

ARIXO GLASS 开发说明 v2.0.0

Revision Record 修订记录

日期	修订版本	修改内容	作者
2018/10/08	0.1.0	初稿	徐雁
2019/3/18	0.1.1	G180 人脸检测 接口添加	赵鑫
2019/3/29	0.1.2	G180 人脸检测 功能暂不可用, 蓝牙集成权限申 请	赵鑫
2019/5/6	0.1.3	CameraClient, LcdClient 新增接 口添加	何开瑞

目录

1	简介.....	4
1.1	目的.....	4
1.2	范围.....	4
2	开发.....	4
2.1	开发语言.....	5
2.2	开发环境.....	5
3	功能介绍.....	5
3.1	Camera	5
3.1.1	分辨率.....	5
3.2	Audio.....	5
3.3	按键.....	6
4	API 说明	6
4.1	ArixoGlassSDKManager	6
	Glass SDK 的初始化类.....	6
4.2	DeviceClient.....	8
	USB 设备连接的管理类.....	8
4.3	CameraClient	10
	Camera 功能类.....	10
4.4	LCDClient	14
	LCD 屏管理类(仅支持 G180 系列).....	14

4.5	按键 Key.....	21
	BluetoothServiceConnection.....	21
5	集成.....	24
5.1	Glass SDK 集成.....	24
5.2	蓝牙 SDK 集成.....	26
6	示例代码.....	27
6.1	Glass SDK:.....	27
6.2	蓝牙 sdk:	31

1 简介

1.1 目的

本文档为使用 ARIXO GLASS 智能眼镜进行应用开发的人员提供基于 Android 平台的开发指南。

1.2 范围

基于 ARIXO GLASS 的 Android 应用开发。

2 开发

2.1 开发语言

Java / C++

2.2 开发环境

系统版本：5.0 以上

开发环境：Android Studio

3 功能介绍

3.1 Camera

3.1.1 分辨率

预览	1. "1920x1080" 2. "1280x720"
拍照	1. "1920x1080" 2. "1280x720"(默认)
录像	1. "1920x1080" 2. "1280x720"

3.2 Audio

ARIXO GLASS 支持 Bluetooth 4.2, 可以与手机直接连接, 遵循标准的 Android API。

3.3 按键

ARIXO GLASS 带有两个按键，一个是上方的按钮，一个是侧面的触摸板，按钮支持单击，双击操作，触摸板支持单击操作。

4 API 说明

4.1 ArixoGlassSDKManager

Glass SDK 的初始化类

ServiceInitListener: SDK Service 初始化状态回调

public void onInitStatus (boolean status);

说明：SDK 的所有 Service 初始化状态的回调

参数	类型	说明
status	boolean	是否初始化完成
返回值：	void	无

1. public static ArixoGlassSDKManager getInstance ();

说明：获取实例 Glass SDK 管理类实例对象

参数	类型	说明
无	无	无
返回值：	ArixoGlassSDKManager	Glass SDK 管理类

2. public void init (Context context, ServiceInitListener listener);

说明：Glass SDK 实例对象的初始化

参数	类型	说明
context	Context	上下文
listener	ServiceInitListener	Service 初始化状态回调接口
返回值：	void	无

3. public void destroy ();

说明：Glass SDK 实例对象销毁释放

参数	类型	说明
无	无	无
返回值：	void	无

4. public boolean isServiceConnected ();

说明：Service 链接状态

参数	类型	说明
无	无	无
返回值：	boolean	true 已链接, false 已断开

5. public CameraClient getCameraClient ();

说明：获取摄像头 Camera 操作类 CameraClient 实例

参数	类型	说明
无	无	无
返回值：	CameraClient	摄像头 Camera 操作类

6. public DeviceClient getDeviceClient ();

说明：获取设备 Device 操作类 DeviceClient 实例

参数	类型	说明
无	无	无
返回值：	DeviceClient	设备 Device 操作类

7. public LCDClient getLCDClient ();

说明：获取显示屏 LCD 操作类 LCDClient 实例

参数	类型	说明
无	无	无
返回值：	LCDClient	显示屏 LCD 操作类

8. public int getCameraType ();

