私有化版本接入说明

集成灯塔分析SDK

1. 引入离线SDK

在主 **module** 的 **build.gradle** 文件中添加 SDK 依赖：

build.gradle添加如下依赖：

```
implementation files('libs/beacon-android-【SDK对应的版本号】.aar')
```

注意确保项目libs目录，包含离线sdk

2. 配置混淆规则

```
-keep class com.tencent.qimei.** { *;}
-keep class com.tencent.qmsp.oaid2.** {*;}
-keep class com.tencent.beacon.** { *;}
```

3. 配置权限

必备权限

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

其他权限（可以不申请）

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

4. 反裁剪配置

由于Android release编译过程会默认把so做一个裁剪，而灯塔的so文件做过一些混淆加固，被裁剪后会导致so无法正常加载，所以需要加上下反裁剪配置

```
android{
    packagingOptions {
        doNotStrip "**/libBeacon.so"
        doNotStrip "**/libQimei.so"
        doNotStrip "**/libQmp.so"
        doNotStrip "**/libqmp.so"
    }
}
```

BeaconReport

灯塔SDK用于上报的主类，该对象为单例对象。灯塔的初始化、上报以及功能接口都为该类提供。 多进程需要分别初始化BeaconReport，独立进行上报。

初始化SDK

在 **Application** 的 **onCreate()** 方法中调用 **BeaconReport.getInstance.start()** 初始化 SDK。其中私有化版本可通过setUploadHost和setConfigHost设置自定义上报域名。

```
// 配置项详情参考后文BeaconConfig介绍，除AndroidID外，如不清楚可以都不填
BeaconConfig config = BeaconConfig.builder()
             .setAndroidID("aaa") // 重要，不设置androidID会影响数据监控(对业务数据无影响)
             .setUploadHost("vibeacon.onezapp.com")// 私有化部署设置自定义上报域名
             .build();
BeaconReport beaconReport = BeaconReport.getInstance();
beaconReport.setAppVersion("填入您的app版本"); // 可选
beaconReport.setChannelID("填入您的channelid"); // 可选
/**
 *  其他配置 如设置userid ,设置qq，以及配置权限采集开关，详情可查看后续功能介绍模块
 */
beaconReport.start(this, APP_KEY, config);
```

上报事件

```
public EventResult report(BeaconEvent beaconEvent);
```

```
Map<String, String> params = new HashMap<>();`
params.put("k1", "v1");`
BeaconEvent event = BeaconEvent.builder()
                .withCode("onClick")// 事件名，必须
                .withParams(params) // 事件参数，非必须
                .withAppKey(APP_KEY) // 非必须，如果不填，默认会使用
```

```
BeaconReport.start()初始化时传入的appKey
                .build();
EventResult result = BeaconReport.getInstance().report(event);
Log.i("TAG", "EventResult{ eventID:" + result.eventID + ", errorCode: " +
result.errorCode + ", errorMsg: " + result.errMsg + "}");
```

若该事件符合上报规范，则 **错误码为0** ，并且返回该事件在SDK中的唯一ID(以实时和普通分别计算)，若不符合上报规范则返回错误码以及信息，错误码对应表见附录。 注意：

1. **若BeaconEvent中AppKey参数传空则默认带上宿主AppKey，如果带上其他appkey会自动开启子通道进行上报**
2. EventCode不可为空！
3. **params中单个value最大长度为10K，kv整体最大为45K，超过限制会截断**

获取灯塔采集参数

```
public BeaconPubParams getCommonParams(Context context);
```

返回灯塔SDK采集到用户信息。

手机型号设置上报

```
public void setModel(String model);
```

注：由于政策合规原因不再默认采集手机型，如需上报，需主动设置手机型号

停止事件上报

```
// @param immediately 如果为true则会马上中断正在进行的任务，false则会等待任务完成后
再停止轮询。默认false
public void stopReport(boolean immediately);
```

