SDK 中提供了两个类 AlivcMediaRecorder 和 AlivcMediaRecorderFactory，其中AlivcMediaRecorder 是推流SDK使用接口类，AlivcMediaRecorderFactory 用来创建推流器器 AlivcMediaRecorder。同时提供了多个事件通知接口，用来监听推流器的各种状态。

类名	功能
AlivcMediaRecorderFactory	创建推流器接口
AlivcMediaRecorder	推流功能接口类
AlivcMediaFormat	推流器可配参数类
OnNetworkStatusListener	推流状态监听接口
OnRecordStatusListener	预览状态监听接口
OnLiveRecordErrorListener	推流错误信息监听接口

AlivcMediaRecorderFactory

功能：

创建推流器接口

AlivcMediaRecorder

成员	功能
createMediaRecorder	创建推流器 AlivcMediaRecorder

详细说明：

AlivcMediaRecorderFactory.createMediaRecorder()

createMediaRecorder 用来创建推流器，返回 AlivcMediaRecorder 类。

返回值：

返回空为错误，正确则为有效的 AlivcMediaRecorder 值。

AlivcMediaRecorder

功能：

推流器接口类

AlivcMediaRecorder，提供推流控制

成员	功能
init	初始化推流器
prepare	开始预览
startRecord	开始推流
switchCamera	切换摄像头
stopRecord	结束推流
reset	释放预览资源，对应的是 prepare
focusing	对焦
setZoom	缩放
setPreviewSize	设置预览大小
addFlag	添加美颜
removeFlag	移除美颜
release	释放资源对应: init
setOnRecordErrorListener	设置推流错误回调
setOnRecordStatusListener	设置推流的状态回调监听
setOnNetworkStatusListener	设置网络状态的回调监听
getVersionName	获取SDK版本名称

下面详细介绍一下各个成员函数的具体使用：

- 初始化推流器

```
/**
* @param context：Android上下文
*/
void init(Context context);
```

- 开始预览

```
/**
* 开始预览
* @param  params：推流过程中不可动态改变的参数.
* @param  surface：预览窗口
*/
void prepare(Map<String, Object> params, Surface surface);
```

- 开始推流

```
/**
* @param outputUrl 推流地址URL
*/
void startRecord(String outputUrl);
```

> 备注：在prepare完成之后调用startRecord进行推流。

- 结束推流

```
void stopRecord();
```

- 切换摄像头

```
//Demo默认为前置摄像头
void switchCamera();
```

- 释放资源

```
void reset();
```

> 备注：*释放预览资源,对应的是prepare*

- 手动对焦

```
/**
* @param xRatio 横向坐标点所占的比例
* @param yRatio 纵向坐标点所占的比例
*/
void focusing(float xRatio, float yRatio);
```

- 缩放

```
/**
* 这是个新增加的接口，老接口为下面的接口，已经废弃，不建议使用
*
* @param scaleFactor参数为缩放比例
*/
void setZoom(float scaleFactor);
```

```
/**
* 这个接口已经废弃了，建议使用上面的接口
*
* @param scaleFactor参数为缩放比例
*/
@Deceperated
void setZoom(float scaleFactor,CaptureRequest.OnCaptureRequestResultListener l
istener);
```

- 设置预览大小

```
/**
* @param width 预览宽
* @param height 预览高
*/
void setPreviewSize(int width, int height);
```

- 开启美颜 / 开启手动对焦 / 开启闪光灯 / 开启静音推流

```
/**
* 增加效果 如：美颜，对焦，闪光灯，静音等
* @see AlivcMediaFormat#FLAG_BEAUTY_ON
* @see AlivcMediaFormat#FLAG_AUTO_FOCUS_ON
* @see AlivcMediaFormat#FLAG_FLASH_MODE_ON
* @see AlivcMediaFormat#FLAG_MUTE_ON
* @see AlivcMediaRecorder#removeFlag(int)
* @param flag
*/
void addFlag(int flag);
```

- 关闭美颜 / 关闭手动对焦 / 关闭闪光灯 / 关闭静音推流

```
/**
* 移除效果 如：美颜，对焦，闪光灯，静音等
* @see AlivcMediaFormat#FLAG_BEAUTY_ON
* @see AlivcMediaFormat#FLAG_AUTO_FOCUS_ON
* @see AlivcMediaFormat#FLAG_FLASH_MODE_ON
* @see AlivcMediaFormat#FLAG_MUTE_ON
* @param flag
*/
void removeFlag(int flag);
```

- 释放资源

```
void release();
```

- 设置推流错误回调

```
/**
* @param listener 直播错误回调
*/
void setOnRecordErrorListener(OnLiveRecordErrorListener listener);
```

