# iOS SDK接入

# 1.添加产品

添加产品请进入qapm.qq.com。

## 2.1. SDK集成

iOS SDK 最低兼容系统版本 iOS 8.0

### 2.1.1. cocoaPods集成

> 在工程根目录Podfile(不是Podfile.lock)文件里面加入如下内容

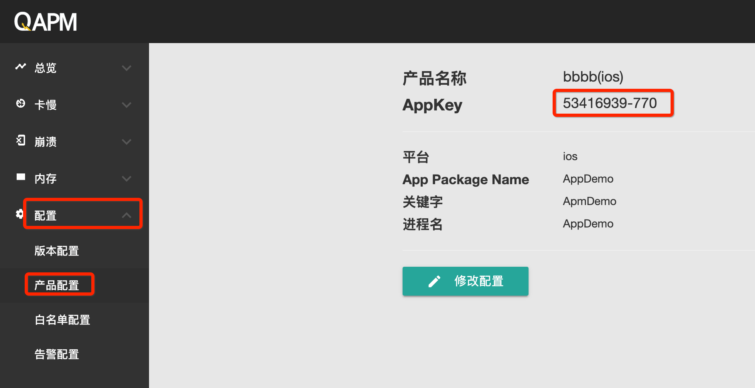
* + pod 'QAPM', :source => 'https://QAPM-cocoapods.pkg.coding.net/qapm\_sdk/ios\_release'

> 以上步骤完成之后，cd到Podfile目录,然后pod install 指令即可

> 将工程的bitcode参数设置为NO

## 2.2. 初始化SDK及web端环境配置

进入apm页面的【配置】-【产品配置】可以查看到AppKey，该key在初始化接入中需要用到。



在工程的AppDelegate.m文件导入头文件

#import <QAPM/QAPM.h>

如果是Swift工程，请在对应bridging-header.h中导入

初始化QAPM 在工程AppDelegate.m的application:didFinishLaunchingWithOptions:方法中初始化：

#if defined(DEBUG)

#define USE\_VM\_LOGGER

#ifdef USE\_VM\_LOGGER

/// 私有API请不要在发布APPSotre时使用。

typedef void (malloc\_logger\_t)(uint32\_t type, uintptr\_t arg1, uintptr\_t arg2, uintptr\_t arg3, uintptr\_t result, uint32\_t num\_hot\_frames\_to\_skip);

extern malloc\_logger\_t\* \_\_syscall\_logger;

#endif

#endif

void loggerFunc(QAPMLoggerLevel level, const char\* log) {

#ifdef RELEASE

if (level <= QAPMLogLevel\_Event) { ///外发版本log

NSLog(@"%@", [NSString stringWithUTF8String:log]);

}

#endif

#ifdef GRAY

if (level <= QAPMLogLevel\_Info) { ///灰度和外发版本log

NSLog(@"%@", [NSString stringWithUTF8String:log]);

}

#endif

#ifdef DEBUG

if (level <= QAPMLogLevel\_Debug) { ///内部版本、灰度和外发版本log

NSLog(@"%@", [NSString stringWithUTF8String:log]);

}

#endif

}

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

/// 设置QAPM 日志输出

NSLog(@"qapm sdk version : %@", [QAPM sdkVersion]);

[QAPM registerLogCallback:loggerFunc];

///开启线上稳定性功能，且设置本地功能开启命中的抽样率，建议开启为50%，即设置下列值为2即可

[[QAPMModelStableConfig getInstance] getModelStable:2];

//手动上传符号表

//考虑到金融私有云用户的安全问题，一般情况下是不建议用自动上传符号表脚本上传，在web端手动上传即可,需要在初始化操作的时候加入如下配置

//[QAPMConfig getInstance].uuidFromDsym = YES;

//自动上传符号表

//自动上传符号表初始化设置,此处uuid的值由自动上传符号表脚本传参而来，详见参考4.15.2.3.4自动上传符号表脚本；建议调试时实时打印uuid的值，如果uuid值为0，会影响正常翻译功能。

[QAPMConfig getInstance].uuidFromDsym = NO;

    NSString \*uuid = [[[NSBundle mainBundle] infoDictionary] objectForKey:@"com.tencent.qapm.uuid"];

**if**(!uuid){

        NSLog(@”请检查从第一个shell脚本传过来的uuid路径”);

    }

    [QAPMConfig getInstance].dysmUuid = uuid;

#ifdef USE\_VM\_LOGGER

/// ！！！Sigkill功能私有API请不要在发布APPSotre时使用。开启这个功能可以监控到VM内存的分配的堆栈。

[[QAPMConfig getInstance].sigkillConfig setVMLogger:(void\*\*)&\_\_syscall\_logger];

#endif

[QAPMConfig getInstance].host =@"https://qapm.qq.com";

[QAPMConfig getInstance].userId = @"设置userId";

[QAPMConfig getInstance].customerAppVersion = @"设置app自定义版本号";

[QAPMConfig getInstance].deviceID = @"自定义deviceId";

/// 启动QAPM

[QAPM startWithAppKey:@"产品唯一的appKey"];

return YES;

}

## 2.3. 功能配置

### 2.3.1. 卡顿检测功能

QAPMMoniterType: QAPMMoniterTypeBlue 卡顿检测将在卡顿时，卡顿时间超过设置的卡顿阈值则采集堆栈进行上报。 卡顿监控分滑动场景监控与非滑动场景监控。设置滑动场景监控会记录该滑动场景的场景名称，非滑动场景监控则场景名称为others。

* **滑动场景的卡顿监控**

在相关页面进行如下代码的打点。

- (void)viewWillDisappear:(BOOL)animated {

[super viewWillDisappear:animated];

[QAPMBlueProfile stopTrackingWithStage:NSStringFromClass([self class])];

}

#pragma mark - TableView Delegate

- (void)scrollViewWillBeginDragging:(UIScrollView \*)scrollView {

[QAPMBlueProfile beginTrackingWithStage:NSStringFromClass([self class])];

}

- (void)scrollViewDidEndDragging:(UIScrollView \*)scrollView willDecelerate:(BOOL)decelerate {

if(!decelerate){

[QAPMBlueProfile stopTrackingWithStage:NSStringFromClass([self class])];

}

}

- (void)scrollViewDidEndDecelerating:(UIScrollView \*)scrollView {

[QAPMBlueProfile stopTrackingWithStage:NSStringFromClass([self class])];

