一、版本历史
二、SDK概要
2.1 SDK集成
2.2 SDK架构
2.2.1 类结构
2.2.2 接口说明
三、SDK初始化配置
3.1 初始化配置
3.2 CarLife初始化接口说明
3.2.1 方法说明
3.2.2 参数说明
init方法
参数说明
返回值
receiver方法
参数说明
返回值
3.2.3 features及configs说明
features
configs
3.3 VoiceManager初始化说明
3.4 显示分辨率的设置说明
参数说明
示例
3.5 设置渲染Surface
参数说明
3.5.1 SurfaceView
3.5.2 OnVideoSizeChangedListener
概要
Public methods
注册监听
3.6 读取本地配置
bdcf文件内容如下
3.7 Configs
四、连接
4.1 获取本地文件配置
4.2 执行连接
4.3 监听连接状态
4.3.1 ConnectionChangeListener
概要
Public methods
注册监听
4.3.2 TransportListener
概要
Public methods
注册监听
示例
4.4 连接进度

4.4.1 ConnectProgressListener
4.4.2 进度定义
4.5 连接异常处理
五、车机端功能定制
六、订阅消息
6.1 CarLifeSubscribable -- 发布者
概要
Public methods
Members
6.1.1 车机端可以发布的信息
车身信息模块ID
6.1.2 手机端可以发布的消息
手机端信息模块ID
6.1.3 示例
实现车身GPS模块信息订阅
使用CarLife.receiver()添加
6.2 CarLifeSubscriber -- 订阅者
概要
Public methods
Members
6.2.1 示例
手机端返回的导航简易诱导信息
使用CarLife.receiver()添加
七、MIC录音规则
7.1 支持唤醒的车机MIC规则
7.2 不支持唤醒的车机MIC规则
7.3 手机MIC录音
八、CarLifeModule
8.1 模块类型
8.2 状态处理
8.2.1 CarLifeModule
概要
Public methods
Members
示例
8.2.2 ModuleStateChangeListener
概要
Public methods
8.3 SDK实现
九、CarLife上下文
9.1 CarLifeContext
9.2 CarLifeReceiver
十、其他
10.1 SDK日志路径

一、版本历史

版本号	修改内容简介	修改日期	修改人
V1.0	创建文档	2021-08-23	wangchangming

二、SDK概要

2.1 SDK集成

- 集成aar

需要将carlife-sdk-release文件复制到Application Module/libs文件夹（没有的需手动创建），并将以下代码添加到您app的build.gradle中:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
depedencies {
    compile(name: 'carlife-sdk-release', ext: 'aar')
}
```

- 添加依赖

在主Module的build.gradle文件添加sdk的依赖

```
dependencies {
    implementation 'com.google.protobuf:protobuf-lite:3.0.1'
}
```

- AndroidManifest配置

```
<!--必要权限-->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- 运行环境配置

本SDK需运行在Android5.0(API Level 21)及以上版本。

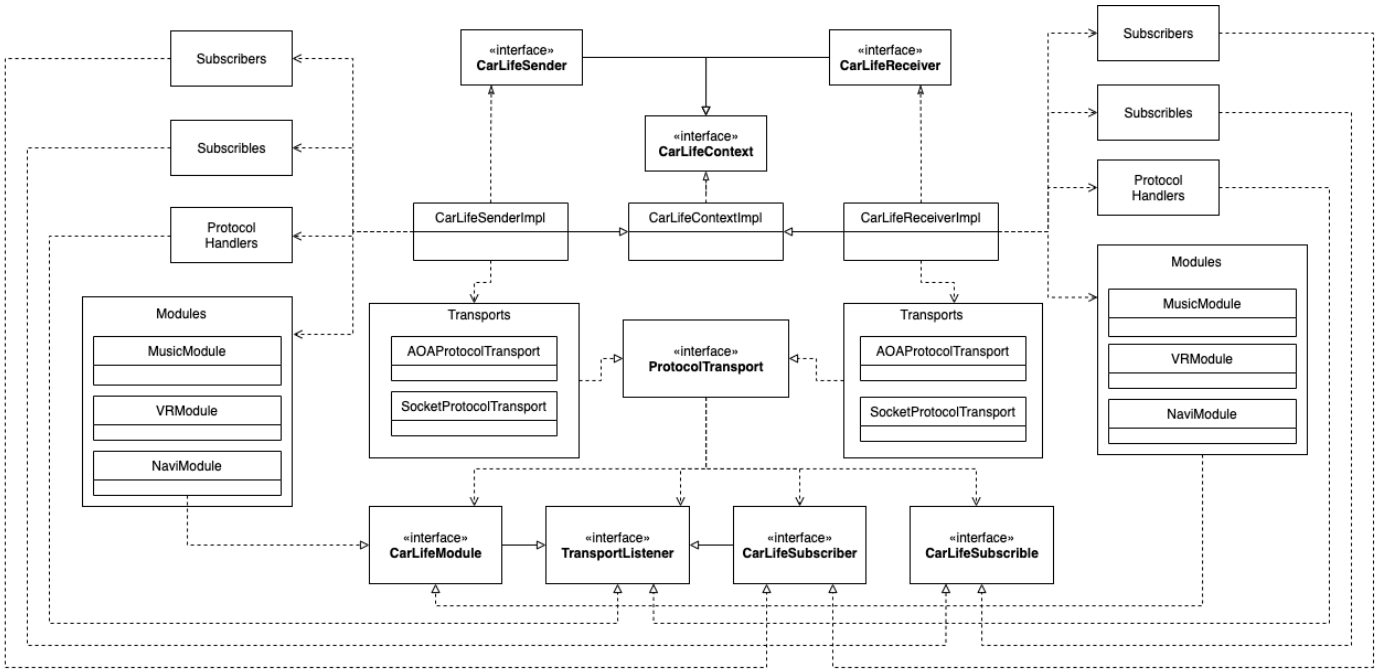
- 代码混淆

如果你需要使用proguard混淆代码，需确保不要混淆SDK的代码。请在proguard-rules.pro文件(或其他混淆文件)尾部添加如下配置。

```
-keep class com.baidu.carlife.** {*;}
-dontwarn com.baidu.carlife.**
```

2.2 SDK架构

2.2.1 类结构



SDK 的里面封装了车机端及手机端相关的连接投屏相关流程，以及消息订阅，模块管理等内容，减少了车厂接入车机端开发时的工作量。其中的CarLifeReceiver代表车机端，其实现者为CarLifeReceiverImpl，封装了订阅、模块管理、连接投屏相关的操作流程。

2.2.2 接口说明

这里将简单介绍车机端开发时涉及到的重要接口，有的SDK已实现，有的还需要车机端自己拓展功能，下面将简单介绍一下相关概念，如需车厂实现部分将在后面部分进行详解，这里仅简单了解相关概念。

- CarLifeContext

CarLife上下文，通用方法封装，通过CarLifeContext可以发送消息、监听状态、加解密等等。

- CarLifeModule

CarLife本身具有模块的概念，并定义了导航、音乐、语音等几个模块，模块本身具备id（唯一表示）、state（状态）。当状态发生变化时，要实时同步给车机。CarLifeContext内部封装了具体实现，使模块实现者只关注业务实现，无需关系协议交互。

- CarLifeSubscriber及CarLifeSubscrible

CarLife中有消息订阅机制，即车机可以跟手机订阅消息（路口信息、红绿灯信息等），手机也可以跟车机订阅消息（GPS、车速等）。有相关的协议交互来完成。CarLifeContext封装了相关协议交互，对外只暴露接口。