说明：获取 Camera 设备的类型

参数	类型	说明
无	无	无
返回值：	int	CameraUtils.TYPE_USB_CAMERA

		USB 摄像头设备 CameraUtils.TYPE_USB_MOVIDIUS 搭载 Movidius 芯片设备
--	--	--

4.2 DeviceClient

USB 设备连接的管理类

DeviceConnectListener: Glass USB 设备连接状态的回调接口

public void onConnect (UsbDevice device);

说明：与设备建立连接成功

参数	类型	说明
device	UsbDevice	设备信息类
返回值：	void	无

public void onDisconnect (UsbDevice device);

说明：与设备连接断开

参数	类型	说明
device	UsbDevice	设备信息类
返回值：	void	无

public void onAttach (UsbDevice device);

说明：插入设备

参数	类型	说明
device	UsbDevice	设备信息类
返回值：	void	无

public void onDettach (UsbDevice device);

说明：拔出设备

参数	类型	说明
device	UsbDevice	设备信息类
返回值：	void	无

public void onCancel (UsbDevice device);

说明：设备连接取消

参数	类型	说明
----	----	----

device	UsbDevice	设备信息类
返回值:	void	无

1. public void registerDeviceListener (DeviceConnectListener deviceListener);

说明: 注册 USB 设备连接状态回调

参数	类型	说明
deviceListener	DeviceConnectListener	设备连接状态回调接口
返回值:	void	无

2. public void unregisterDeviceListener (DeviceConnectListener deviceListener);

说明: 注销 USB 设备连接的监听

参数	类型	说明
deviceListener	DeviceConnectListener	设备连接状态回调接口
返回值:	void	无

3. public boolean isConnected ();

说明: 设备是否连接

参数	类型	说明
无	无	无
返回值:	boolean	true 已连接, false 已断开

4. public UsbDevice getCurrentUsbDevice ();

说明: 获取当前已连接的 USB 设备

参数	类型	说明
无	无	无
返回值:	UsbDevice	设备信息类

5. public void release ();

说明: DeviceClient 实例的销毁释放

参数	类型	说明
无	无	无
返回值:	void	无

4.3 CameraClient

Camera 功能类

CameraClientCallback: Camera 摄像头状态回调接口

public void onCameraOpened ();

说明：打开 Camera 时回调

参数	类型	说明
无	无	无
返回值：	void	无

public void onCameraClosed ();

说明：关闭 Camera 时回调

参数	类型	说明
无	无	无
返回值：	void	无

PreviewFrameCallback: 摄像头帧数据回调接口

public void onFrameAvailable (byte[] data, int width, int height);

说明：视频帧数据回调

参数	类型	说明
data	byte[]	视频帧数据
width	int	视频宽度
height	int	视频高度
返回值：	void	无

1. public void open (int width, int height, CameraClientCallback clientCallback);

说明：打开摄像头

参数	类型	说明
width	int	视频宽度

height	int	视频高度
clientCallback	CameraClientCallback	Camera 连接状态监听
返回值:	void	无

2. public boolean isOpened ();

说明: 摄像头是否已经打开

参数	类型	说明
无	无	无
返回值:	boolean	true 打开, false 关闭

3. public void release ();

说明: CamerClient 实例销毁释放

参数	类型	说明
无	无	无
返回值:	void	无

4. public void disconnect ();

说明: 断开与 Camera 的连接

参数	类型	说明
无	无	无
返回值:	void	无

5. public void resize (int width, int height);

说明: 设置预览宽高(暂不支持动态切换分辨率)

参数	类型	说明
width	int	视频宽度
height	int	视频高度
返回值:	void	无

6. public void startRecording (String path, String fileName);

说明: 开始录像

参数	类型	说明
path	String	录像文件存放位置的绝对路径 (不包含文件名)

fileName	String	录像文件名(文件名后缀只能是.mp4, 如果不是会自动添加.mp4)
返回值:	void	无

7. public void startRecording (String path);

说明: 开始录像

参数	类型	说明
path	String	录像文件存放绝对路径(不包含文件名)
返回值:	void	无

8. public void stopRecording ();

说明: 停止录像

参数	类型	说明
无	无	无
返回值:	void	无

9. public boolean isRecording ();