}

滑动场景卡顿监控除了会上报该滑动场景的卡顿数据外，还会统计出该场景的一个流畅度指标。

* **非滑动场景的卡顿监控**

如果需要监控非滑动场景的卡顿，请在启动QAPM之前设置[QAPMConfiggetInstance].blueConfig.monitorOtherStageEnable = YES; 默认为开启。

#### 2.3.1.1. 功能配置

> [QAPMConfig getInstance].blueConfig功能配置

@interface QAPMBlueConfig : NSObject

@property (nonatomic, assign) float sampleRate;

/\*\*

设置blue卡顿检测阈值(单位：s)

\*/

@property (nonatomic, assign) NSTimeInterval stuckThreshold;

/\*\*

设置blue系统方法堆栈记录开关，默认开启

\*/

@property (nonatomic, assign) BOOL systemStackTraceEnable;

/\*\*

打开所有场景的掉帧堆栈（除滑动外其它场景上报时的关键字为"others"),默认开启

该接口开启后会以CADisplayLink的刷新间隔（16.6ms）不断抓取主线程堆栈，可根据需要选择是否打开

在退后台的时候由于线程优先级降低，会使检测时间产生极大误差，强烈建议退后台的时候调用[QAPMBlueProfile updateMonitorOtherStageEnable:NO]关闭监控，在进前台时可以恢复监控！！！！

\*/

@property (nonatomic, assign) BOOL monitorOtherStageEnable;

@end

@interface QAPMBlueProfile : NSObject

/\*\*

开始记录掉帧，建议滑动开始时调用

\* @param stage 用来标识当前页面(一般为当前VC类名）

\*/

+ (void)beginTrackingWithStage:(NSString \*)stage;

/\*\*

结束记录掉帧，滑动结束时调用

\* @param stage 用来标识当前页面(一般为当前VC类名）

\*/

+ (void)stopTrackingWithStage:(NSString \*)stage;

/\*\*

更新所有场景的掉帧堆栈开关（除滑动外其它场景上报时的关键字为"others"),默认开启。

更新[QAPMConfig getInstance].blueConfig.monitorOtherStageEnable

在退后台的时候由于线程优先级降低，会使检测时间产生极大误差，强烈建议退后台的时候调用[QAPMBlueProfile updateMonitorOtherStageEnable:NO]关闭监控，在进前台时可以恢复监控！

\*/

+ (void)updateMonitorOtherStageEnable:(BOOL)enable;

/\*\*

滑动场景区分，如果不需要则设置为0

滑动结束时调用

\* @param type 设置为0时只有“Normal\_Scroll"的数据，当设置为其他值时，掉帧数据里面会多一个类型为"UserDefineScollType\_x"的数据

\*/

+ (void)setScrollType:(int32\_t)type;

@end

### 2.3.2. VC泄露检测功能

QAPMMoniterType: QAPMMoniterTypeYellow VC泄露检测将在退出VC后，在设置的阈值后进行检测一次是否已经被释放。如检测到泄露将立刻上报后台。 如果是白名单VC，则按照如下配置进行配置，将不进行VC检测。

#### 2.3.2.1. 功能配置

#pragma mark - Yellow(检测VC泄露功能)配置

@interface QAPMYellowConfig : NSObject

/\*\*

设置检测VC泄露阈值(单位：s)

\*/

@property (nonatomic, assign) NSTimeInterval leakInterval;

@end

@interface QAPMYellowProfile : NSObject

/\*\*

设置VC白名单类(对于需要在VC退出后驻留内存的VC)

@param set 白名单VC，set中的对象为NSString对象，是白名单VC类名，如果没有白名单则不设置

@param array 白名单基类VC，array中的对象为NSString对象，是白名单VC基类名，这些基类对象的所有子类都添加白名单

\*/

+ (void)setWhiteVCList:(NSSet \*)set baseVCArray:(NSArray \*)array;

/\*\*

针对白名单VC，可自定义检测时机，非白名单VC无需实现

注意：该方法在VC退出后调用，注意不要在dealloc方法中调用改方法，因为VC内存泄漏时无法执行dealloc

@param VC 白名单VC

\*/

+ (void)startVCLeakObservation:(UIViewController \*)VC;

/\*\*

设置该对象为白名单对象，无需监控

@param obj 白名单对象

\*/

+ (void)markedAsWhiteObj:(NSObject \*)obj;

@end

### 2.3.3. QQLeak(检测内存对象泄露功能)

QAPMMoniterType: QAPMMoniterTypeQQLeak QQLeak功能开启后，将记录内存对象堆栈的分配。在开启后，需要进行一次手动调用检测，检测时会挂起所有线程，耗时会比较长。建议在研发流程内使用。

#### 2.3.3.1. 功能配置

#pragma mark - QQLeak(检测内存对象泄露功能)配置

@interface QAPMQQLeakConfig : NSObject

@end

#pragma mark - 资源使用情况监控

@interface QAPMResourceMonitorConfig : NSObject

@end

@interface QAPMQQLeakProfile : NSObject

/\*\*

开始记录内存分配堆栈，需要开启后才能进行检测。

\*/

+ (void)startStackLogging;

/\*\*

停止记录内存分配堆栈

\*/

+ (void)stopStackLogging;

/\*\*

执行一次泄露检测，建议在主线程调用，该操作会挂起所有子线程进行泄露检测（该操作较耗时，平均耗时在1s以上，请限制调用频率）

\*/

+ (void)executeLeakCheck;

@end

### 2.3.4. 资源使用情况监控功能

QAPMMoniterType: QAPMMoniterTypeResourceMonitor 资源使用情况监控将每一秒采集CPU、内存、IO、网络等数据，每一分钟上报一下数据。

#### 2.3.4.1. 功能配置

#pragma mark - 资源使用情况监控功能配置

@interface QAPMResourceMonitorProfile : NSObject

/\*\* 设置资源使用监控起始标记

@param tag tag名称 \*/

+ (void)setBeginTag:(NSString \* \_\_nonnull)tag;

/\*\* 设置资源使用监控结束标记

@param tag tag名称 \*/

+ (void)setStopTag:(NSString \* \_\_nonnull)tag;