- CarLifeReceiver

CarLifeReceiver车机端抽象接口，具备车机端的特有能力，反控事件发送、语音收音、音频播放与焦点管理等。

- CarLifeSender

CarLifeSender手机端抽象接口，具备手机端的特有能力，虚拟屏幕创建、反控事件处理、硬按键处理、文件传输相关。

注：车厂接入时无需关心CarLifeSender，此仅与手机端有关

- ProtocolTransport

ProtocolTransport负责建立连接相关，包括AOA连接、WIFI连接、Wi-Fi直连相关的连接，此类业务操作相关在SDK中完成，里面有完善的断开重练机制，也有连接过程中各个状态的回调，车机端连接只需调用CarLife.receiver().connect()即可。

三、SDK初始化配置

3.1 初始化配置

> 重点：
>
> 1. CarLife SDK 需要在主线程中初始化；
> 2. 初始化时需要传入相关配置。

开发者需要在Application#onCreate()方法中调用以下代码来初始化CarLife SDK 。

```
class VehicleApplication : Application() {

    var vehicleBind: VehicleService.VehicleBind? = null

    override fun onCreate() {
        super.onCreate()
        CarlifeConfUtil.getInstance().init()
        initReceiver()
    }

    @SuppressLint("ClickableViewAccessibility")
    private fun initReceiver() {
        val screenWidth = resources.displayMetrics.widthPixels
        val screenHeight = resources.displayMetrics.heightPixels
        val displaySpec = DisplaySpec(
            this,
            screenWidth.coerceAtLeast(screenHeight).coerceAtMost(1920),
            screenWidth.coerceAtMost(screenHeight).coerceAtMost(1080),
            30
        )

        // 车机端支持特性，更多配置参照Configs.FEATURE_*相关
```

```
        val features = mapOf(
            FEATURE_CONFIG_USB_MTU to 16 * 1024,
            FEATURE_CONFIG_I_FRAME_INTERVAL to 300
        )

        // 初始化配置，更多配置参照Configs.CONFIG_*相关
        val configs = mapOf(
            CONFIG_LOG_LEVEL to Log.DEBUG,
            CONFIG_CONTENT_ENCRYPTION to true,
            CONFIG_USE_ASYNC_USB_MODE to false,
            CONFIG_PROTOCOL_VERSION to 3
        )

        Log.d(Constants.TAG, "VehicleApplication initReceiver $screenWidth,
$screenHeight $displaySpec")
        CarLife.init(
            this,
            "20029999",
            "12345678",
            features,
            CarlifeActivity::class.java, // 当车机端开始接收投屏数据之前，会拉起此Activity
            configs)
        VoiceManager.init(CarLife.receiver())
        CarLife.receiver().setDisplaySpec(displaySpec)
    }
}
```

3.2 CarLife初始化接口说明

```
CarLife.init(
            this,
            "20029999",
            "12345678",
            features,
            CarlifeActivity::class.java,
            configs)
```

CarLife类里面有两个静态方法，init和receiver()，还有一个私有成员变量receiver。其中init方法用于初始化receiver的，此方法必须在主线程中调用；receiver是用于返回receiver的值。这里初始化完成后，**CarLife.receiver()** 的返回就不为空，后续就可以使用了。

3.2.1 方法说明

方法名	方法介绍
init(Context context, String channel, String cuid, Map<String, Int> features, Class activityClass, Map<String, Any> configs)	用于初始化CarLife SDK，无返回值
receiver()	返回CarLife SDK的上下文对象

3.2.2 参数说明

init方法

参数说明

参数	含义
context	Context: Application上下文对象
channel	String: 车机渠道号，用于手机端验证车机端的合法性
cuid	String: 车机端设备唯一标识符
features	Map<String, Int>: 车机端支持的功能集合。（如车机端是否支持语音唤醒） 一般设置一些固定的feature，无需从配置文件读取的。从配置文件读取的另有API接口设置
activityClass	Class: 车机端App的主Activity
configs	Map<String, Any>: 车机端相关配置集合。（如车机端Wi-Fi直连的名称、是否加密等）

返回值

Returns	
Void	无返回值

receiver方法

参数说明

参数	含义
无参数	

返回值

Returns	
CarLifeReceiver	CarLife SDK的上下文对象，用于sdk的相关操作

3.2.3 features及configs说明

features及configs这里只是传入的一些配置，后面都可以使用**CarLife.receiver()** 调用相应方法进行修改。其中key值在Configs中已定义完毕，直接使用即可。

features

Key	Value(Int)	含义
Configs.FEATURE_CONFIG_USB_MTU	16 * 1024	USB连接时，车机端指定的最大包的size。 注：可不指定，主要为了处理在某些车机端，包太大导致发送失败
Configs.FEATURE_CONFIG_FOCUS_UI	1	定制焦点态。0: 触控 1:焦点 2:焦点+触控 默认触控
Configs.FEATURE_CONFIG_I_FRAME_INTERVAL	300	关键帧间隔 默认为1
Configs.FEATURE_CONFIG_VOICE_MIC	1	语音MIC。 0表示使用车机MIC，1表示使用手机MIC，2表示车机端MIC不支持， 3表示车机端支持回声消噪，排列方式是先左声道\右声道， 4表示车机端支持回声消噪，排列方式是先右声道\左声道。 默认使用车机MIC
Configs.FEATURE_CONFIG_VOICE_WAKEUP	1	车机是否支持语音唤醒。 0:不支持 1:支持 默认支持唤醒
Configs.FEATURE_CONFIG_BLUETOOTH_AUTO_PAIR	1	定制蓝牙自动匹配。0:不匹配 1:匹配 默认不自动匹配。 注：目前android手机获取不到蓝牙地址。只有部分合作手机能够获取，如果获取到了才自动匹配
Configs.FEATURE_CONFIG_BLUETOOTH_INTERNAL_UI	0	定制蓝牙电话。0:不可用 1:可用 默认不可用
Configs.FEATURE_CONFIG_MEDIA_SAMPLE_RATE	0	媒体采样率，定制MD传入HU的PCM流的采样率 0:跟随手机系统 1:48k 默认跟随手机系统，一般是其值是44.1k
Configs.FEATURE_CONFIG_AUDIO_TRANSMISSION_MODE	0	定制MD向HU传输音频流方式 0: 专用音频通道 1:蓝牙通道 默认为使用专用音频通道进行音频传输
Configs.FEATURE_CONFIG_ENGINE_TYPE	0	定制运行CarLife的车机类型 0:燃油车 1:电动车 默认为燃油车
Configs.FEATURE_CONFIG_INPUT_DISABLE	0	是否根据车速屏蔽输入法 0:是 1:否 默认是只要HU发过来车速过来就启用屏蔽

注：这里的功能定制最后会通过MSG_CMD_HU_FEATURE_CONFIG_RESPONSE协议传递给手机端。

configs

Key	Value(Object)	含义
Configs.CONFIG_LOG_LEVEL	Log.DEBUG	sdk里面的日志级别 **注：只在sdk初始化时配置生效**
Configs.CONFIG_SAVE_AUDIO_FILE	"MI8"	车机端wifi直连的名称
Configs.CONFIG_CONTENT_ENCRYPTION	true	是否加密
Configs.CONFIG_USE_ASYNC_USB_MODE	false	是否使用同步 默认为false，Android4.3及以下需设置为true
Configs.CONFIG_PROTOCOL_VERSION	3	车机端协议版本 手机端会验证此版本，当手机CarLife版本小于车机端版本时，会连接不成功 **注：只在SDK初始化时配置生效**
Configs.CONFIG_HU_BT_NAME	hu_name	车机端蓝牙名称
Configs.CONFIG_HU_BT_MAC	xxx-xxx-xxx	车机端蓝牙MAC地址
....		
....		