说明: 是否正在录像

参数	类型	说明
无	无	无
返回值:	boolean	true 正在录像, false 未录像

10. public void addSurface (Surface surface);

说明: 添加预览 Surface

参数	类型	说明
surface	Surface	预览的 Surface 对象
返回值:	void	无

11. public void removeSurface (Surface surface);

说明: 删除预览 Surface

参数	类型	说明
surface	Surface	预览的 Surface 对象
返回值:	void	无

12. public void captureStill (String path);

说明：拍照

参数	类型	说明
path	String	图片存放绝对路径(含文件名)
返回值：	void	无

13. public Size getPreviewSize ();

说明：获取 Camera 当前预览尺寸

参数	类型	说明
无	无	无
返回值：	Size	Camera 当前预览尺寸(摄像头 open 成功后才能获取到值)

14. public List<Size> getSupportedPreviewSizes ();

说明：获取 Camera 支持的预览尺寸列表

参数	类型	说明
无	无	无
返回值：		Camera 支持的所有预览尺寸列表 (摄像头 open 成功后才能获取到值)

15. public int setCameraParameter (int flag, int value);

说明：设置 Camera 参数(**G180 系列暂不支持**)

参数	类型	说明
flag	int	Camera 参数项
value	int	Camera 参数值
返回值：	int	设置后的该参数项的值

16. public int getCameraParameter (int flag);

说明：获取 Camera 指定参数(**G180 系列暂不支持**)

参数	类型	说明
flag	int	Camera 参数项 详见 ICameraClient.java
返回值：	int	该参数项的值

17. public void setPreviewFrameCallback (PreviewFrameCallback callback);

说明：设置视频帧数据回调监听器

参数	类型	说明
callback	PreviewFrameCallback	视频帧数据回调接口
返回值：	void	无

4.4 LCDClient

LCD 屏管理类(仅支持 G180 系列)

LCDLightListener: LCD 亮度变化回调接口

public void onLightChanged (int light);

说明：取消设备连接(暂不支持)

参数	类型	说明
light	int	亮度
返回值：	void	无

SpecialModelListener: 特殊模型运行结果回调

public abstract void onModelSetup (boolean success);

说明：模型加载结果

参数	类型	说明
success	boolean	模型加载结果
返回值：	void	无

public void onFaceModelResult (FaceResultInfos faceResultInfos){};

说明：人脸检测结果

参数	类型	说明
faceResultInfos	FaceResultInfos	人脸检测结果

返回值:	void	无
------	------	---

NormalModelListener: 普通模型运行结果回调

void onModelSetup (boolean success);

说明: 模型加载结果

参数	类型	说明
success	boolean	模型加载结果
返回值:	void	无

void onNormalModelResult (NormalResultInfos normalResultInfos);

说明: 模型运行结果

参数	类型	说明
normalResultInfos	NormalResultInfos	模型运行结果
返回值:	void	无

1. public void startCaptureRecord (Context context);

说明: 开启同屏显示

参数	类型	说明
context	Context	上下文
返回值:	void	无

2. public void stopCaptureRecord ();

说明: 关闭同屏显示

参数	类型	说明
无	无	无
返回值:	void	无

3. public void setLCDLuminance (int luminance);

说明: 设置 LCD 亮度

参数	类型	说明
luminance	int	需要设置的亮度
返回值:	void	无

4. public int getLCDLuminance ();

说明：获取 LCD 亮度

参数	类型	说明
无	无	无
返回值：	int	当前 LCD 的亮度值

5. public void setLCDDisplayMode (int mode);

说明：设置 LCD 的显示模式

参数	类型	说明
mode	int	LCD 显示模式 ILCDClient. LCD_MODE_ASYNC (异屏) ILCDClient. LCD_MODE_SYNC (同屏)
返回值：	void	无

6. public int getLCDDisplayMode ();

说明：获取 LCD 的显示模式

参数	类型	说明
无	无	无
返回值：	int	当前 LCD 的显示模式

7. public void setLCDAutoBrightness (boolean autoSet);

说明：打开/关闭 LCD 亮度自动调节功能(暂不支持)