@end

### 2.3.5. 内存最大使用值监控(触顶率)

QAPMMoniterType: QAPMMoniterTypeMaxMemoryStatistic 触顶率将记录上一次App使用时的最大内存占用值，将在下次启动后上报。每个机型有一个内存的阈值，当最大内存占用值超过阈值则认为触顶。

#### 2.3.5.1. 功能配置

#pragma mark - 内存最大使用值监控(触顶率)配置

@interface QAPMMaxMemoryStatisticConfig : NSObject

@end

### 2.3.6. 大块内存分配监控功能

QAPMMoniterType: QAPMMoniterTypeBigChunkMemoryMonitor 通过设置内存分配监控的阈值，当内存分配超过阈值，将采集内存分配堆栈进行上报。

#### 2.3.6.1. 功能配置

#pragma mark - 大块内存分配监控配置

@interface QAPMBigChunkMemoryMonitorConfig : NSObject

/\*\*

设置单次超大堆内存监控阈值（bytes)，阈值设置较大时，性能开销几乎影响不计。默认阈值50M。

\*/

@property (nonatomic, assign) size\_t singleChunkMallocThreadholdInByte;

@end

### 2.3.7. Crash监控功能

QAPMMoniterType: QAPMMonitorTypeCrash

Crash日志会在下次启动SDK后上报数据。

@interface QAPMCrashMonitorProfile : NSObject

/\*\* Crash监控是否在运行

@return YES or NO \*/

+ (BOOL)isRunnning; @end

#### 2.3.8.Foom功能

QAPMMoniterType: QAPMMonitorTypeFoom

**@interface** QAPMSigkillConfig : NSObject

/\*\*

设置开启堆内存堆栈监控，将记录堆对象分配堆栈。

\*/

**@property** (**nonatomic**, **assign**) **BOOL** mallocMemoryDetectorEnable;

/\*\*

设置堆内存监控分配阈值 threshholdInBytes(bytes)，默认30M。

\*/

**@property** (**nonatomic**, **assign**) size\_t mallocMemoryThreshholdInByte;

/\*\*

设置VM内存监控分配阈值 threshholdInBytes(bytes)，默认30M。

\*/

**@property** (**nonatomic**, **assign**) size\_t vmMemoryThreshholdInByte;

/\*\*

设置堆内存监控抽样因子

如factor=10，则按照1/10抽样,factor=100,则按照1/100抽样。默认50。

\*/

**@property** (**nonatomic**, **assign**) uint32\_t mallocSampleFactor;

/\*\*

设置不进行抽样的内存阀值（bytes）

如sampleThreshhold=1024\*1024，则超过1Mb的内存分配不进行抽样,默认30\*1024。

\*/

**@property** (**nonatomic**, **assign**) uint32\_t mallocNoSampleThreshold;

/\*\*

设置开启VM堆栈监控，将记录堆对象分配堆栈。设置私有API \_\_syscall\_logger会带来app store审核不通过的风险，切记在提交关闭该监控，否则可能会审核不通过。

该功能会影响Instruments的Allocation工具无法使用。

设置方法：

typedef void (malloc\_logger\_t)(uint32\_t type, uintptr\_t arg1, uintptr\_t arg2, uintptr\_t arg3, uintptr\_t result, uint32\_t num\_hot\_frames\_to\_skip);

extern malloc\_logger\_t\* \_\_syscall\_logger;

[[QAPMConfig getInstance].sigkillConfig setVMLogger:(void\*\*)&\_\_syscall\_logger];

\*/

- (**void**)setVMLogger:(**void** \***\_Nonnull** \***\_Nonnull**)logger;

### 2.3.9deadock功能

QAPMMoniterType: QAPMMonitorTypeDeadlock

### 2.3.10. normal\_crash、foom、deadlock 自定义字段配置功能

**（1） 功能说明**

使用该功能便于业务自定义设置相关字段

}**（2） 相关接口**

/\*\*

**@return** 代表的依次是blame\_team和blame\_reason，根据QAPMUploadEventType的功能类型来自定义返回值

\*/

+ (NSDictionary<NSString \*, NSString \*> \*)eventUpSendEventWithTyped:(QAPMUploadEventCallback)callBack;

/\*\*

用于输出上报事件的自定义字段，参数为事件类型以及相应的堆栈信息。

\*/

**typedef** NSDictionary \*(\*QAPMUploadEventCallback)(QAPMUploadEventType eventType, **id** stackInfo);

**（3） 示例代码，在AppDelegate.m定义如下**

NSDictionary<NSString \*, NSString \*> \* Callback(QAPMUploadEventType eventType,**id** stackInfo) {

//NSLog(@" eventType:%zi\n stackIn:%@ \n",eventType, stackInfo);

NSDictionary \*dic;

**switch** (eventType) {

**case** 0:

dic = @{@"卡顿团队":@"测试用的1"};

**break**;

**case** 1:

dic = @{@"foom团队":@"测试用的2"};

**break**;;

**case** 2:

dic = @{@"deadlock团队的":@"测试用的4"};

**break**;

**case** 3:

dic = @{@"普通崩溃团队的":@"测试用的4"};

**break**;

**default**:

**break**;

}

**return** dic;

在didFinishLaunchingWithOptions方法里面等待QAPM初始化完如下调用

[QAPM eventUpSendEventWithTyped:Callback];

2.3.9. 启动耗时监控功能

@interface QAPMLaunchProfile : NSObject

/\*\*

开启启动耗时监控的调用 , 请在Main函数开始调用时设置

\*/

+ (void)setupLaunchMonitor;

/\*\*

【必须调用API】请在AppDidFinishLaunch开始调用时设置。

\*/

+ (void)setAppDidFinishLaunchBeginTimestamp;

/\*\*

【必须调用API】请在第一个页面ViewDidApppear开始调用时设置。

\*/

+ (void)setFirtstViewDidApppearTimestamp;

/\*\*

设置自定义打点区间开始，该区间需要在启动时间区间内。begin与end的scene需要一致。

当设置了 setFirtstViewDidApppearTimestamp 后，后面设置的自定义打点区间将不会被统计。

@param scene 场景名

\*/

+ (void)setBeginTimestampForScene:(NSString \*)scene;

/\*\*

设置自定义打点区间结束，该区间需要在启动时间区间内。begin与end的scene需要一致。

当设置了 setFirtstViewDidApppearTimestamp 后，后面设置的自定义打点区间将不会被统计。

@param scene 场景名

\*/

+ (void)setEndTimestampForScene:(NSString \*)scene;

@end

**（3） 代码示例**

1. 在main函数进行启动启动监控组件

int main(int argc, char \* argv[]) {

@autoreleasepool {

/// 设置堆栈采集抽样因子。例如设置1/10抽样，则设置fatctor = 10。默认为100。

[QAPMConfig getInstance].launchConfig.launchSampleFactor = 1;

/// 设置启动耗时阈值，当超过阈值会上报堆栈数据。默认为4000ms。

[QAPMConfig getInstance].launchConfig.launchthreshold = 3;

////设置Debug模式，Debug模式下连接Xcode也会进行上报启动耗时，默认为NO.