3.3 VoiceManager初始化说明

```
VoiceManager.init(CarLife.receiver())
```

初始化录音相关，启动录音线程，参数为**CarLife.receiver()** 。车机MIC录音相关操作在SDK中处理，无需车厂管理。

3.4 显示分辨率的设置说明

```
CarLife.receiver().setDisplaySpec(displaySpec)
```

车机端发送给手机端支持的分辨率及帧率

参数说明

参数	含义
displaySpec	DisplaySpec：存储显示宽、高及分辨率的类

```
package com.baidu.carlife.sdk.sender.display

import android.content.Context
import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
data class DisplaySpec (val context: Context, val width: Int, val height: Int, val
frameRate: Int) {
    val densityDpi: Int = context.resources.displayMetrics.densityDpi

    val ratio: Float = width.toFloat() / height
}
```

注：车机端设置分辨率建议1080P，最低应该为720P。

示例

```
// 视频编码参数
val displaySpec = DisplaySpec(
    this,
    1920,
    1080
    30)
```

3.5 设置渲染Surface

```
CarLife.receiver().setSurface(mSurface)
```

参数说明

参数	含义
mSurface	Surface： 用于渲染投屏数据。使用sdk中提供的SurfaceView获得。

3.5.1 SurfaceView

SDK里面提供了两种SurfaceView：RemoteDisplayGLView 和 RemoteDisplayView，其中RemoteDisplayGLView可用与Android 5.0以上高版本，Android5.0以下低版本需使用 RemoteDisplayView。车厂接入时可选择其中一种使用。

```
class RemoteDisplayGLView @JvmOverloads constructor(context: Context, attrs:
AttributeSet? = null):GLSurfaceView(context, attrs)
class RemoteDisplayView @JvmOverloads constructor(context: Context, attrs:
AttributeSet? = null):SurfaceView(context, attrs)
```

示例

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/root_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CarlifeActivity">

    <com.baidu.carlife.sdk.receiver.view.RemoteDisplayGLView
        android:id="@+id/video_surface_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/root_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CarlifeActivity">

    <com.baidu.carlife.sdk.receiver.view.RemoteDisplayView
        android:id="@+id/video_surface_view"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

注：SurfaceView的宽高可以自定义，最后显示效果会根据手机端传输过来的分辨率进行计算显示在SurfaceView上

```
    override fun onVideoSizeChanged(videoWidth: Int, videoHeight: Int) {
        post {
            val ratio = videoWidth.toFloat() / videoHeight
            val (destWidth, destHeight) = ScaleUtils.inside(measuredWidth,
measuredHeight, ratio)
            layoutParams = layoutParams.apply {
                width = destWidth
                height = destHeight
            }
        }
    }
```

3.5.2 OnVideoSizeChangedListener

实现OnVideoSizeChangedListener接口并注册可以监听视频分辨率的更改。

```
package com.baidu.carlife.sdk.receiver;

import com.baidu.carlife.sdk.util.annotations.DoNotStrip;

@DoNotStrip
public interface OnVideoSizeChangedListener {
    void onVideoSizeChanged(int width, int height);
}
```

概要

Public methods

返回值	方法
void	onVideoSizeChanged(int width, int height) 手机端传输的视频分辨率发生改变时调用

注册监听

```
CarLife.receiver().addOnVideoSizeChangedListener(***)
```

3.6 读取本地配置

```
CarlifeConfUtil.getInstance().init()
```

这里是读取本地**/data/local/tmp/bdcf** 的配置文件，文件里面配置了每个车机端独有的配置，如车机渠道号等。在开始连接前需确保此配置文件读取成功完毕。

bdcf文件内容如下

```
#Channel Id: Baidu will provide the channel id to all OEM
20352101

# Audio Track Type 0:普通双音轨(路畅、华阳、飞? 1:单音?掌讯,仅用于测? 2:定制双音?1(BYD、远?
3:定制双音?2（长安） 4：百度定制双音轨
AUDIO_TRACK_TYPE = 0

# 是否使用车机端MiC 0:使用车机端mic 1:使用手机端mic 2:不支持
VOICE_MIC = 0

# 是否支持唤醒
VOICE_WAKEUP = 1

# 是否支持Carlife内置蓝牙电话
BLUETOOTH_INTERNAL_UI = 0

# 是否支持蓝牙自动匹配
BLUETOOTH_AUTO_PAIR = 0

# Focus Ui
FOCUS_UI = 1

# 音频采样 0:跟随手机系统 1:48k
MEDIA_SAMPLE_RATE = 0

# Android手机的连接类 0:ADB 1:AOA
CONNECT_TYPE_ANDROID = 1

# Iphone手机的连接类 0:NCM 1:Wifi 2:NCM&Wifi(会显示切换的开关)
CONNECT_TYPE_IPHONE = 1

# iPhone手机通过USB的连接方式 0：IPV6  1：IPV4  2：USB共享
IPHONE_USB_CONNECT_TYPE = 0
```

```
# iPhone手机通过USB—NCM连接映射在车机端网卡的名称
IPHONE_NCM_ETHERNET_NAME = usb0

#Next song
#KEYCODE_SEEK_ADD = 23

#Previous song
#KEYCODE_SEEK_SUB = 22

#定制MD向HU传输音频流方向 0：专用独立通道传输 1：蓝牙
AUDIO_TRANSMISSION_MODE=0

# 车机端支持的无线连接类型 0：不支持  1：热点  2：直连  3：都支持
CONFIG_WIRLESS_TYPE = 3

# 车机端支持的无线连接频率 0：2.4G  1：5G
CONFIG_WIRLESS_FREQUENCY = 1

# 是否支持蓝牙音频
CONFIG_BLUETOOTH_AUDIO = false

# 车机端wifi直连的名字
CONFIG_WIFI_DIRECT_NAME = MI8

# 手机端蓝牙名称
CONFIG_TARGET_BLUETOOTH_NAME = 1234MIX

# 车机端蓝牙名称
CONFIG_HU_BT_NAME = xxx

# 车机端蓝牙MAC地址
CONFIG_HU_BT_MAC = yyy-zzz-xxx

# 是否保存音频源文件
CONFIG_SAVE_AUDIO_FILE = false
```

这里的Feature定制读取成功后，可以使用如下两个API接口设置给SDK：

```
CarLife.receiver().setFeatures(features: Map<String, Int>)
CarLife.receiver().setFeature(key: String, feature: Int)
```

注：设置的Feature如需同步给手机端，需在连接前完成。

3.7 Configs

```
public class Configs {
    // 本地设置参数

    // 日志文件尺寸上限，超过了创建新的
    public static final String CONFIG_MAX_LOG_FILE_SIZE = "CONFIG_MAX_LOG_FILE_SIZE";
    // 日志文件数量上限，超过了删除老的
    public static final String CONFIG_MAX_LOG_FILE_COUNT = "CONFIG_MAX_LOG_FILE_COUNT";
    // 打印日志级别
    public static final String CONFIG_LOG_LEVEL = "CONFIG_LOG_LEVEL";
    // 是否对传输的数据进行加密
    public static final String CONFIG_CONTENT_ENCRYPTION = "CONFIG_CONTENT_ENCRYPTION";

    // statistics info，receiver初始化时需要传入，必须传入
    public static final String CONFIG_STATISTICS_INFO = "CONFIG_STATISTICS_INFO";

    // 是否锁定屏幕
    public static final String CONFIG_LOCK_ORIENTATION =
"ScreenCaptureDisplay_LOCK_ORIENTATION";
    // 是否保存音频数据到文件
    public static final String CONFIG_SAVE_AUDIO_FILE = "CONFIG_SAVE_AUDIO_FILE";
    // 是否保存视频数据到文件
    public static final String CONFIG_SAVE_VIDEO_FILE = "CONFIG_SAVE_VIDEO_FILE";

    // 是否开启全局投屏
    public static final String CONFIG_ENABLE_SYSTEM_SCREEN_CAPTURE =
"CONFIG_ENABLE_SYSTEM_SCREEN_CAPTURE";

    // 是否是沙盒环境
    public static final String CONFIG_USE_SANDBOX = "CONFIG_USE_SANDBOX";

    //音频是否Carlife通道播放
    public static final String CONFIG_USE_REMOTE_PLAY = "CONFIG_USE_REMOTE_PLAY";

    // 车机端设置参数
    // 是否实用Mic收音
    public static final String FEATURE_CONFIG_VOICE_MIC = "VOICE_MIC";
    // 是否支持自动唤醒
    public static final String FEATURE_CONFIG_VOICE_WAKEUP = "VOICE_WAKEUP";
    // 是否支持蓝牙自动配对
    public static final String FEATURE_CONFIG_BLUETOOTH_AUTO_PAIR =
"BLUETOOTH_AUTO_PAIR";
    // 是否支持蓝牙电话UI
    public static final String FEATURE_CONFIG_BLUETOOTH_INTERNAL_UI =
"BLUETOOTH_INTERNAL_UI";
    // 是否是焦点态车机
    public static final String FEATURE_CONFIG_FOCUS_UI = "FOCUS_UI";
    // 是否加密内容
    public static final String FEATURE_CONFIG_CONTENT_ENCRYPTION =
"CONTENT_ENCRYPTION";
```

```
    // 支持的连接类型
    public static final String FEATURE_CONFIG_CONNECT_TYPE = "CONNECT_TYPE";
    // 是否支持多指触控
    public static final String FEATURE_CONFIG_MULTI_TOUCH = "MULTI_TOUCH";
    // 是否开启蓝牙音频
    public static final String FEATURE_CONFIG_BT_AUDIO = "BLUETOOTH_AUDIO";
}
```