> 备注：*具体的错误信息会在下面介绍*

- 设置推流的状态回调监听

```
/**
* @param listener 直播状态回调
*/
void setOnRecordStatusListener(OnRecordStatusListener listener);
```

> 备注：*回调具体的信息在下面介绍*

- 设置网络状态的回调监听

```
/**
* @param listener 直播网络状态回调
*/
void setOnNetworkStatusListener(OnNetworkStatusListener listener);
```

> 备注：*回调具体的信息在下面介绍*

- 添加水印图片

首先需要创建一个水印信息对像——AlivcWatermark

```
AlivcWatermark mWatermark = new AlivcWatermark.Builder()
                .watermarkUrl(bundle.getString(WATERMARK_PATH))  //水印图片地址
                .paddingX(bundle.getInt(WATERMARK_DX))           //水印图片在x轴
的偏移
                .paddingY(bundle.getInt(WATERMARK_DY))           //水印图片在y轴
的偏移
                .site(bundle.getInt(WATERMARK_SITE))             //水印图片位置
                .build();
```

水印位置有四个常量表示：

常量值	含义
AlivcWatermark.SITE*TOP*LEFT	左上角
AlivcWatermark.SITE*TOP*RIGHT	右上角
AlivcWatermark.SITE*BOTTOM*LEFT	左下角
AlivcWatermark.SITE*BOTTOM*RIGHT	右下角

然后需要将该水印信息描述对像通过 AlivcMediaRecorder#prepare 接口的 map 参数传入进去

```
mConfigure.put(AlivcMediaFormat.KEY_WATERMARK, mWatermark); //配置水印信息

mMediaRecorder.prepare(mConfigure, mPreviewSurface);
```

- 获取SDK版本名称

```
String getVersionName()
```

AlivcMediaFormat

在开始推流前需要调用 prepare(Map params, Surface surface); 其中的 map 参数的 Key 值指定即为 AlivcMediaFormat 的字段。以下为 AlivcMediaFormat 的字段概要：

限定符和类型	字段	说明
static int	CAMERA*FACING*BACK	后置摄像头
static int	CAMERA*FACING*FRONT	前置摄像头
static int	DISPLAY*ROTATION*0	旋转角度 0 度
static int	DISPLAY*ROTATION*90	旋转角度 90 度

static int	DISPLAY*ROTATION*180	旋转角度 180 度
static int	DISPLAY*ROTATION*270	旋转角度 270 度
static int	FALG*FALSH*MODE_ON	已过时
static int	FLAG*FLASH*MODE_ON	开启闪光灯
static int	FLAG*AUTO*FOCUS_ON	自动对焦开启
static int	FLAG*BEAUTY*ON	美颜开启
static int	FLAG*BLUR*ON	模糊开启
static int	FLAG*MUTE*ON	静音开启
static int	KEY*AUDIO*BITRATE	音频码率（建议设置为 32000)
static int	KEY*AUDIO*SAMPLE_RATE	音频采样率（建议设置为 44100）
static int	KEY*BEST*VIDEO_BITRATE	最优码率 （单位：bps）
static int	KEY*CAMERA*FACING	摄像头方向，Value: CAMERA*FACING*BACK CAMERA*FACING*FRONT
static int	KEY*DISPLAY*ROTATION	旋转角度值，Value: DISPLAY*ROTATION*0 DISPLAY*ROTATION*90 DISPLAY*ROTATION*180 DISPLAY*ROTATION*270
static int	KEY*EXPOSURE*COMPENSATION	曝光度 设定范围：[0，100] -1 或者不设表示自动曝光
static int	KEY*FRAME*RATE	帧率
static int	KEY*I*FRAME_INTERNAL	关键帧间隔
static int	KEY*INITIAL*VIDEO_BITRATE	初始码率 （单位：bps）
static int	KEY*MAX*VIDEO_BITRATE	最大码率 （单位：bps）
static int	KEY*MAX*ZOOM_LEVEL	最大缩放级别 ＞0
static int	KEY*MIN*VIDEO_BITRATE	最小码率 （单位：bps）
		输出分辨率 value: OUTPUT*RESOLUTION*240P