由于Debug下Xcode可能会额外加载一些动态库，导致启动耗时不准确，建议不调试进行上报数据。

[QAPMConfig getInstance].launchConfig.debugEnable = YES;

/// 开始启动耗时统计功能，需要在main函数中进行启动。

[QAPMLaunchProfile setupLaunchMonitor];

return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));

}

}

b.打点2个必须位置： \* application:didFinishLaunchingWithOptions 函数调用 [QAPMLaunchProfile setAppDidFinishLaunchBeginTimestamp]; \* 启动结束时机，一般是第一个页面viewDidAppear函数，在viewDidAppear最后调用[QAPMLaunchProfile setFirtstViewDidApppearTimestamp];

/// AppDelegate文件

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

/// 函数最开始调用启动监控组件API。

[QAPMLaunchProfile setAppDidFinishLaunchBeginTimestamp];

/// 业务逻辑

/// ....

}

/// 第一个显示页面的viewDidAppear函数

- (void)viewDidAppear:(BOOL)animated {

[super viewDidAppear:animated];

/// 业务逻辑

/// ....

/// 最后调用启动监控组件API

[QAPMLaunchProfile setFirtstViewDidApppearTimestamp];

}

### 2.3.11. Web监控功能

目前WKWebView支持 iOS >= 11 。

WebView页面性能监控数据在页面加载结束后延迟2秒上报，页面加载时间超过3秒会在上报数据中附带个例数据。该功能能够监控web网络资源加载耗时、jserror监控。

#### 2.3.11.1. 功能配置

1. **Web端配置**

**注意：如果用的是手动集成的方式**需要将framework里面的js\_sdk以Add Files to方式引入到工程里面；如果是用的cocoaPods集成方式则不用

如果用到TMFWebOffline离线包功能，工程里面的wkwebview相关页面的头文件需要引入

#import <TMFQWebView/QBWKWebView.h>,。

且遵循TMFWebOfflineWebViewControllerProtocol代理，wkwebview继承TMFWkWebView 如下设置

#import <TMFQWebView/QBWKWebView.h>

@interface WKWebviewViewController ()<WKUIDelegate, WKNavigationDelegate, WKScriptMessageHandler,TMFWebOfflineWebViewControllerProtocol>

{

TMFWkWebView \*wkWebView;

}

1. **iOS SDK 配置**
   1. 类文件中添加 #import <QAPM/QAPM.h> 导入SDK头文件。
   2. 在 WKWebView 的代理方法webView:didCommitNavigation:中添加如下代码，以提供web获取native相关信息接口。

- (void)webView:(WKWebView \*)webView didFinishNavigation:(WKNavigation \*)navigation {

[webView evaluateJavaScript:[QAPMWebViewProfile qapmBaseInfo:@" "] completionHandler:**nil**];

[webView evaluateJavaScript:[QAPMWebViewProfile qapmJsStart] completionHandler:**nil**];

}

webview上报日志信息可通过Safari浏览器调试webview的方式查看。

### 2.3.12. 用户行为Athena功能

#### ­2.3.12.1. 功能配置

项目中用到Athena功能的时候初始化配置如下

1. 相关接口

**@interface** QAPMUBSMonitor : NSObject

/\*\*

用户自定义用户行为操作调用,外部用户接口，调用该接口时请完成QAPM的一系列初始化操作，设置完QAPM的appKey后调用。

**@param** category 类别

**@param** label 事件标签

**@param** action 操作

**@param** value 数据值

**@param** tags 字符串的map标记

**@param** values 数值的map标记

**@return** 用户行为event uuid

\*/

- (NSString \*)generateUserEvent:(NSString \*)category

label:(NSString \*)label

action:(NSString \*)action

value:(NSNumber \*)value

tags:(NSDictionary<NSString \*, NSString \*> \*)tags

values:(NSDictionary<NSString \*, NSNumber \*> \*)values;

/\*\*

添加分桶实验(例如A/B test)，可添加多个，多次调用即可.

\*/

- (**void**)addBucket:(NSString \*)bucket;

/\*\*

移除单个bucket.

\*/

- (**void**)removeBucket:(NSString \*)bucket;

/\*\*

移除所有bucket.

\*/

- (**void**)removeAllBuckets;

### 2.3.13. 用户隐私协议

#### 2.3.13.1. 功能配置

隐私合规政策： 因隐私合规要求，在用户同意隐私合规之前请确保不调用QAPM的任何接口，此外QAPM仍然需要设备级的唯一标识用于确定设备的唯一性，用于用户指标级的计算

// 当用户授权后，方可正常初始化QAPM

if (isAgree) {

[QAPMLaunchProfile setAppDidFinishLaunchBeginTimestamp];//启动耗时函数的第一个打点

// 需要传入设备的唯一标识

[QAPMConfig getInstance].deviceID = @"自定义deviceId"; // 可以上传设备唯一标识，如IDFV的值

[QAPMConfig getInstance].userId = @"设置userId"; //手机号、第三方登录账号，此接口可以多次在代码位置使用

// 正常初始化代码贴入，参考文档2部分

}

#### 2.3.13.1. deviceId变更可能产生的影响

若在5.1.5版本之前已经使用自行传入设备标识符的方式接入QAPM，则可以继续采用传入自行生成的设备标识符。对原有的数据不会造成任何影响。

在5.1.4及之前的版本使用QAPM自行采集标识符号的方式接入QAPM，而在5.1.5及以上版本采用自行生成标识符号传入，则可能对crash率指标的总体数据有影响，历史数据和新版本的版本级数据均不会受到影响。