四、连接

4.1 获取本地文件配置

车机端开始连接前，可以读取一下本地的配置文件，把相应数据存储在缓存中（如：车机渠道号、车机wifi直连的名称、wifi的type类型、是否支持蓝牙音频等），读取完毕后，可以使用sdk上下文更改相应的设置，如下代码：

```
    fun init() {
        Logger.e(TAG, "++++++++++++++++++++Baidu Carlife Begin++++++++++++++++++++")

        // 根据配置文件设置相应的配置
        CarLife.receiver().setConfig(Configs.CONFIG_WIFI_DIRECT_NAME, "MI8")
        CarLife.receiver().setConfig(Configs.CONFIG_TARGET_BLUETOOTH_NAME, "1234MIX")
        CarLife.receiver().initStatisticsInfo(CommonParams.VEHICLE_CHANNEL, "12345678")
        CarLife.receiver().setConfig(Configs.CONFIG_WIRLESS_TYPE,
CarlifeConfUtil.getInstance().getIntProperty(Configs.CONFIG_WIRLESS_TYPE))
        CarLife.receiver().setConfig(Configs.CONFIG_WIRLESS_FREQUENCY,
CarlifeConfUtil.getInstance().getIntProperty(Configs.CONFIG_WIRLESS_FREQUENCY))
        CarLife.receiver().setConfig(Configs.CONFIG_USE_BT_AUDIO,
CarlifeConfUtil.getInstance().getBooleanProperty(Configs.CONFIG_USE_BT_AUDIO))

    }
```

4.2 执行连接

配置文件设置完毕后，就可以执行连接过程了。

```
CarLife.receiver().connect();
```

注：连接过程中需要通过协议通道传递给手机端的配置都需要在连接前设置完成。

4.3 监听连接状态

4.3.1 ConnectionChangeListener

```
package com.baidu.carlife.sdk

import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
interface ConnectionChangeListener {
    /**
     * 连接建立，分下面三种情况
     * 1 AOA连接建立
     * 2 ADB六条Socket建立成功
     * 3 Wifi六条Socket建立成功
     */
    @JvmDefault
    fun onConnectionAttached(context: CarLifeContext) {}

    /**
     * 在未detach的前提下，重新进行attach操作，会回调reattach
     */
    @JvmDefault
    fun onConnectionReattached(context: CarLifeContext) { }

    /**
     * 连接断开
     * 1 USB断开
     * 2 网络断开
     */
    @JvmDefault
    fun onConnectionDetached(context: CarLifeContext) {}

    /**
     * 协议校验成功
     * statistics info 校验成功时
     */
    @JvmDefault
    fun onConnectionEstablished(context: CarLifeContext) {}

    /**
     * 与设备协议版本不兼容
     * ProtocolVersionMatch 校验失败时
     */
    @JvmDefault
    fun onConnectionVersionNotSupprt(context: CarLifeContext) {}

    /**
```

```
     * 车机渠道号验证失败
     * CarlifeAuthenResult 校验失败时
     */
    @JvmDefault
    fun onConnectionAuthenFailed(context: CarLifeContext) {}
}
```

连接状态可以实现ConnectionChangeListener接口并注册来监听

概要

Public methods

返回值	方法
void	onConnectionAttached(context: CarLifeContext) 连接建立，分为三种情况：AOA连接建立、ADB六条Socket建立成功、wifi六条Socket建立成功
void	onConnectionReattached(context: CarLifeContext) 在未detach的前提下，重新进行attach操作，会回调reattach
void	onConnectionDetached(context: CarLifeContext) 连接断开：USB断开、网络断开
void	onConnectionEstablished(context: CarLifeContext) 协议校验成功时会回调此方法
void	onConnectionVersionNotSupprt(context: CarLifeContext) 与手机端CarLife 协议版本不兼容时会回调此方法
void	onConnectionAuthenFailed(context: CarLifeContext) 车机渠道号不合法时会回调此方法

注册监听

```
CarLife.receiver().registerConnectionChangeListener(this)
```

4.3.2 TransportListener

```
package com.baidu.carlife.sdk.internal.transport

import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.ConnectionChangeListener
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.util.annotations.DoNotStrip
```

```
@DoNotStrip
interface TransportListener: ConnectionChangeListener {
    /**
     * called by transport when send a message
     * Avoid time-consuming operations
     * @return 是否继续停止分发
     */
    @JvmDefault
    fun onSendMessage(context: CarLifeContext, message: CarLifeMessage): Boolean {
        return false
    }

    /**
     * called by transport when receive a message
     * Avoid time-consuming operations
     * @return 是否继续停止分发
     */
    @JvmDefault
    fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage): Boolean {
        return false
    }
}
```

TransportListener继承了ConnectionChangeListener类，同时新增了收发协议的监听方法。如需要同时监听连接状态及收发协议的类可直接实现此接口并通过**CarLife.receiver()** 完成注册即可。

概要

Public methods

返回值	方法
Boolean	onSendMessage(context: CarLifeContext, message: CarLifeMessage) 车机端发送的每一条协议消息都可以在此监听
Boolean	onReceiveMessage(context: CarLifeContext, message: CarLifeMessage) 车机端收到的每一条协议消息都可以在此监听

注册监听

```
CarLife.receiver().registerTransportListener(this)
```

示例

在onReceiveMessage方法监听连接异常消息

```
    override fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage):
Boolean {
        when(message.serviceType) {
            ServiceTypes.MSG_CMD_CONNECT_EXCEPTION -> {
                val response= message.protoPayload as CarlifeConnectException
                handleConnectException(response)
            }
        }
        return false
    }
```

4.4 连接进度

连接进度可实现ConnectProgressListener接口并完成注册，可监听，进度值由SDK来控制。

4.4.1 ConnectProgressListener

```
package com.baidu.carlife.sdk.receiver

interface ConnectProgressListener {
    /**
     * 当前连接进度
     * @param progress 0 -> 100
     */
    fun onProgress(progress: Int)
}
```

注册监听

```
CarLife.receiver().addConnectProgressListener(this)
```

4.4.2 进度定义

进度值	含义
0	onConnectionAttached方法触发时 此时车机端开始发送MSG_CMD_HU_PROTOCOL_VERSION
30	收到手机端发送的MSG_CMD_PROTOCOL_VERSION_MATCH_STATUS消息，并且协议版本匹配正确 协议不匹配时触发onConnectionVersionNotSupprt方法，需提示升级手机端CarLife
70	onConnectionEstablished方法触发时 此处证明车机渠道号合法，当不合法时，触发onConnectionAuthenFailed方法，需提示该车机为非法车机
100	车机端发出MSG_CMD_VIDEO_ENCODER_START消息时，可认为连接成功 接下来只需要手机端发送投屏数据即可