参数	类型	说明
autoSet	boolean	是否自动调节
返回值：	void	无

8. public boolean getLCDAutoBrightness ();

说明：获取 LCD 亮度自动调节功能状态(暂不支持)

参数	类型	说明
无	无	无
返回值：	boolean	true 打开 false 关闭

9. public boolean setFlashLightStatus (boolean status);

说明：打开闪光灯(暂不支持)

参数	类型	说明
status	boolean	打开或关闭闪光灯
返回值：	boolean	true 打开 false 关闭

10. public LCDInfo getLCDInfo ();

说明：获取 LCD 显示屏相关信息

参数	类型	说明
无	无	无
返回值：	LCDInfo	LCD 设备信息

11. public GlassInfo getGlassInfo ();

说明：获取 Glass 的设备信息

参数	类型	说明
无	无	无
返回值：	GlassInfo	Glass 设备信息

12. public void registerLCDLightListener (LCDLightListener listener);

说明：注册 LCD 亮度变化监听(暂不支持)

参数	类型	说明
listener	LCDLightListener	LCD 亮度变化监听
返回值：	void	无

13. public void unregisterLCDLightListener (LCDLightListener listener);

说明：注销 LCD 亮度变化监听(暂不支持)

参数	类型	说明
listener	LCDLightListener	LCD 亮度变化监听
返回值：	void	无

14. public boolean isScreenSyncing ();

说明：是否同屏显示中

参数	类型	说明
无	无	无

返回值:	boolean	true 为同屏, false 为异屏
------	---------	---------------------

15. public void setupSpacialModel (int type, int shvNum, String modelPath, SpecialModelListener listener);

说明: 加载特殊模型入口

参数	类型	说明
type	int	特殊模型类型: 1: 人脸检测(mobilenet-ssd) 2: 车牌检测(mobilenet-ssd)
shvNum	int	shave 使用数量
modelPath	String	模型文件绝对路径
listener	SpecialModelListener	模型运行结果回调
返回值:	void	无

16. public void startSpacialModel ();

说明: 开始运行模型

参数	类型	说明
无	无	无
返回值:	void	无

17. public void stopSpacialModel ();

说明: 停止运行模型

参数	类型	说明
无	无	无
返回值:	void	无

18. public void setupNormalModel (String graphName, String dimW, String dimH, String mean0, String mean1, String mean2, String std, String shvNum, String labelCount, String outFrame, String labelPath, String modelPath, NormalModelListener listener);

说明: 加载普通模型入口

参数	类型	说明
graphName	String	模型的名称 如: mobilenet-ssd
dimW	String	预处理图片的宽
dimH	String	预处理图片的高

mean0	String	预处理图片的均值化值 0~255
mean1	String	预处理图片的均值化值 0~255
mean2	String	预处理图片的均值化值 0~255
std	String	预处理图片的缩放因子
shvNum	String	运行模型时需要 shaves 的个数
labelCount	String	标签中类型的个数
outFrame	String	是否返回识别的图片帧，1 是返回，0 是不返回
labelPath	String	标签文件绝对路径
modelPath	String	模型文件绝对路径
listener	NormalModelListener	模型运行结果回调
返回值：	void	无

19. public void startNormalModel ();

说明：开始运行普通模型

参数	类型	说明
无	无	无
返回值：	void	无

20. public void stopNormalModel ();

说明：停止运行普通模型

参数	类型	说明
无	无	无
返回值：	void	无

21. public void sendCustomViewData (byte[] data, int alpha);

说明：发送自定义 View 数据到 LCD

参数	类型	说明
data	byte[]	通过获取到的显示屏的宽高为容器做背景，将需要显示的内容放在容器内，最后将整个容器截图为 RGBA 的 byte 数组(如果需要容器背景全透明，需要将背景色置为 0x00000000)
alpha	int	View 在显示屏上显示时的整体透明度(0~255)

返回值:	void	无
------	------	---

22. public void clearGlassCustomView ();

说明：清除 LCD 显示屏上显示的自定义 View 数据

参数	类型	说明
无	无	无
返回值:	void	无

23. public void writeMsgBoxToLCD (String boxInfo);