若您想要避免因更新deviceId带来的对于crash率的总体数据的影响，您可以通过自行实现以下方法生成deviceId：

deviceID = [NSString md5ToStr: openUDID];

生成openUDID的方式可参考如下开源库： [openUDID开源库](https://github.com/ylechelle/OpenUDID) 生成设备标识符并自行传入。

由于在SDK内调用获取openUDID与在业务代码中获取openUDID的实现位置不一致，生成的openUDID仍然可能存在差异。

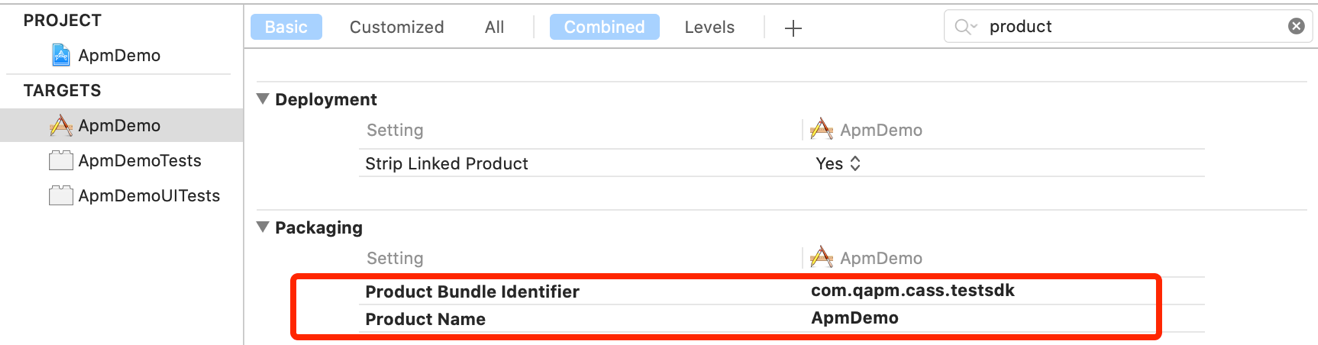
## 2.4. 符号表配置

### 2.4.1. 符号表关键字配置

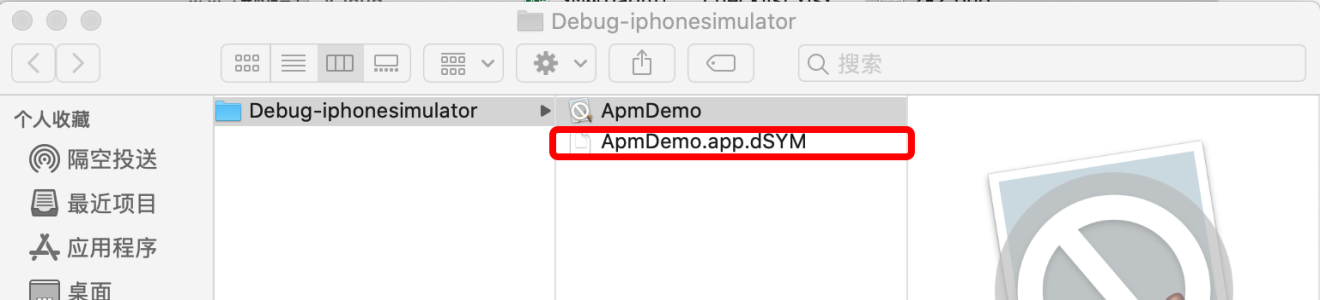
**设置关键字是翻译堆栈的前提，如不设置关键字将无法翻译！**

具体设置如下：

iOS默认的关键字为Build Setting ->Product Name ，以QAPM测试代码为例，如下图所示

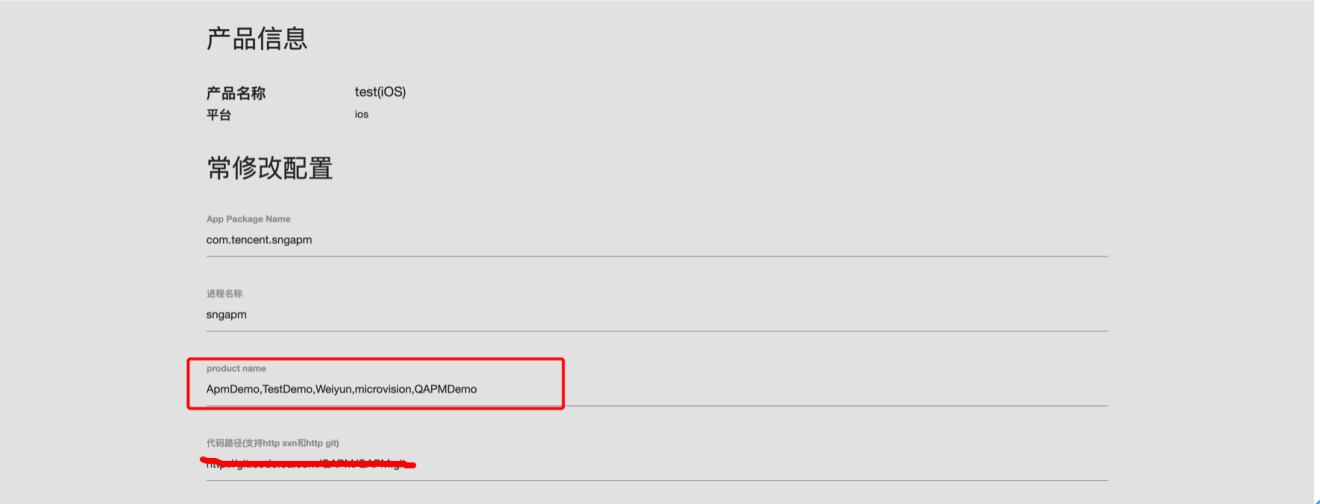


关键字以解压出来的符号表名称为准，下图的关键字则为"ApmDemo"