static int	KEY*OUTPUT*RESOLUTION	OUTPUT*RESOLUTION*360P OUTPUT*RESOLUTION*480P OUTPUT*RESOLUTION*540P OUTPUT*RESOLUTION*720P OUTPUT*RESOLUTION*1080P
static int	KEY*RECONNECT*TIMEOUT	自动重连超时，默认是 5000ms
static int	KEY_WATERMARK	水印配置需要构造 AlivcWatermark。Builder 这个结构体
static int	OUTPUT*RESOLUTION*240P	推流输出分辨率：240P
static int	OUTPUT*RESOLUTION*360P	推流输出分辨率：360P
static int	OUTPUT*RESOLUTION*480P	推流输出分辨率：480P
static int	OUTPUT*RESOLUTION*540P	推流输出分辨率：540P
static int	OUTPUT*RESOLUTION*720P	推流输出分辨率：720P
static int	OUTPUT*RESOLUTION*1080P	推流输出分辨率：1080P
static int	BEAUTY*LEVEL*ONE	美颜级别 1，一共 7 个级别，级别越高，美颜程度越深
static int	BEAUTY*LEVEL*TWO	美颜级别 2，一共 7 个级别，级别越高，美颜程度越深
static int	BEAUTY*LEVEL*THREE	美颜级别 3，一共 7 个级别，级别越高，美颜程度越深
static int	BEAUTY*LEVEL*FOUR	美颜级别 4，一共 7 个级别，级别越高，美颜程度越深
static int	BEAUTY*LEVEL*FIVE	美颜级别 5，一共 7 个级别，级别越高，美颜程度越深
static int	BEAUTY*LEVEL*SIX	美颜级别 6，一共 7 个级别，级别越高，美颜程度越深
static int	BEAUTY*LEVEL*SEVEN	美颜级别 7，一共 7 个级别，级别越高，美颜程度越深

OnNetworkStatusListener

当调用 startRecord() 后，推流器开始推流，当前的网络连接状态会返回，用户需要注册该事件，以便获取到该时间通知，在结束和开始的状态都会给予返回。

```
public interface OnNetworkStatusListener {

  /**
   * 网络较差时的回调，此时推流buffer为满的状态，会执行丢包，此时数据流不能正常推送
   */
  void onNetworkBusy();

  /**
   * 网络空闲状态，此时本地推流buffer不满，数据流可正常发送
   */
  void onNetworkFree();

  /**
   * @param   status
   */
  void onConnectionStatusChange(int status);

  /**
   * 重连失败
   * @return false:停止重连 true:继续重连
   * 说明： s d k 检测到检测到需要重连时将会自动执行重连，直到重连成功或者重连尝试超时，
   * 超时时间可以通过{@link AlivcMediaFormat#KEY_RECONNECT_TIMEOUT}来设置，
   * 默认为 5 s，超时后将触发此回调，若返回true表示继续开始新的一轮尝试，返回false，
   * 表示不再尝试
   */
  boolean onNetworkReconnectFailed();
}
```

注意：这里 *onNetworkReconnect()* 接口已经移除掉，现在网络重连为 sdk 自动执行，在重连指定时间后弱依然不能重连成功，则会执行 *onNetworkReconnectFailed()* 的回调，表示重连失败。

OnRecordStatusListener

当 prepare 完成后，会发出该事件通知消息，用户需要注册该事件，采集开始得到摄像头的状态以便做一些事情，比如进入对焦一次。

```
public interface OnRecordStatusListener {
    /*** 摄像头打开成功   */
    void onDeviceAttach();
    /*** 开启预览成功   */
    void onSessionAttach();
    /**   * 停止预览   */
    void onSessionDetach();
    /*** 关闭摄像头   */
    void onDeviceDetach();
    /**摄像头打开失败**/
    void onDeviceAttachFailed(int facing)
}
```

OnLiveRecordErrorListener

当开始推流，返回给开发者的错误信息。

```
public interface OnLiveRecordErrorListener {
    /**
     * @param  errorCode 获取到的错误类型Code
     * @see AlivcStatusCode#ERROR_**
     */
    void onError(int errorCode);
}
```

> 备注：具体 ErrorCode 详情可参考下面的 ErrorCode 说明

ErrorCode说明

ErrorCode常量	数值	含义	可能出现的场景
AlivcStatusCode.ERROR*BROKEN*PIPE	-32	管道中断	推流时进行了违法操作，比如同时推流同一个地址，或者重复推流，服务器端会主动关闭 socket，引起 broken pipe