说明：发送画框的相关信息到 LCD 显示屏

参数	类型	说明
boxInfo	String	<p>boxInfo: json 格式的字符串，里面的信息是写在光机上的信息。</p> <p>举例：</p> <pre>{ "num":"2", "width":"1280", "height":"720", "infos": [{"class":"person", "box":"100,100,120,120", "boxcolor":"ff00ff", // BGR "boxline":"4" }, {"class":"car", "box":"150,150,170,170" "boxcolor":"ff00ff", "boxline":"4" }] }</pre> <p>num: 识别的个数 width: 识别源图片的宽</p>

		height: 识别源图片的高 infos: 识别的具体信息, 数组个数为 num class: 识别的类名称 box: 识别的物体坐标 x1,y1,x2,y2(left,top,right,bottom) boxcolor: 框的颜色, BGR 方式, 每个通道占 8 位 boxline: 框的线宽(单位是像素点)
返回值:	void	无

24. public void release ();

说明: LCDClient 实例的销毁释放

参数	类型	说明
无	无	无
返回值:	void	无

4.5 按键 Key

BluetoothServiceConnection

1. public static BluetoothServiceConnection getInstance ();

说明: 获取实例对象

参数	类型	说明
无	无	无
返回值:	BluetoothServiceConnection	BluetoothServiceConnection 实例对象

2. public void init (Context context);

说明: 初始化

参数	类型	说明
----	----	----

context	Context	上下文
返回值:	void	无

3. public BluetoothServiceConnection registerBluetoothDeviceConnectionListener (IBluetoothServiceCommunication.BluetoothDeviceConnectionListener deviceConnectionListener);

说明：注册蓝牙设备连接状态监听器

参数	类型	说明
deviceConnecti onListener	BluetoothDeviceConnecti onListener	蓝牙设备连接状态监听
返回值:	BluetoothServiceConnecti on	BluetoothServiceConnection 实 例对象

4. public BluetoothServiceConnection unregisterBluetoothDeviceConnectionListener (IBluetoothServiceCommunication.BluetoothDeviceConnectionListener deviceConnectionListener);

说明：注销蓝牙设备连接状态监听器

参数	类型	说明
deviceConnecti onListener	BluetoothDeviceConnecti onListener	蓝牙设备连接状态监听
返回值:	BluetoothServiceConnecti on	BluetoothServiceConnection 实 例对象

5. public BluetoothServiceConnection registerBluetoothEventListener (IBluetoothServiceCommunication.BluetoothEventListener eventListener);

说明：注册蓝牙按键键值监听器

参数	类型	说明
eventListener	BluetoothEventListener	蓝牙按键监听
返回值:	BluetoothServiceConnecti on	BluetoothServiceConnection 实 例对象

6. public BluetoothServiceConnection unregisterBluetoothEventListener (IBluetoothServiceCommunication.BluetoothEventListener eventListener);

说明：注销蓝牙按键键值监听器

参数	类型	说明
eventListener	BluetoothEventListener	蓝牙按键键值监听
返回值:	BluetoothServiceConnection	BluetoothServiceConnection 实例对象

7. public BluetoothServiceConnection registerBluetoothActionListener (BluetoothServiceCommunication.BluetoothActionListener actionListener);

说明：注册蓝牙动作监听器

参数	类型	说明
actionListener	BluetoothActionListener	蓝牙按键动作监听
返回值:	BluetoothServiceConnection	BluetoothServiceConnection 实例对象

8. public BluetoothServiceConnection unregisterBluetoothActionListener (BluetoothServiceCommunication.BluetoothActionListener actionListener);

说明：注销蓝牙动作监听器

参数	类型	说明
actionListener	BluetoothActionListener	蓝牙按键动作监听
返回值:	BluetoothServiceConnection	BluetoothServiceConnection 实例对象

9. public void startSearching ();

说明：开始扫描蓝牙设备

参数	类型	说明
无	无	无
返回值:	void	无

10. public void cancelSearch ();

说明：取消蓝牙扫描

参数	类型	说明
无	无	无
返回值:	void	无

11. public void connectDevice (String address);

说明: 连接蓝牙设备

参数	类型	说明
address	String	蓝牙地址
返回值:	void	无

12. public BluetoothDevice getCurrentDevice ();