对于有多个符号表的产品，需要配置多个关键字(以QAPM代码为例,有多个关键字的配置)，修改如下





### 2.4.2. 符号表配置

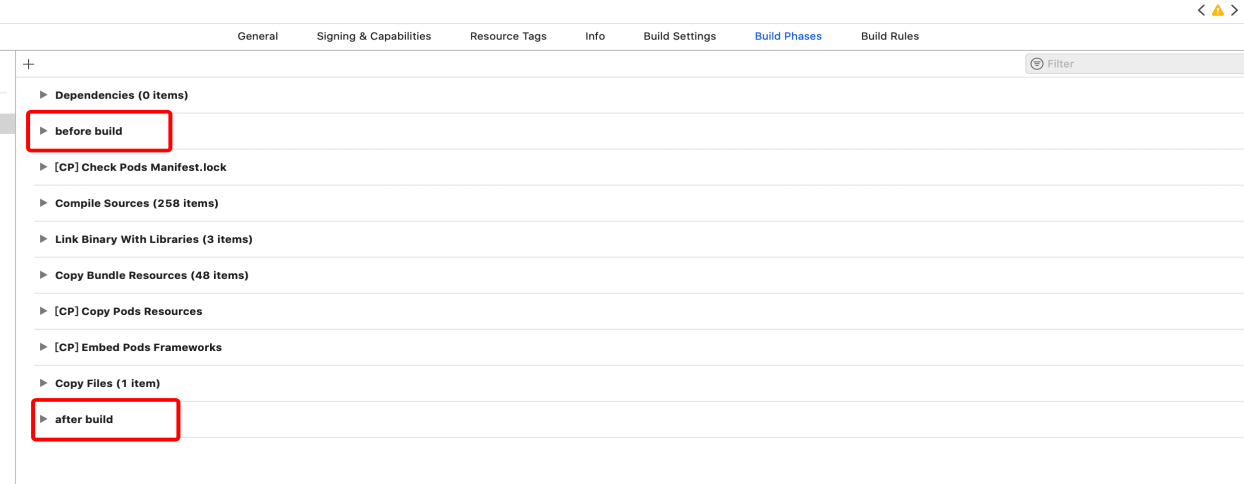
#### 2.4.2.2. 公有云外网、私有云

##### 自动上传符号表

1. 请在终端输入md5sum，检查是否包含该指令，如没有，可以根据下面文章配置

<https://blog.csdn.net/cup_chenyubo/article/details/52982986>

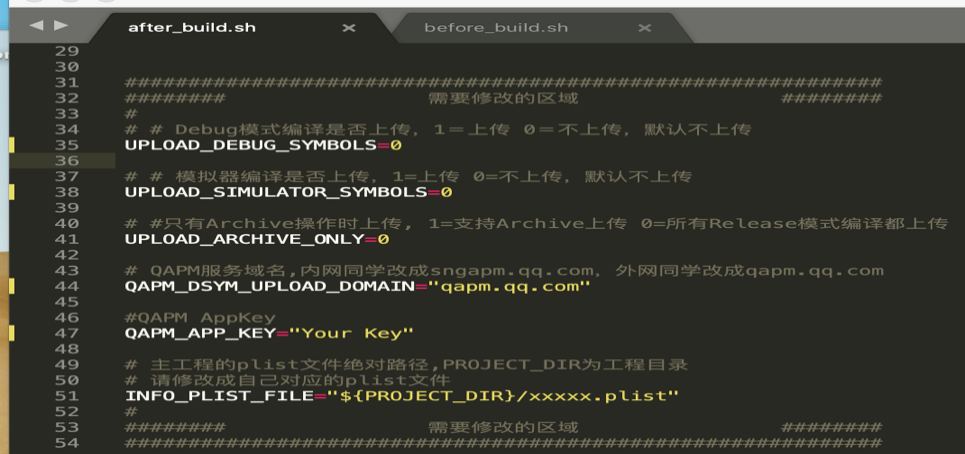
1. 压缩包中包含两份脚本，分别是before\_build.sh和after\_build.sh，将before\_build.sh中的脚本拷贝出来，放到编译的最前面，将after\_build.sh中的脚本拷贝出来，放到编译前的最后面，如下图所示



1. 脚本需要修改的地方，两个脚本都有个需要修改的区域，请根据自身需求修改。

**注意：**

建议Debug、模拟器编译不上传符号表，上传步骤会有符号表校验、压缩以及上传，会增大编译耗时



4） 请在QAPM初始化的代码中加入以下代码

参考如下

[QAPMConfig getInstance].uuidFromDsym = **NO**;

NSString \*uuid = [[[NSBundle mainBundle] infoDictionary] objectForKey:@"com.tencent.qapm.uuid"];