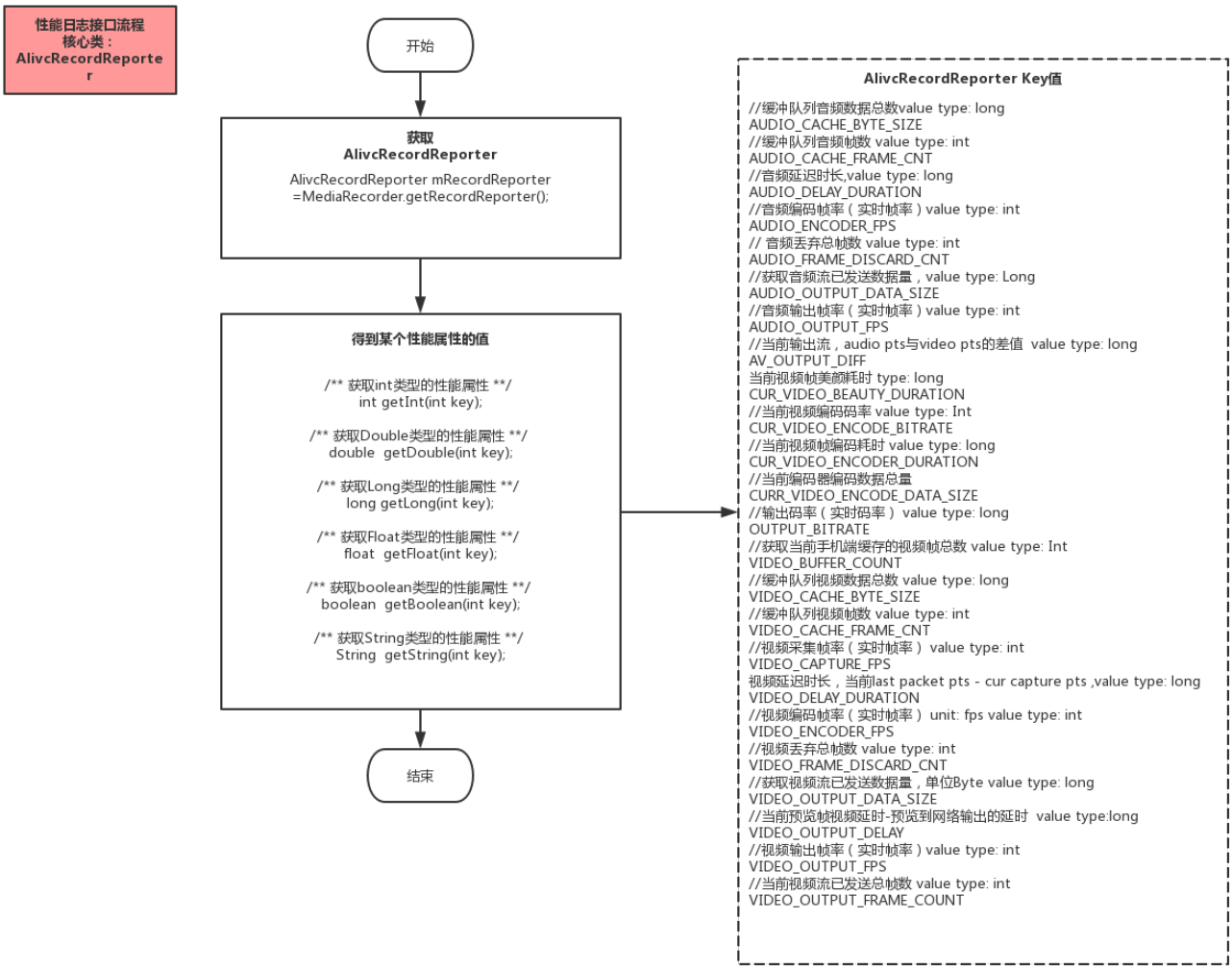
AlivcStatusCode.ERORR*OUT*OF_MEMORY	-12	内存不足	手机内存不足时导致底层某些内存开辟失败引起该错误
AlivcStatusCode.ERROR_IO	-5	I/O错误	导致该错误的情况比较多，比如网络环境较差或者推流域名错误等导致 DNS 解析失败等
AlivcStatusCode.ERROR*ILLEGAL*ARGUMENT	-22	参数非法	该错误通常发生在帧数据错误的情况下
AlivcStatusCode.ERROR*NETWORK*UNREACHABLE	-101	网络不可达	该错误通常发生在网络无法传输数据的情况，或者推流过程网络中断等情况
AlivcStatusCode.ERROR*SERVER*CLOSED_CONNECTION	-104	服务器关闭链接	发生违法操作时，服务器会主动断开链接
AlivcStatusCode.ERROR*CONNECTION*TIMEOUT	-110	网络链接超时	网络较差时导致链接超时或者数据发送超时
		鉴	

AlivcStatusCode.ERROR*AUTH*FAILED	-1313558101	权失败	推流地址开启鉴权时，鉴权失败
AlivcStatusCode.ERROR*OPERATION*NOT_PERMITTED	-1	操作不允许	发生违法操作时，服务器端主动断开链接导致
AlivcStatusCode.ERROR*CONNECTION*REFUSED	-111	服务器拒绝链接	域名解析错误，或者其他异常导致无法链接服务器时