说明: 获取当前连接的蓝牙设备

参数	类型	说明
无	无	无
返回值:	BluetoothDevice	BluetoothDevice 实例对象

13. public void clearCache ();

说明: 清除默认蓝牙设备缓存

参数	类型	说明
无	无	无
返回值: void	void	无

14. public void destroy ();

说明: 注销

参数	类型	说明
无	无	无
返回值:	void	无

5 集成

5.1 Glass SDK 集成

1、添加依赖

1.1、aar 集成方式

- 1、解压后的文件夹下，找到 sdk 文件夹，里面包含 ArixoGlassSDK*.aar 文件
- 2、在工程目录-app 下，新建 libs 文件夹，将 sdk 文件夹下的 aar 文件全部拷贝至 libs

3、在工程目录-app 下，打开 build.gradle 文件，在 dependencies 节点下加入，并点击同步项目 SyncProject

```
implementation fileTree(include: ['*.aar'], dir: 'libs')
```

1.2、gradle 集成方式

1、在项目的 gradle 文件中添加

```
allprojects {  
    repositories {  
        google()  
        jcenter()  
        maven { url "https://raw.githubusercontent.com/matrixz-cn/AriXoSDK/master" }  
    }  
}
```

2、工程 app 目录下 gradle 文件添加

```
implementation 'com.arixo:glasssdk:1.0'
```

2、注册相关服务

在工程目录-app 下 AndroidManifest.xml 中<Application>节点下加入：

```
<activity  
    android:name="com.arixo.glassframework.service.CaptureActivity"  
    android:process=":arixoservice"  
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />  
<service  
    android:name="com.arixo.glassframework.service.UVCService"  
    android:exported="true"  
    android:process=":arixoservice">  
    <intent-filter>  
        <action  
            android:name="com.arixo.glassframework.service.IUVCService">  
        </action>  
        <action  
            android:name="com.arixo.glassframework.service.IUVCService" />  
    </intent-filter>  
</service>  
<service
```

```
android:name="com.arixo.glassframework.service.DeviceConnectService"
    android:exported="true"
    android:process=":arixoservice">
        <intent-filter>
            <action
                android:name="com.arixo.glassframework.service.IDeviceC
                onnectService" />
        </intent-filter>
</service>
<service
    android:name="com.arixo.glassframework.service.LCDService"
    android:exported="true"
    android:process=":arixoservice">
        <intent-filter>
            <action
                android:name="com.arixo.glassframework.service.ILCDServi
                ce" />
        </intent-filter>
</service>
```

3、在工程目录-app 下 AndroidManifest 中添加如下权限：

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
// Android 9.0 需要添加 Camera 权限
<uses-permission android:name="android.permission.CAMERA" />
```

注意：Android 6.0 及其以上需动态申请 permission，请在适当的位置申请权限

5.2 蓝牙 SDK 集成

1. 添加蓝牙 aar 文件到 libs 目录下

2. AndroidManifest.xml 中添加以下权限

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission
    android:name="android.permission.BLUETOOTH_ADMIN" />
```

```
<uses-permission  
android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
<uses-permission  
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

3. AndroidManifest.xml 中<Application>节点下加入

```
<service  
    android:name="com.arixo.bluetooth.library.connection.BluetoothService"  
    android:exported="false" />
```

6 示例代码

6.1 Glass SDK:

1. 初始化，请在适当的时候做初始化操作

```
private boolean mInitStatus;  
private DeviceClient mDeviceClient;  
private CameraClient mCameraClient;  
private LCDClient mLCDClient;
```

```
ArixoGlassSDKManager.getInstance().init(mContext, mServiceInitListener);
```

```
ServiceInitListener mServiceInitListener = new ServiceInitListener {
```

```
    @Override
```

```
    public void onInitStatus(boolean status) {
```

```
        try {
```

```
            if (status) {
```

```
                mDebviceClient=ArixoGlassSDKManager.getInstance().getDevi  
ceClient();
```

```
                if (mDeviceClient != null) {
```

```
                    mDeviceClient.registerDeviceListener(mDeviceConnectListener  
);
```

