闪光灯功能预研报告整理

闪光灯功能的判断

闪光灯功能必然需要用到闪光灯，所以如何判断是否有闪光灯功能是首先需要解决的问题

- PackageManagerService

PKMS中的方法 `hasSystemFeature` 可以用来判断系统的一些特性，如下方式即可判断是否支持闪光灯功能

```
PackageManager pkgMgr = context.getPackageManager();
pkgMgr.hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH);
```

但是并不是十分可靠，StackOverFlow上关于该问题的描述

- Carmera.Parameters

通过 `Carmera.Parameters` 可以获取到Carmera的一些包括是否支持闪光灯的参数，这个API还是比较可靠的，但是需要获取Camera资源，需要考虑资源的释放，如下是使用方法

```
    Parameters parameters = mCamera.getParameters();
    if (parameters.getFlashMode() == null) {
        return false;
    }
    List<String> supportedFlashModes = parameters
            .getSupportedFlashModes();
    if (supportedFlashModes == null
            || supportedFlashModes.isEmpty()
            || supportedFlashModes.size() == 1
            && supportedFlashModes.get(0).equals(
            Parameters.FLASH_MODE_OFF)) {
        return false;
```

- Carmera2

从5.0开始，可以完全控制安卓设备相机的新Camera2（android.hardware.Camera2）API被引入了进来。在以前的Camera（android.hardware.Camera）中，对相机的手动控制需要更改系统才能实现，而且API也不友好。老的CameraAPI在5.0上已经过时，在未来的app开发中推荐的是Camera2，使用Camera2检测闪光灯功能的方式如下：

```
mCameraManager = (CameraManager) context.getSystemService(Context.CAMERA_SERVICE);

String[] cameraIdList = mCameraManager.getCameraIdList();
if (cameraIdList == null || cameraIdList.length == 0) {
    return false;
}

String identify;
CameraCharacteristics cameraCharacteristics;
for (int i = 0; i < cameraIdList.length; i++) {
    identify = cameraIdList[i];
    cameraCharacteristics = mCameraManager.getCameraCharacteristics(identify);
    isAvailable = cameraCharacteristics.get(CameraCharacteristics.FLASH_INFO_AVAILABLE);
    if (isAvailable) {
        mFirstAvildId = identify;
        return true;
    }
}
```

且这个过程不需要获取Camera资源，所以不需要考虑资源的释放

打开方式

- 使用Camera的API 最常见的方式是通过打开系统Camera方式来，同时把摄像预览界面的大小设置成1px并开启闪光灯模式来实现，这过程打开了Camera，所以需要在不使用的时候得释放Camera，使用方式包括摄像头的开启、摄像预览的 `SurfaceView` 的创建、摄像头参数设置和开始摄像，过程比较繁琐

开启摄像头

```
mCamera = Camera.open();
```

定义预览的 `SurfaceView`

```
private class MySurfaceView extends SurfaceView implements
        SurfaceHolder.Callback {

    public MySurfaceView(Context context) {
        super(context);
        initHolder();
    }

    @SuppressWarnings("deprecation")
    private void initHolder() {
        SurfaceHolder holder = getHolder();
        holder.addCallback(this);
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS); // 在某些机型上，这一项必须设置
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        mHolder = holder;
        setLightOn();

    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
                               int height) {
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        setLightOff();
        mHolder = null;
    }

}
```

创建预览 `SurfaceView`

```
private void createSurfaceView() {
    try {
        mWindowManager = (WindowManager) mContext
                .getSystemService(Context.WINDOW_SERVICE); // 获取WindowManager
        // 添加手电筒所需要的SurfaceView
        mFlashTorchSurface = new MySurfaceView(mContext);
        // 由于会在SurfaceView中进行预览，所以将背景色设置为黑色，使用户看不到
        mFlashTorchSurface.setBackgroundColor(Color.BLACK);
        WindowManager.LayoutParams mCallViewParams = new WindowManager.LayoutParams(); // 设置LayoutParams(
        mCallViewParams.type = WindowManager.LayoutParams.TYPE_SYSTEM_ALERT;
        mCallViewParams.format = PixelFormat.RGBA_8888; // 设置透明背景
        mCallViewParams.width = 1;
        mCallViewParams.height = 1;
        mCallViewParams.gravity = Gravity.RIGHT | Gravity.TOP;
        mCallViewParams.flags = WindowManager.LayoutParams.FLAG_NOT_TOUCH_MODAL
                | WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE; // 让当前View失去焦点
        mWindowManager.addView(mFlashTorchSurface, mCallViewParams);
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
            mFlashTorchSurface = null;
        }
    }
```

摄像头参数设置和开启摄像

```
    private void setLightOn() {
        if (mCamera == null) {
            return;
        }
        try {
            // 防止startPreview、setParameters抛出RuntimeException异常导致崩溃
            Parameters param = mCamera.getParameters();
            param.setFlashMode(Parameters.FLASH_MODE_TORCH);
            mCamera.setParameters(param);
            mCamera.setPreviewDisplay(mHolder);
            mCamera.startPreview();
            openSuc();
        } catch (Exception e) {
            openErr(e.getMessage());
            powerOff();
        }
    }
```

- 使用Camera2的API

关于Camera和Camera2之间的关系之前也已经提到过了，且使用Camera的API来开启闪关灯的确十分繁琐，但如果使用Camera2的API则会十分简洁，且不需要打开Camera，不需要考虑Camera资源的释放，并且不独占闪光灯资源，所以可以被其他需要Camera和闪光灯资源的App占用，使用方式：

```
    mCameraManager.setTorchMode(mFirstAvildId, true);//API23之后引入的
```

闪光灯状态的判断

在Android5.0以后的原生系统里，在手机的快速设置面板里多了手电筒功能（当然手机必须支持闪关灯功能才能看到），这也是新Camera2（android.hardware.Camera2）API开始引入的系统版本。并且能在闪光灯是否被占用的情况下，在快速设置面板里做出相应的调整，在摄像头被其他资源占用的时候，手电筒的快捷菜单消失或不可以点击，而Camera2的API中有一个这样的回调可以用来监听

```
    public static abstract class AvailabilityCallback{
      //A new camera has become available to use.
     public void onCameraAvailable(String cameraId){};
     //A previously-available camera has become unavailable for use.
     public void onCameraUnavailable(String cameraId) {};

    }
```

猜测是使用该回调来实现的，经过查看Android 5.0 `SystemUI` 的相关源代码也验证了这个猜测（附件有阅读 `SystemUI` 相关源码的分析笔记）。至于Android5.0以下，暂时并未找到可靠的API。