4.5 连接异常处理

连接过程中，可能会出现以下几个问题：

- 手机端CarLife 版本未升级，导致与车机端协议版本不匹配，此时会回调onConnectionVersionNotSupprt方法，车机端可提示用户升级手机端CarLife后再进行连接操作；
- 车机端渠道号不合法，在手机端会验证车机端发送过来的渠道号，当不合法时，此时会回调onConnectionAuthenFailed方法，车机端可提示用户当前车机为非法车机；
- 当手机端为安装CarLife时，AOA连接时在手机会有系统框提示用户，但是车机端的现象时发送MSG_CMD_HU_PROTOCOL_VERSION协议一直没有响应，进度会一直在0%显示，此时这里车机端app可设置超时机制；
- 当CarLife手机端与车机端连接成功后，断开的情况下，此时会延时1秒后自动再次进行连接，由sdk自行处理。
- 其他。如有遇到，再补充。

五、车机端功能定制

车机端的定制相关功能的支持情况是通过发送MSG_CMD_HU_FEATURE_CONFIG_RESPONSE消息并携带消息给到手机端的，这部分在SDK里面处理了，车机端app只需要在初始化CarLife前或者连接前通过设置相应的feature即可。

- CarLife的init方法执行前配置，然后传给init方法的features参数，如下：

```
val features = mapOf(
    FEATURE_CONFIG_USB_MTU to 16 * 1024,
    FEATURE_CONFIG_FOCUS_UI to 1,
    FEATURE_CONFIG_I_FRAME_INTERVAL to 300
    FEATURE_CONFIG_VOICE_MIC to 1,
```

```
            FEATURE_CONFIG_VOICE_WAKEUP to 1
        )

        CarLife.init(
            ...,
            ...,
            ...,
            features,
            ...,
            ...)
```

- 开始连接前通过CarLife.receiver()设置feature参数，代码如下：

```
CarLife.receiver().setFeature(key: String, feature: Int)
CarLife.receiver().setFeatures(features: Map<String, Int>)
```

六、订阅消息

CarLife中有消息订阅的机制，订阅相关消息涉及到手机端订阅车机端的消息、车机端订阅手机端的消息。车机端与手机端互为订阅者与发布者。**由相关的协议交互来完成，CarLifeContext封装了相关协议交互，对外只暴露接口。**

6.1 CarLifeSubscribable -- 发布者

手机端可以获取车载相关数据，包括车速、GPS定位信息、陀螺仪加速信息、油耗等。此时车机端可以称为发布者；当车机端向手机端订阅导航相关的辅助信息时，则此时手机端可以称为发布者。

SDK里面已定义好接口，提供信息的类只需实现该接口，并把实现了该发布者类通过**CarLife.receiver()** 添加即可。

```
// 被订阅者
interface CarLifeSubscribable {
    // 消息id
    val id: Int

    // 订阅者开始订阅，被订阅者应该在此方法中开启数据线程，定期发给订阅者
    fun subscribe()

    // 订阅者停止订阅，被订阅者应该在此方法中停止数据线程，停止发给订阅者
    fun unsubscribe()
}
```

概要

Public methods

返回值	方法
void	subscribe() 在此开始发送数据
void	unsubscribe() 在此停止发送数据

Members

成员	含义
Id	Int: 信息模块ID，当前发布者提供的信息类型
supportFlag	Boolean: 是否支持订阅

6.1.1 车机端可以发布的信息

车身信息模块ID

```
enum ModuleID
{
    CAR_DATA_GPS=0;
    CAR_DATA_VELOCITY=1;
    CAR_DATA_GYROSCOPE=2;
    CAR_DATA_ACCELERATION=3;
    CAR_DATA_GEAR=4;
    CAR_DATA_OIL=5;
}
```

名称	含义
CAR_DATA_GPS	车身GPS模块
CAR_DATA_VELOCITY	车身速度模块
CAR_DATA_GYROSCOPE	车身陀螺仪模块
CAR_DATA_ACCELERATION	车身加速度模块
CAR_DATA_GEAR	车身档位模块
CAR_DATA_OIL	车身油量模块

注：见文档149页

6.1.2 手机端可以发布的消息

车机端需要使用CarLife里的导航模块提供的HUD信息或者电子眼信息时，可发起订阅。

手机端信息模块ID

```
enum ModuleID
{
    CARLIFE_DATA_TURNBYTURN=0;
    CARLFIE_DATA_ASSISTANTGUIDE=1;
}
```

名称	含义
CARLIFE_DATA_TURNBYTURN	CARLIFE_DATA_TURNBYTURN是简易诱导信息
CARLFIE_DATA_ASSISTANTGUIDE	CARLFIE_DATA_ASSISTANTGUIDE是辅助诱导信息

注：见152页

6.1.3 示例

实现车身GPS模块信息订阅

```
package com.baidu.carlifevehicle

import com.baidu.carlife.protobuf.CarlifeCarGpsProto
import com.baidu.carlife.protobuf.CarlifeNaviAssitantGuideInfoProto
import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.CarLifeSubscribable
import com.baidu.carlife.sdk.Constants
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.internal.protocol.ServiceTypes
import com.baidu.carlife.sdk.sender.CarLife
import com.baidu.carlife.sdk.util.TimerUtils

class CarDataGPSSubscribable(private val context: CarLifeContext): CarLifeSubscribable
{
    override val id: Int = 0
    override var SupportFlag: Boolean = true

    override fun subscribe() {
        //开始给手机端发送GPS信息
        TimerUtils.schedule(subscribeGpsData, 1 * 1000, 2 * 1000)
    }
```

```
    override fun unsubscribe() {
        //停止给手机端发送GPS信息
        TimerUtils.stop(subscribeGpsData)
    }

    private val subscribeGpsData = Runnable {
        //开始给手机端发送当前汽车位置信息
        var message = CarLifeMessage.obtain(Constants.MSG_CHANNEL_CMD,
ServiceTypes.MSG_CMD_CAR_GPS)
        message.payload(
            CarlifeCarGpsProto.CarlifeCarGps.newBuilder()
                .setAntennaState(1)
                .setSignalQuality(2)
                .setLatitude(20)
                .setLongitude(10)
                .setHeight(1000)
                .setSpeed(80)
                .setHeading(90)
                .setYear(2021)
                .setMonth(7)
                .setDay(2)
                .setHrs(18)
                .setMin(30)
                .setSec(1)
                .setFix(1)
                .setHdop(2)
                .setPdop(1)
                .setVdop(3)
                .setSatsUsed(2)
                .setSatsVisible(2)
                .setHorPosError(4)
                .setVertPosError(5)
                .setNorthSpeed(6)
                .setEastSpeed(7)
                .setVertSpeed(8)
                .setTimeStamp(System.currentTimeMillis())
                .build())
        context.postMessage(message)
    }
}
```

注：各字段信息见文档144页

使用CarLife.receiver()添加

```
CarLife.receiver().addSubscribable(CarDataGPSSubscribable(CarLife.receiver()))
```

6.2 CarLifeSubscriber -- 订阅者

手机端向车机端订阅油耗等相关信息时，手机端为订阅者；车机端向手机端订阅导航相关简易诱导信息时，车机端为订阅者。

SDK里面已定义好接口，订阅者只需实现订阅接口接口，并把实现了该订阅者类通过**CarLife.receiver()** 添加即可。

```
// 订阅者
interface CarLifeSubscriber: TransportListener {
    // 消息id
    val id: Int

    // 被订阅者返回事件消息、可以获取此消息是否支持订阅
    fun process(context: CarLifeContext, info: Any)


    // 如果成功订阅了，通过此接口获取相关的消息信息
    fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage) {}
}
```