```
                    mCameraClient=ArixoGlassSDKManager.getInsatance().getCa  
meraClient();
```

```
                    if(CameraUtils.TYPE_USB_CAMERA!=ArixoGlassSDKManager.g
```

```
etInstance().getCameraType()) {
    mLCDClient=ArixoGlassSDKManager.getInstance().getLCDClient();
}

    if (mCameraClient != null && mCameraClient.isOpened()) {
        openCamera();
    }
}

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

mInitStatus = status;
}
};
```

```
DeviceConnectListener mDeviceConnectListener = new DeviceConnectListener
{
    @Override
    public void onAttach(UsbDevice usbDevice) {
        if (mOnDeviceConnectListener != null) {
            mOnDeviceConnectListener.onAttach();
        }
    }

    @Override
    public void onDeAttach(UsbDevice usbDevice) {
        if (mOnDeviceConnectListener != null) {
            mOnDeviceConnectListener.onDeAttach();
        }
    }

    @Override
    public void onConnect(UsbDevice usbDevice) {
        if (mCameraClient != null) {
            openCamera();
        }
    }
}
```

```
        if (mOnDeviceConnectListener != null) {
            mOnDeviceConnectListener.onConnect();
        }
    } else {
        mCameraClient=ArixoGlassSDKManager.getInstance().getCameraClient();
        if (mCameraClient != null && mCameraClient.isOpened()) {
            openCamera();
            if (mOnDeviceConnectListener != null) {
                mOnDeviceConnectListener.onConnect();
            }
        }
    }
}

@Override
public void onDisconnect(UsbDevice usbDevice) {
    if (mCameraClient != null) {
        mCameraClient.disconnect();
        mCameraClient = null;
        if (mOnDeviceConnectListener != null) {
            mOnDeviceConnectListener.onDisconnect();
        }
    }
}

@Override
public void onCancel(UsbDevice usbDevice) {
    if (mOnDeviceConnectListener != null) {
        mOnDeviceConnectListener.onCancel();
    }
}

};

private void openCamera() {
    mCameraClient.open(mClientCallback);
}
```

```
mCameraClient.resize(videoWidth, videoHeight);
mCameraClient.connect();
mCameraClient.setPreviewFrameCallback(mPreviewFrameCallback);

}

CameraClientCallback mCameraClientCallback = new CameraClientCallback {

    @Override
    public void onCameraOpened() {
        if (mCameraClient != null) {
            mCameraClient.addSurface(mSurfaceView.getHolder.getSurface(),
                false);
        }
    }

    @Override
    public void onCamera() {
        if (mCameraClient != null) {
            mCameraClient.removeSurface(mSurfaceView.getHolder.getSurface());
        }
    }
};

PreviewFrameCallback mPreviewFrameCallback = new PreviewFrameCallback {

    @Override
    public void onFrameAvailable(byte[] data, int width, int height) {
        Log.d(TAG, String.valueOf(data.length));
    }
};
```

6.2 蓝牙 sdk:

```
        case IBluetoothServiceCommunication.EVENT_DOUBLE:
            break;
    }
    break;
case IBluetoothServiceCommunication.TOUCH_PAD:
    switch (event) {
        case IBluetoothServiceCommunication.EVENT_SHORT:
            break;
    }
}
};

private IBluetoothServiceCommunication.BluetoothActionListener
actionListener = action -> {
    switch (action) {
        case IBluetoothServiceCommunication.ACTION_UP:
            break;
        case IBluetoothServiceCommunication.ACTION_DOWN:
            break;
    }
};

public void initBluetoothService() {
    BluetoothServiceConnection.getInstance()
        .registerBluetoothDeviceConnectionListener(bluetooth
DeviceConnectionListener)
        .registerBluetoothActionListener(actionListener)
        .registerBluetoothEventListener(eventListener)
        .init(this);
}

public void unInitBluetoothService() {
    BluetoothServiceConnection.getInstance()
```



```
.unregisterBluetoothEventListener(eventListener)
.unregisterBluetoothActionListener(actionListener)
.unregisterBluetoothDeviceConnectionListener(bluetooth
DeviceConnectionListener)
    .destroy();
}
```