暂停轮询上报，期间生成的事件可以正常入库存储；调用resumeReport()或者重新初始化可恢复上报。

恢复事件上报

```
public void resumeReport();
```

当调用了停止事件上报后需要恢复灯塔SDK轮询时调用。

JS和App的通信

集成了灯塔Web SDK的H5页面，在嵌入到App后，H5内的事件可以通过App进行发送，事件发送前会添加上App采集到的预置属性。该功能默认是关闭状态，如果需要开启，需要在H5端和App端同时进行配置，App端配置如下：

1. Activity onCreate时，允许JS和App的通信，并传入当前webView。

```
BeaconJsReport beaconJsReport = new BeaconJsReport();
// 开启内嵌H5通过App上报埋点的通路
beaconJsReport.enableBridge(webView);
```

2. webview userAgent 添加自定义标记:isApp

```
// webview userAgent 添加自定义标记:isApp
WebSettings webSettings = mWebView.getSettings();
webSettings.setUserAgentString(userAgent + " isApp");
```

注意：若webview有setWebChromeClient，需要实现继承自BeaconWebChromeClient的WebChromeClient，并在enableBridge时传入。若重写onConsoleMessage后return true拦截了消息，则SDK将不会处理h5传到app端的消息。若需使用app端和h5的通路，请保持不拦截。 代码参考如下：

```
// 实现继承自BeaconWebChromeClient的WebChromeClient，并在enableBridge时传入
MyWebChromeClient myWebChromeClient = new MyWebChromeClient();
mWebView.setWebChromeClient(myWebChromeClient);
mBeaconJsReport.enableBridge(mWebView, myWebChromeClient);

// webview userAgent 添加自定义标记:isApp
WebSettings webSettings = mWebView.getSettings();
webSettings.setUserAgentString(userAgent + " isApp");

public class MyWebChromeClient extends BeaconWebChromeClient {
    @Override
    public boolean onConsoleMessage(ConsoleMessage consoleMessage) {
        Log.i(TAG, "onConsoleMessage:" + consoleMessage.message());
        // 注意：这里如果 return true 拦截了，SDK将不会处理h5传到app端的消息。若需使用 app 端和 h5 的通路，请保持不拦截
        return super.onConsoleMessage(consoleMessage);
    }

    @Override
    public boolean onJsPrompt(WebView view, String url, String message, String defaultValue,
            JsPromptResult result) {
        Log.i(TAG, "onJsPrompt url:" + url + ",  message: " + message + ", defaultValue: " + defaultValue);
        return super.onJsPrompt(view, url, message, defaultValue, result);
    }
}
```

Activity onDestory时，关闭JS和App的通信

```
// 关闭内嵌H5通过App上报埋点的通路
beaconJsReport.disableBridge();
```

获取当前SDK版本

```
public String getSDKVersion();
```

BeaconEvent

```
public final class BeaconEvent {
    private String appKey; // 事件AppKey
    private String code; // 事件名
    private EventType type; //事件类型
    private Map<String, String> params;// 事件参数
//...
```

例如：

```
Map<String, String> params = new HashMap<>();
params.put("k1", "v1");
BeaconEvent event = BeaconEvent.builder()
                .withCode("onClick") //必填
                .withType(EventType.REALTIME) //非必须，默认为普通事件
                .withParams(params) //非必须
                .withAppKey(appKey)// 非必须，默认为宿主AppKey
                .withIsSucceed(true)// 非必须，默认为true
                .build();
```