概要

Public methods

返回值	方法
void	process(context: CarLifeContext, info: Any) 处理订阅消息，如油耗信息

Members

成员	含义
id	Int: 信息模块ID 注：这里的信息模块ID同发布者是一样的

6.2.1 示例

手机端返回的导航简易诱导信息

```
package com.baidu.carlifevehicle

import android.os.Handler
import android.os.Looper
import com.baidu.carlife.protobuf.CarlifeNaviAssitantGuideInfoProto
import com.baidu.carlife.protobuf.CarlifeSubscribeMobileCarLifeInfoListProto
import com.baidu.carlife.protobuf.CarlifeSubscribeMobileCarLifeInfoProto
import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.CarLifeSubscriber
import com.baidu.carlife.sdk.Constants
import com.baidu.carlife.sdk.internal.protocol.CarLifeMessage
import com.baidu.carlife.sdk.internal.protocol.ServiceTypes
import com.baidu.carlife.sdk.util.Logger

class AssistantGuideSubscriber(private val context: CarLifeContext) : CarLifeSubscriber
{
    private val TAG = "AssistantGuideSubscriber"

    override val id: Int = 1

    private var handler = Handler(Looper.getMainLooper())

    override fun process(context: CarLifeContext, info: Any) {
        var carLifeInfo = info as
CarlifeSubscribeMobileCarLifeInfoProto.CarlifeSubscribeMobileCarLifeInfo

        if (carLifeInfo.supportFlag) {
            snedCarlifeDataSubscribe(true)


            handler.postDelayed(Runnable {
                snedCarlifeDataSubscribe(false)
            }, 30 * 1000)
        }
    }

    override fun onReceiveMessage(context: CarLifeContext, message: CarLifeMessage):
Boolean {
        when (message.serviceType) {
            ServiceTypes.MSG_CMD_NAV_ASSISTANT_GUIDE_INFO ->
handleCarlifeDataSubscribeInfo(context, message)
        }

        return false
    }
```

```
    private fun snedCarlifeDataSubscribe(isStart: Boolean) {
        val infoListBuilder =

CarlifeSubscribeMobileCarLifeInfoListProto.CarlifeSubscribeMobileCarLifeInfoList.newBui
lder()
        val infoBuilder =
CarlifeSubscribeMobileCarLifeInfoProto.CarlifeSubscribeMobileCarLifeInfo.newBuilder()
        infoBuilder.moduleID = id
        infoBuilder.supportFlag = isStart
        infoListBuilder.addSubscribemobileCarLifeInfo(infoBuilder.build())

        infoListBuilder.cnt = infoListBuilder.subscribemobileCarLifeInfoCount

        val response = CarLifeMessage.obtain(
            Constants.MSG_CHANNEL_CMD,
            if (isStart) ServiceTypes.MSG_CMD_CARLIFE_DATA_SUBSCRIBE_START else
ServiceTypes.MSG_CMD_CARLIFE_DATA_SUBSCRIBE_STOP
        )
        response.payload(infoListBuilder.build())
        context.postMessage(response)
    }


    private fun handleCarlifeDataSubscribeInfo(context: CarLifeContext, message:
CarLifeMessage) {
        var response = message.protoPayload as?
CarlifeNaviAssitantGuideInfoProto.CarlifeNaviAssitantGuideInfo
        //开始处理移动设备Carlife发送过来的电子狗信息
        Logger.d(TAG, response?.toString())
    }
}
```

使用CarLife.receiver()添加

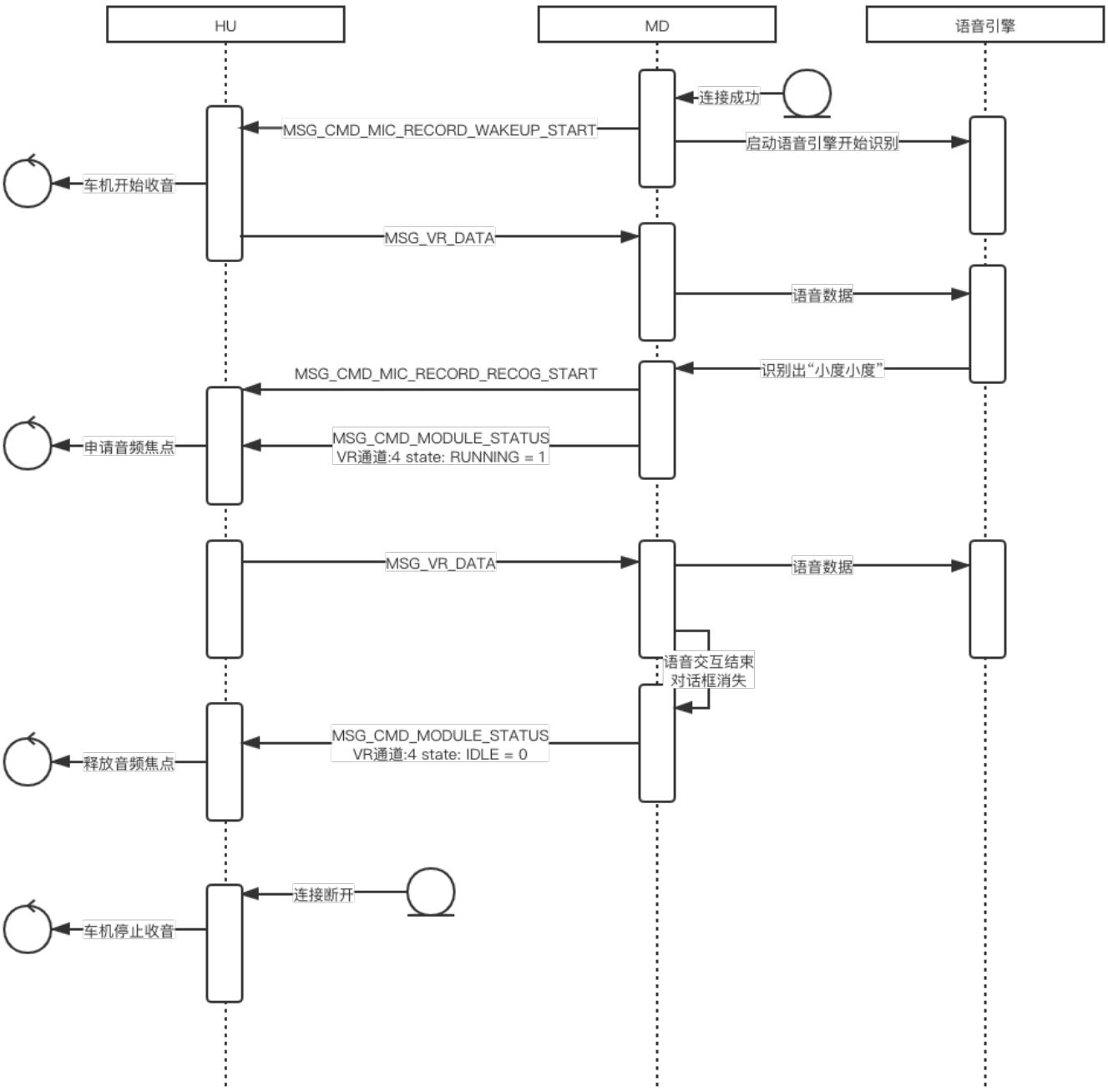
```
CarLife.receiver().addSubscriber(AssistantGuideSubscriber(CarLife.receiver()))
```