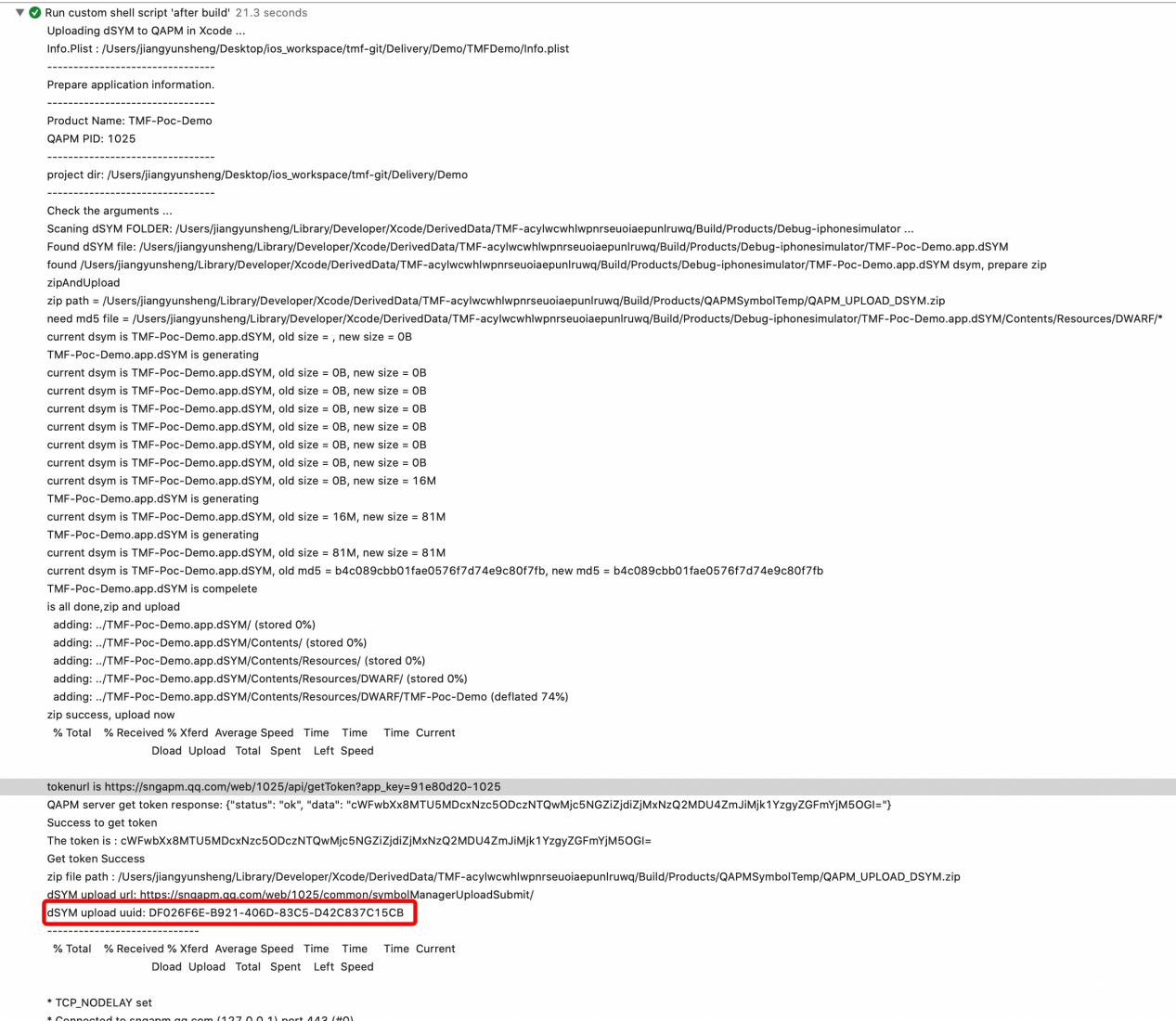
**if**(!uuid){

uuid = @"0";

}

[QAPMConfig getInstance].dysmUuid = uuid;

5） 默认上传符号表生成目录下的所有.dSYM结尾的符号表文件。测试脚本期间可以将Debug和模拟器选项置为1，执行编译，出现以下日志则代表上传成功。上传结束后，默认将源压缩包以及before\_build.sh中注入的uuid删除掉



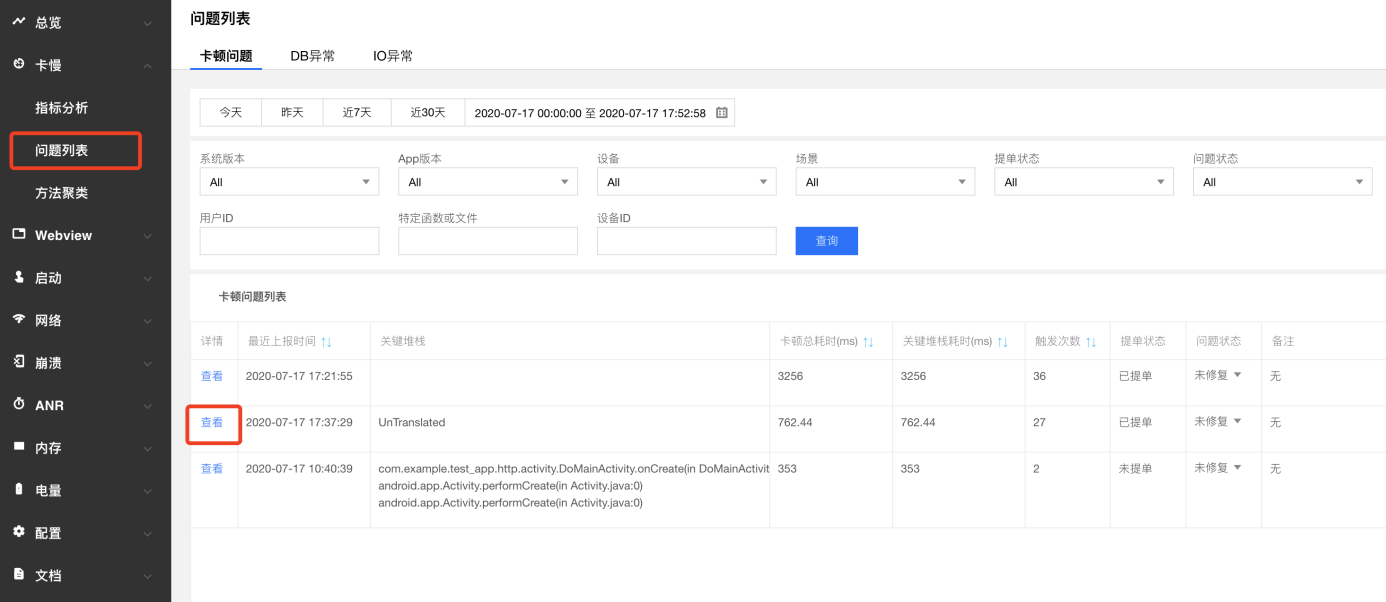
省略一部分……



##### 手工上传

针对无法通过打包自动上传符号表的情况，利用已上报的数据个例上传符号表（该方法对已上报的数据不会再做翻译处理）。

在用户个例页面中就有上传的入口，例如点击【卡慢】>【问题列表】>【详情】进入详情页面。



在详情页面中点击上传符号表按钮



进入符号表上传页面，点击【开始上传/加载】，然后选择该构建对应的maping文件即可。



##### （3） 如何检查SDK设置的uuid成功上报？

1. 打开sngapm.qq.com页面，打开"卡慢--卡顿个例"。

社交网站的手机截图

描述已自动生成

1. 在“App版本”中选择对应版本号，查询后点开“详情”。

手机屏幕截图

描述已自动生成 (3) 在“个例详情”里构建ID为该版本的符号表UUID (Android：编译日志搜索关键字QAPM*UUID，可找到该版本的UUID。iOS：检查ipa文件主工程的Info.plist里面的com.tencent.qapm.uuid字段是否和该构件ID一致)，则证明设置成功。*

## 2.5. 查看QAPM工作日志

### 2.5.1. 设置查看工作日志

在调用[QAPM startWithAppKey:]启动QAPM SDK前，设置日志输出函数。 可以根据不同发布版本情况进行输出日志控制：

void loggerFunc(QAPMLoggerLevel level, const char\* log) {

#ifdef RELEASE

if (level <= QAPMLogLevel\_Event) { ///外发版本log

NSLog(@"%@", [NSString stringWithUTF8String:log]);