BeaconConfig

在初始化时传入配置，除androidID外，其他都可以不填

```
public class BeaconConfig {
   private final int maxDBCount;//DB存储的最大事件条数(实时和普通分开计算)，默认为
1万条，最大存储事件条数区间[10000，50000]
   private final boolean strictMode;// 严苛模式，默认false
   private final boolean logAble;// 日志开关，默认false
   private final boolean abroad;// 是否开启海外版，对qimei有影响，默认false
   private final boolean eventReportEnable; // 是否打开事件上报功能,默认true
   private final boolean auditEnable; // 是否开启稽核功能，默认true
```

```
    private final boolean bidEnable;// 是否开启BeaconID信息采集,默认true
    private final boolean collectMACEnable;//是否采集MAC地址信息，默认true
    private final boolean collectIMEIEnable;// 采集IMEI、IMSI信息，默认true
    private final long realtimePollingTime;//实时事件上报轮询间隔(ms)
    private final long normalPollingTIme;// 普通事件上报轮询间隔(ms)
    private final NetAdapter httpAdapter;// 设置OKHttpClient
```

例如：

```
 BeaconConfig config = BeaconConfig.builder()
         .strictMode(true)
         .logAble(true)
         .maxDBCount(20_000)
         .collectIMEIEnable(false)
         .collectMACEnable(false)
         .setNormalPollingTime(3000)
         .setRealtimePollingTime(1000)
         .setHttpAdapter(OkHttpAdapter.create(new OkHttpClient()))
         .build();
```

私有化部署设置自定义域名

```
BeaconConfig.builder()
.setUploadHost("vibeacon.onezapp.com")
.build();
```

EventResult

```
public final class EventResult{
 public int errorCode;
 public long eventID;
 public String errMsg;
}
```

EventType

```
public enum EventType {
    // 普通事件
    NORMAL,
    // 实时事件
    REALTIME
}
```

BeaconPubParams

灯塔采集信息对象

```
public class BeaconPubParams {
    private String boundleId;                          // B: App包名
    private String appVersion;                         // G: 产品版本
    private String sdkId;                              // G: SDK Id
    private String sdkVersion;                         // G: SDK 版本
    private String productId;                          // G: AppKey
    private String beaconId;                           // EV: Beacon Id
    private String appFirstInstallTime;               // EV：宿主App首次安装时间
    private String appLastUpdatedTime;                // EV: 宿主App最近一更新时间
    private String platform;                           // G: 平台
    private String dtMf;                               // EV: manufacturer，新
增采集
    private String osVersion;                          // G: 固件版本
    private String hardwareOs;                         // 设备信息
    private String brand;                               // EV: 品牌
    private String model;                               // G: 机型
    private String language;                            // EV: 语言
    private String resolution;                          // EV: 分辨率
    private String dpi;                                 // EV: Density per
inch
    private String gpu;                                 // GPU info
    private String isRooted;                           // EV: 是否rooted
    private String fingerprint;                         // EV: 指纹信息
    private String qimei;                               // EV: QIMEI
    private String imei;                                // EV: IEMI
    private String dtImei2;                            // EV: IMEI2:新增采集
    private String dtMeid;                             // EV: MEID: 新增采集
    private String imsi;                                // EV: IMSI
    private String androidId;                          // EV: ANDROID_ID
    private String modelApn;                           // G: 设备设置APN
    private String mac;                                 // EV: MAC
    private String wifiMac;                            // EV: WiFi Mac
    private String wifiSsid;                           // EV: WiFi ssid
    private String allSsid;                            // a109：扫描当前设备连接
的路由器下的所有设备IP和mac地址，第一组为本机的IP和mac地址
    private String networkType;                        // 网络类型
    private String cid;                                 // EV：SD卡id
    // getter,setter
```

本地demo使用

1. 使用Android Studio打开Demo文件，安装并运行Demo

2. 配置参数，执行上报 

3. 登录平台，查看上报数据

SDK更新日志

V2.1.0

202-05-24

- 本地缓存上限取值区间为[10000, 50000]
- 增加上传失败重试策略，上报间隔 = 初始上报间隔 * 2^(上报失败次数)

附录

1. 事件错误码对应表

错误码	含义
0	成功
101	事件被后台配置抽样
102	事件模块功能被关闭
103	事件被提交到DB失败
104	当前事件没有对应的通道
105	事件整体kv字符串大于45K
106	事件名为空