七、MIC录音规则

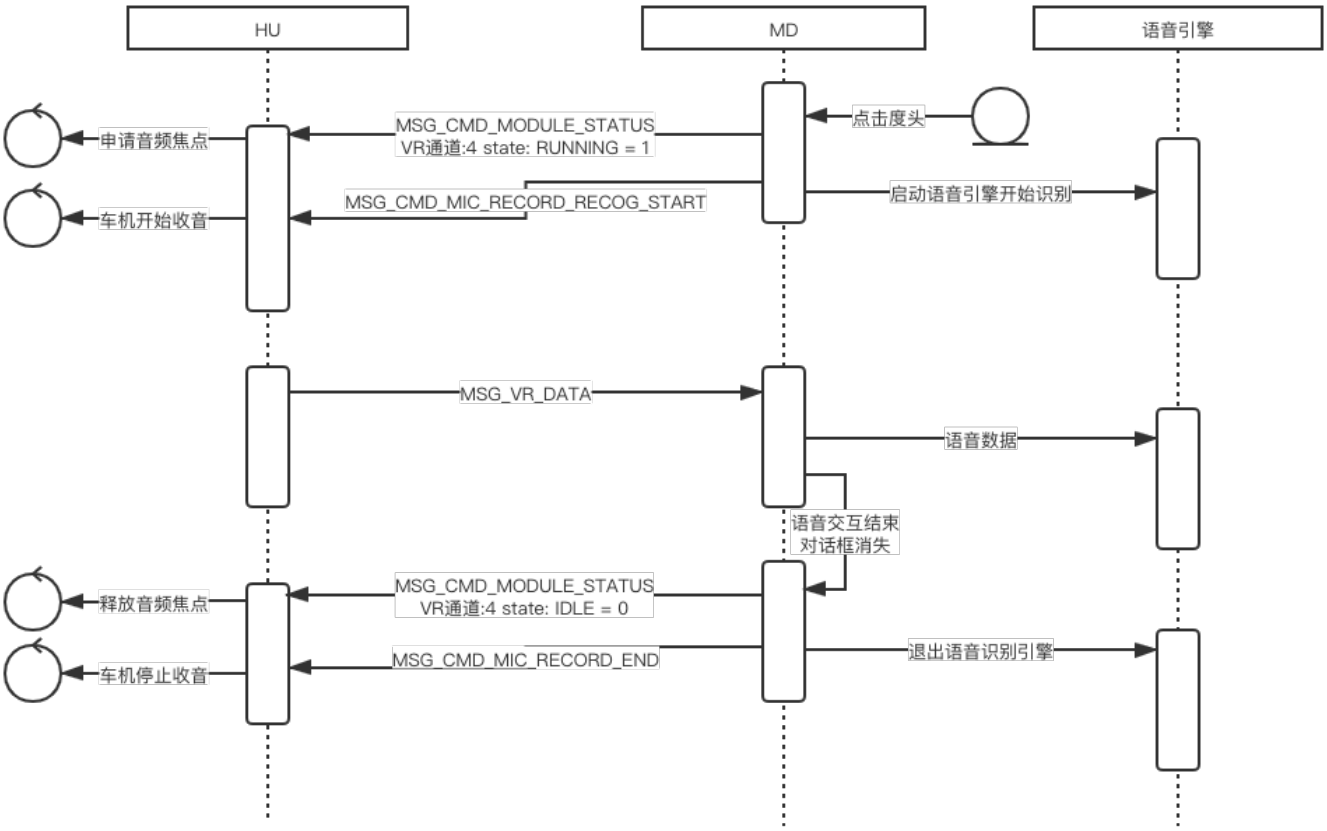
车机的MIC收音规则已经在SDK中实现，车厂需要给应用申请录音权限即可，无需其他操作，收音情况目前只支持以下四种规则。

录音	车机MIC	手机MIC
支持唤醒	7.1	
不支持唤醒	7.2	

7.1 支持唤醒的车机MIC规则



7.2 不支持唤醒的车机MIC规则



7.3 手机MIC录音

手机MIC的录音逻辑无需车机端处理收音相关消息。

八、CarLifeModule

CarLifeModule CarLife本身具备模块的概念并定义了导航、音乐、语音等几个模块，模块本身具备id（唯一表示）、state（状态）。当状态发生变化时，要实时同步给车机。**CarLifeContext内部封装了具体实现，使模块实现者只关注业务实现，无需关心协议交互。**

当移动设备端相关模块的状态改变时，就会发送给车机端，主要用于移动设备与车机互联时同步状态。涉及状态有：电话状态、导航状态、音乐状态、语音状态、连接状态、MIC状态、mediaPCM状态(仅用于IOS设备)、电子狗状态、cruise状态。

8.1 模块类型

模块名称	模块ID	枚举值	含义
电话	1	PHONE_STATUS_IDLE=0; PHONE_STATUS_INCOMING=1; PHONE_STATUS_OUTING=2;	
导航	2	NAVI_STATUS_IDLE=0; NAVI_STATUS_RUNNING =1;	
音乐	3	MUSIC_STATUS_IDLE=0; MUSIC_STATUS_RUNNING=1;	
语音	4	VR_STATUS_RECORD _IDLE =0; VR_STATUS_RECORD_RUNNING =1;	
连接	5	CONNECT_STATUS_ADB=1; CONNECT_STATUS_AOA=2; CONNECT_STATUS_NCM_ANDROID=3; CONNECT_STATUS_NCM_IOS =4; CONNECT_STATUS_WIFI=5;	
MIC	6	MIC_STATUS_USE_VEHICLE_MIC =0; MIC_STATUS_USE_MOBILE_MIC =1;	
MediaPCM	7	MEDIAPCM_STATUS_IDLE=0; MEDIAPCM_STATUS_RUNNING=1;	目前仅用于iOS设备。某些支持iPOD音乐播放的车机，当车机端CarLife切后台时，可以使用该命令暂停MD向HU 传输音乐PCM流，避免底层宽带不够用。
EDog	8	EDOG_STATUS_IDLE =0; EDOG_STATUS_RUNNING =1;	
Cruise	9	CRUISE_STATUS_IDLE =0; CRUISE_STATUS_RUNNING=1;	

8.2 状态处理

CarLifeSDK封装了一个抽象类，车机端APP如需监听某个模块的状态信息，直接继承该类即可，该抽象类为CarLifeModule.kt，如下：

```
package com.baidu.carlife.sdk

import com.baidu.carlife.sdk.internal.transport.TransportListener
import com.baidu.carlife.sdk.util.annotations.DoNotStrip

@DoNotStrip
abstract class CarLifeModule: TransportListener {
    protected var listener: ModuleStateChangeListener? = null

    abstract val id: Int
```

```
    open fun reader(): StreamReader {
        throw UnsupportedOperationException("module $id don't support read as stream")
    }

    // 设置模块状态，外部只需要设置状态即可，不需要关系消息如何发送，发送逻辑由CarLifeContext来实现
    var state: Int = 0
        set(value) {
            val oldState = field
            if (field != value) {
                field = value
                onModuleStateChanged(value, oldState)
            }
        }

    fun setStateChangeListener(listener: ModuleStateChangeListener?) {
        this.listener = listener
    }

    open fun onModuleStateChanged(newState: Int, oldState: Int) {
        listener?.onModuleStateChanged(this)
    }

    // 连接状态回调，根据需要重写
    override fun onConnectionAttached(context: CarLifeContext) { }

    override fun onConnectionDetached(context: CarLifeContext) { }

    override fun onConnectionEstablished(context: CarLifeContext) { }
}
```

实现该抽象类后，即使用**CarLife.receiver()** 注册即可，如下：

```
CarLife.receiver().addModule(****)
```

涉及模块状态的改变都会回调到onModuleStateChanged方法中。

8.2.1 CarLifeModule

CarLifeModule实现了TransportListener接口，可以监听连接状态的改变以及监听收发的协议消息。

概要

Public methods

返回值	方法
StreamReader	reader() VR数据相关操作
void	setStateChangeListener(listener: ModuleStateChangeListener?) 设置监听，如有需要的话，可以监听module状态的改变
void	onModuleStateChanged(newState: Int, oldState: Int) 状态改变时调用该方法

Members

成员	含义
listener	ModuleStateChangeListener：状态更改监听类
id	Int：模块ID，如音乐模块的值为3
state	Int：状态值ID

