//
//       builder.create().show();

        ShowDownloadDialog();
    }

    private void ShowDownloadDialog() {
        AlertDialog.Builder dialog = new
AlertDialog.Builder(mContext);
        dialog.setTitle("软件版本更新");
        LayoutInflater inflater = LayoutInflater.from(mContext);
        View v = inflater.inflate(R.layout.progress, null);
        mProgress = (ProgressBar) v.findViewById(R.id.progress);
        txtStatus = v.findViewById(R.id.txtStatus);
        dialog.setView(v);
        dialog.setNegativeButton("取消", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                intercept = true;
            }
        });
        dialog.show();
        DownloadApk();
    }

    /**
     * 检查是否更新的内容
     */
    public void CheckUpdate() {
        new Thread(new Runnable() {
            @RequiresApi(api = Build.VERSION_CODES.O)
```

```
            @Override
            public void run() {
                String localVersion = "1";
                try {
                    localVersion =
mContext.getPackageManager().getPackageInfo(mContext.getPackageName()
, 0).versionName;
                } catch (PackageManager.NameNotFoundException e) {
                    e.printStackTrace();
                }
                String versionName = "1";
                String outputFile = "";
                String config = doGet(checkUrl);
                Log.i(TAG, "config: "+config);
                if (config != null && config.length() > 0) {
                    Log.i(TAG, "config != null && config.length() > 0");
                    Log.i(TAG, "Build.VERSION.SDK_INT:  "+
Build.VERSION.SDK_INT);

                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N)
{
                        Matcher m =
Pattern.compile("\"outputFile\":\\s*\"(?<m>[^\"]*?)\"").matcher(confi
g);
                        Log.i(TAG, "m1: "+m);

//                        if (m.find()) {
//                            outputFile = m.group("m");
//                        }
//                        m =
Pattern.compile("\"versionName\":\\s*\"(?<m>[^\"]*?)\"").matcher(conf
ig);
//                        Log.i(TAG, "m2: "+m);
//                        if (m.find()) {
//                            String v = m.group("m");
//                            Log.i(TAG, "v: "+v);
//                            versionName = m.group("m").replace("v1.0.",
"");
//                        }
                        try {
                            JSONObject jsonObject = new
JSONObject(config);
                            String str=jsonObject.getString("elements");
                            JSONArray array= new JSONArray(str);
```

```
                        JSONObject Data
=(JSONObject)(array.getJSONObject(0));
                        String ver=Data.getString("versionName");
                        Log.i(TAG, "versionName: "+ver);
                        Log.i(TAG, "localVersion: "+localVersion);
                        versionName = ver;
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }
            if (Float.parseFloat(localVersion) <
Float.parseFloat(versionName)) {
                apkUrl = apkUrl;//ysa 修改，原：apkUrl = apkUrl +
outputFile;
                mHandler.sendEmptyMessage(SHOWDOWN);
            } else {
                Log.i(TAG, "已是最新版本无需更新");
                return;
            }
        }
    }).start();
}

/**
 * 从服务器下载 APK 安装包
 */
public void DownloadApk() {
    downLoadThread = new Thread(DownApkWork);
    downLoadThread.start();
}

private Runnable DownApkWork = new Runnable() {
    @Override
    public void run() {
        URL url;
        try {
            url = new URL(apkUrl);
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.connect();
            int length = conn.getContentLength();
            InputStream ins = conn.getInputStream();
            FileOutputStream fos = new FileOutputStream(apkFile);
```

```
                int count = 0;
                byte[] buf = new byte[1024];
                while (!intercept) {
                    int numread = ins.read(buf);
                    count += numread;
                    progress = (int) (((float) count / length) * 100);
                    // 下载进度
                    mHandler.sendEmptyMessage(DOWN_UPDATE);
                    if (numread <= 0) {
                        // 下载完成通知安装
                        mHandler.sendEmptyMessage(DOWN_OVER);
                        break;
                    }
                    fos.write(buf, 0, numread);
                }
                fos.close();
                ins.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };

    /**
     * 安装APK内容
     */
    public void installAPK() {
        try {
            if (!apkFile.exists()) {
                Log.i(TAG,"!apkFile.exists()");
                return;
            }

            Intent intent = new Intent(Intent.ACTION_VIEW);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);//安装完成后打开新版本
            intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION); //给目标应用一个临时授权
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {//判断版本大于等于7.0
                //如果SDK版本>=24，即：Build.VERSION.SDK_INT >= 24，使用FileProvider兼容安装apk
                String packageName =
```

```
mContext.getApplicationContext().getPackageName();
                String authority = new
StringBuilder(packageName).append(".fileprovider").toString();
                Uri apkUri = FileProvider.getUriForFile(mContext,
authority, apkFile);
                intent.setDataAndType(apkUri,
"application/vnd.android.package-archive");
            } else {
                intent.setDataAndType(Uri.fromFile(apkFile),
"application/vnd.android.package-archive");
            }
            mContext.startActivity(intent);

android.os.Process.killProcess(android.os.Process.myPid());//安装完之
后会提示"完成" "打开"。

            Log.i(TAG,"安装成功!!!!!!!!!!!!!!!!!!!!");

        } catch (Exception e) {
        }
    }

    private Handler mHandler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            switch (msg.what) {
                case SHOWDOWN:
                    ShowUpdateDialog();
                    break;
                case DOWN_UPDATE:
                    txtStatus.setText(progress + "%");
                    mProgress.setProgress(progress);
                    break;
                case DOWN_OVER:
                    Toast.makeText(mContext, "下载完毕",
Toast.LENGTH_SHORT).show();
                    installAPK();
                    break;
                default:
                    break;
            }
        }

    };
```

```
    public static String doGet(String httpurl) {
        HttpURLConnection connection = null;
        InputStream is = null;
        BufferedReader br = null;
        String result = null;
        try {
            URL url = new URL(httpurl);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setConnectTimeout(15000);
            connection.setReadTimeout(60000);
            connection.connect();
            Log.i(TAG, "connection.getResponseCode():
"+connection.getResponseCode());
            if (connection.getResponseCode() == 200) {
                is = connection.getInputStream();
                br = new BufferedReader(new InputStreamReader(is, "UTF-
8"));
                StringBuffer sbf = new StringBuffer();
                String temp = null;
                while ((temp = br.readLine()) != null) {
                    sbf.append(temp);
                    sbf.append("\r\n");
                }
                result = sbf.toString();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (null != is) {
                try {
                    is.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
```

```
            }

            connection.disconnect();
        }
        return result;
    }
}
```

步骤 5、在 MainActivity.java 中进行检查配置。在方法中加入代码：处理权限申请

```
//检查更新
try {
    //6.0才用动态权限
    if (Build.VERSION.SDK_INT >= 23) {
        String[] permissions = {
                Manifest.permission.READ_EXTERNAL_STORAGE,
                Manifest.permission.WRITE_EXTERNAL_STORAGE,
                Manifest.permission.ACCESS_WIFI_STATE,
                Manifest.permission.INTERNET};
        List<String> permissionList = new ArrayList<>();
        for (int i = 0; i < permissions.length; i++) {
            if (ActivityCompat.checkSelfPermission(this, permissions[i]) != PackageManager.PERMISSION_GRANTED) {
                permissionList.add(permissions[i]);
            }
        }
        if (permissionList.size() <= 0) {
            //说明权限都已经通过，可以做你想做的事情去
            //自动更新
            Looper.prepare();
            AutoUpdater manager = new AutoUpdater(MainActivity.this);
            Thread.sleep(500);
            manager.CheckUpdate();
            Looper.loop();
        } else {
            //存在未允许的权限
            ActivityCompat.requestPermissions(this, permissions, 100);
        }
    }
} catch (Exception ex) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(MainActivity.this, "自动更新异常: " + ex.getMessage(), Toast.LENGTH_SHORT).show();
            Log.i(TAG,"自动更新异常: "+ ex.getMessage());
        }
    });
```