打点日志

- 获取性能日志

  所有的性能日志都通过 AlivcRecordReporter 来获取,通过AlivcMediaRecorder#getRecordReporter() 获取 AlivcRecordReporter 的对象。具体接口调用流程见下图：



注意：AlivcMediaRecorder#getRecordReporter() 必须在AlivcMediaRecordReporter#init() 方法之后，AlivcMediaRecordReporter#release() 方法之前调用，否则将返回一个 null 对象

/** 获取int类型的性能属性 **/ AlivcRecordReporter#getInt(int key);

/** 获取Double类型的性能属性 **/ AlivcRecordReporter#getDouble(int key);

/** 获取Long类型的性能属性 **/ AlivcRecordReporter#getLong(int key);

/** 获取Float类型的性能属性 **/ AlivcRecordReporter#getFloat(int key);

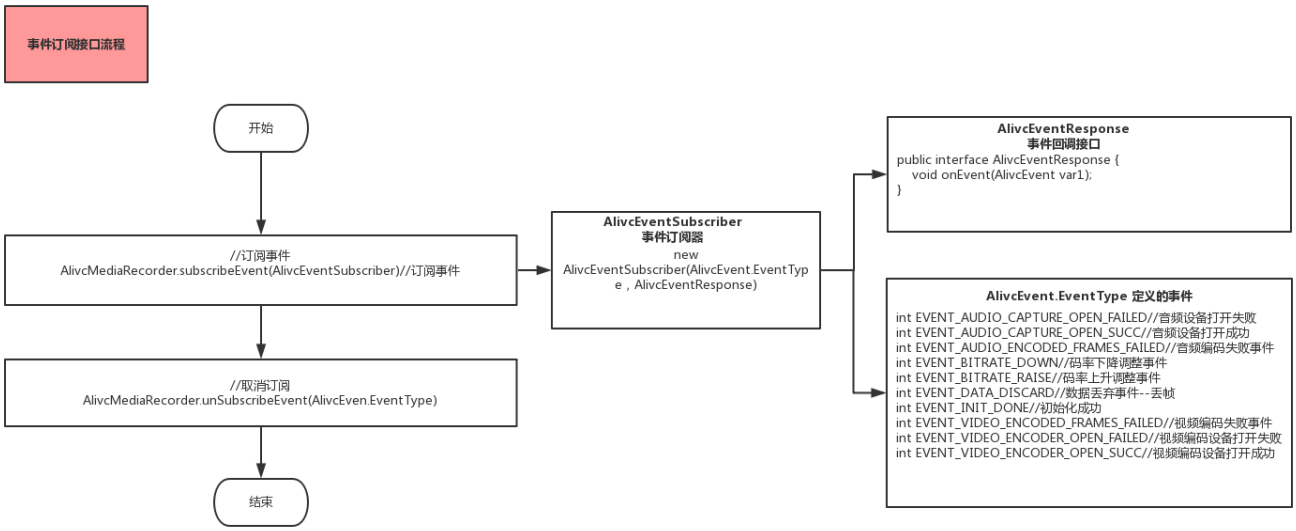
/** 获取boolean类型的性能属性 **/ AlivcRecordReporter#getBoolean(int key);

/** 获取String类型的性能属性 **/ AlivcRecordReporter#getString(int key);

相关属性的 Key 值都可通过 AlivcRecordReporter 拿到，比如视频采集帧率的 Key 值——AlivcRecordReporter#VIDEO*CAPTURE*FPS，其对应的属性值类型可在相应的类文档中查看。

- 事件通知

打点事件是通过订阅的形式获取监听，通过 AlivcMediaRecorder#subscribeEvent() 订阅一个事件，通过 AlivcMediaRecorder#unSubscribeEvent() 取消订阅。具体接口调用流程见下图：



订阅事件：

需要创建一个 AlivcEventSubscriber 对象，该对象中包含要订阅的事件类型以及对应的事件响应，事件类型可以通过 AlivcEvent#EventType 类中定义的常量来表示。

事件响应需要实现 AlivcEventResponse 接口，其中的 onEvent 表示事件发生，其参数 AlivcEvent 代表一个事件，AlivcEvent#getBundle() 方法获取一个 Bundle 对象，该对象存储了事件的一些相关信息，具体每个事件对应有哪些信息可以查看相关类文档，获取事件信息的 Key 值可以通过 AlivcEvent$EventBundleKey 中定义的常量表示。