示例

监听电话状态。

```
package com.baidu.carlifevehicle.module

import com.baidu.carlife.sdk.CarLifeContext
import com.baidu.carlife.sdk.CarLifeModule
import com.baidu.carlife.sdk.Constants
import com.baidu.carlife.sdk.receiver.OnPhoneStateChangeListener
import com.baidu.carlife.sdk.util.Logger
import com.baidu.carlifevehicle.CarlifeActivity


import com.baidu.carlifevehicle.message.MsgHandlerCenter
import com.baidu.carlifevehicle.util.CommonParams

class PhoneModule(private val context: CarLifeContext,
                  private val activity: CarlifeActivity,
                  val callback: OnPhoneStateChangeListener)
    : CarLifeModule() {
    override val id: Int = Constants.PHONE_MODULE_ID

    override fun onModuleStateChanged(newState: Int, oldState: Int) {
        Logger.d(Constants.TAG, "newState: $newState, activity.mIsConnectException:
${activity.mIsConnectException}")
        // 电话状态车机端无需任何处理
```

```
        when(newState) {
            Constants.PHONE_STATUS_IDLE  -> {
                this@PhoneModule.callback.onStateChanged(false, false)
                if (activity.mIsConnectException) {

MsgHandlerCenter.dispatchMessage(CommonParams.MSG_MAIN_DISPLAY_MAIN_FRAGMENT)
                } else {CarlifeActivity
                    //通话结束后直接返回到投屏界面

MsgHandlerCenter.dispatchMessage(CommonParams.MSG_MAIN_DISPLAY_TOUCH_FRAGMENT)
                }
            }

            else -> {
                this@PhoneModule.callback.onStateChanged(true, newState ==
Constants.PHONE_STATUS_INCOMING)

MsgHandlerCenter.dispatchMessage(CommonParams.MSG_MAIN_DISPLAY_EXCEPTION_FRAGMENT)
            }
        }
    }
}
```

使用**CarLife.receiver()** 添加：

```
        mPhoneModule = PhoneModule(CarLife.receiver(), this, this)
        CarLife.receiver().addModule(mPhoneModule)
```

8.2.2 ModuleStateChangeListener

状态监听器

```
package com.baidu.carlife.sdk;

import com.baidu.carlife.sdk.util.annotations.DoNotStrip;

@DoNotStrip
public interface ModuleStateChangeListener {
    void onModuleStateChanged(CarLifeModule module);
}
```

实现该接口可以监听某个模块的状态变更，使用Module实例的setStateChangeListener方法设置即可。

概要

Public methods

返回值	方法
void	onModuleStateChanged(CarLifeModule module) 状态变更回调

8.3 SDK实现

CarLife SDK 里面实现了音乐、导航、语音模块的状态处理，无需车厂再进行处理，其他模块车厂如有需要，可以继承CarLifeModule去实现，并使用**CarLife.receiver()** 添加即可。

九、CarLife上下文

SDK的所有接口调用都依赖CarLife上下文，所以这里列出其所以API接口，三方开发者可根据其实例进行调用。

9.1 CarLifeContext

CarLife上下文，通用方法封装，通过CarLifeContext可以发送消息、监听状态、加解密等等。

```
interface CarLifeContext {
    companion object {
        /**
         * 连接断开
         * 1 USB断开
         * 2 网络断开
         */
        const val CONNECTION_DETACHED    = 0
        /**
         * 连接建立，分下面三种情况
         * 1 AOA连接建立
         * 2 ADB六条Socket建立成功
         * 3 Wifi六条Socket建立成功
         */
        const val CONNECTION_ATTACHED    = 1

        /**
         * 重新attach，发生在
         * CONNECTION_ATTACHED -> CONNECTION_ATTACHED
         * CONNECTION_ESTABLISHED -> CONNECTION_ATTACHED
         */
        const val CONNECTION_REATTACH    = 2
```

```
    /**
     * 协议校验成功
     * statistics info 校验成功时
     */
    const val CONNECTION_ESTABLISHED = 3

    /**
     * 连接类型
     * aoa连接
     */
    const val CONNECTION_TYPE_AOA     = 0
    /**
     * 连接类型
     * wifi 热点连接
     */
    const val CONNECTION_TYPE_HOTSPOT = 1
    /**
     * 连接类型
     * wifi 直连
     */
    const val CONNECTION_TYPE_WIFIDIRECT = 2
}

// 获取连接状态
val connectionState: Int

// 获取连接类型
var connectionType: Int

// 设置支持的特性
fun setFeatures(features: List<CarlifeFeatureConfigProto.CarlifeFeatureConfig>)

// 设置支持的特性
fun setFeature(key: String, feature: Int)

// 获取支持的特性
fun getFeature(key: String, defaultConfig: Int): Int

// 枚举支持的特性
fun listFeatures(): List<CarlifeFeatureConfigProto.CarlifeFeatureConfig>

// 异步发送消息
fun postMessage(message: CarLifeMessage)

// 同步发送消息
fun sendMessage(message: CarLifeMessage)

// 设置配置项
```

```
fun setConfig(key: String, config: Any)

// 删除配置项
fun removeConfig(key: String)

// 获取配置项
fun <T> getConfig(key: String, defaultConfig: T): T

// 对消息进行加密
fun encrypt(message: CarLifeMessage): CarLifeMessage

// 对消息进行解密
fun decrypt(message: CarLifeMessage): CarLifeMessage

// 获取音频焦点
fun requestAudioFocus(listener: AudioManager.OnAudioFocusChangeListener,
                      streamType: Int,
                      focusType: Int,
                      focusGrantDelayed: Boolean = false,
                      focusForceGrant: Boolean = false): Int

// 释放音频焦点
fun abandonAudioFocus(listener: AudioManager.OnAudioFocusChangeListener)

// 查找模块
fun findModule(id: Int): CarLifeModule?

// 枚举模块
fun listModule(): List<CarLifeModule>

// 添加模块
fun addModule(module: CarLifeModule)

// 删除模块
fun removeModule(module: CarLifeModule)

// 查找订阅者
fun findSubscriber(id: Int): CarLifeSubscriber?

// 添加订阅者
fun addSubscriber(subscriber: CarLifeSubscriber)

// 删除订阅者
fun removeSubscriber(subscriber: CarLifeSubscriber)

// 查找被订阅者
fun findSubscribable(id: Int): CarLifeSubscribable?

// 添加被订阅者
```

```
    fun addSubscrible(subscribable: CarLifeSubscribable)

    // 删除被订阅者
    fun removeSubscrible(subscribable: CarLifeSubscribable)
}
```

9.2 CarLifeReceiver

CarLifeReceiver 车机端抽象接口，具备车机端的特有能力，反控事件发送、语音收音、音频播放与焦点管理等。

```
interface CarLifeReceiver: CarLifeContext {
    // 连接成功时，需要拉起的Activity
    val activityClass: Class<out Activity>

    /**
     * 添加连接进度监听器
     */
    fun addConnectProgressListener(listener: ConnectProgressListener)

    /**
     * 删除连接进度监听器
     */
    fun removeConnectProgressListener(listener: ConnectProgressListener)

    // 设置文件接收监听器
    fun setFileTransferListener(listener: FileTransferListener?)

    // 设置屏幕参数、屏幕宽高，帧率
    fun setDisplaySpec(displaySpec: DisplaySpec)

    // 创建好SurfaceView之后，调用此方法传递Surface，用于构建MediaCodec，解码渲染
    fun setSurface(surface: Surface?)

    // Activity 生命周期回调
    fun onActivityStarted()

    fun onActivityStopped()

    // 传递反控事件
    fun onTouchEvent(event: MotionEvent)

    // 传递硬按键消息
    fun onKeyEvent(keyCode: Int)
}
```

十、其他

10.1 SDK日志路径

scard/Android/data/应用包名/files/log/* ， SDK日志关键字“CarLife_SDK”