}

#endif

#ifdef GRAY

if (level <= QAPMLogLevel\_Info) { ///灰度和外发版本log

NSLog(@"%@", [NSString stringWithUTF8String:log]);

}

#endif

#ifdef DEBUG

if (level <= QAPMLogLevel\_Debug) { ///内部版本、灰度和外发版本log

NSLog(@"%@", [NSString stringWithUTF8String:log]);

}

#endif

}

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

/// 设置QAPM 日志输出

[QAPM registerLogCallback:loggerFunc];

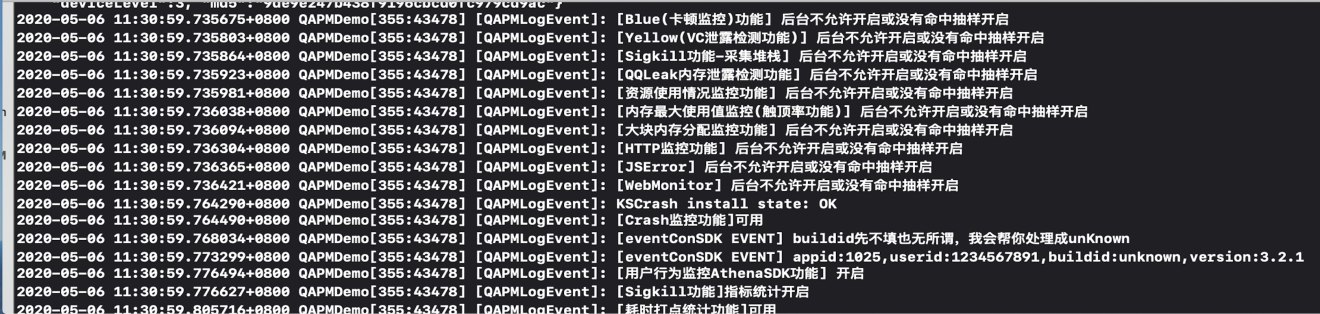
/// ...

/// 设置启动QAPM SDK

}

### 2.5.2. 上报日志分析

在接入完成SDK之后，通常情况下会通过分析日志来确定上报是否成功



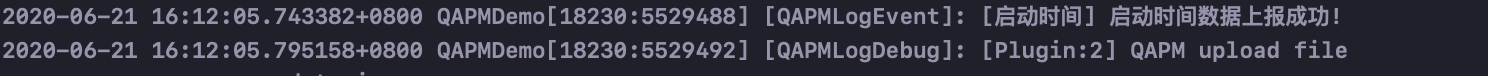
如果出现没有命中的字样则是说明有抽样率，iOS端默认抽样率为0.01，此时需要在web端配置用户或者设备白名单

当配置完白名单后，再次编译会出现以下日志,如果没有配置白名单请参照上图的4.15.2.1.2进行配置，命中抽样率如下图



通过初始化日志，可以看到初始化成功且命中用户抽样率，然后就是各功能上报成功的验证

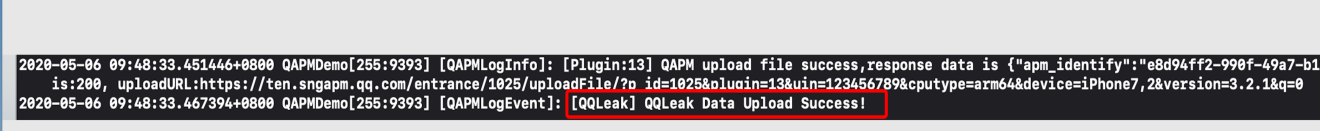
1. 启动耗时的上报



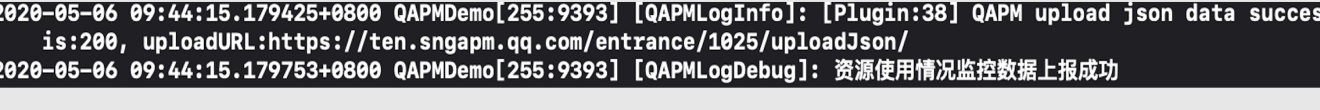
1. 卡顿个例的上报



1. 内存泄露上报



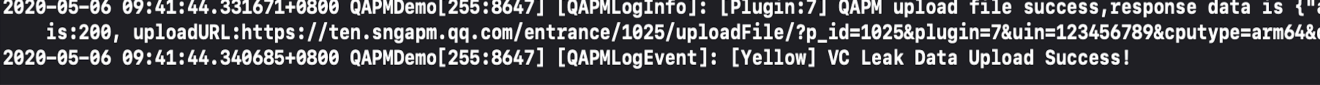
1. 资源使用情况上报



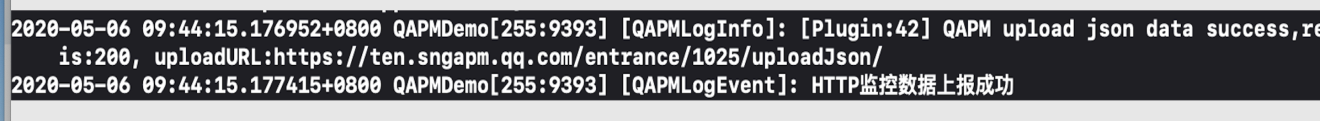
1. SIGKILL 上报



1. VC泄露上报

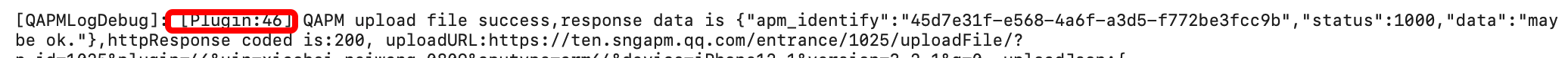


1. HTTP监控上报



1. 普通崩溃(normal crash)的上报

在触发normal crash的上报的时候，请不要将数据线连接Xcode，触发完normal crash后，下次重启app的时候即可看到上报信息，该上报日志可通过Mac自带的控制台查看上报日志，日志如下



1. Webview和JSerror的上报

Webview和jserror的上报，在查看webview的上报日志的时候，需要借助浏览器中的开发者工具，通过浏览器查看上报日志。



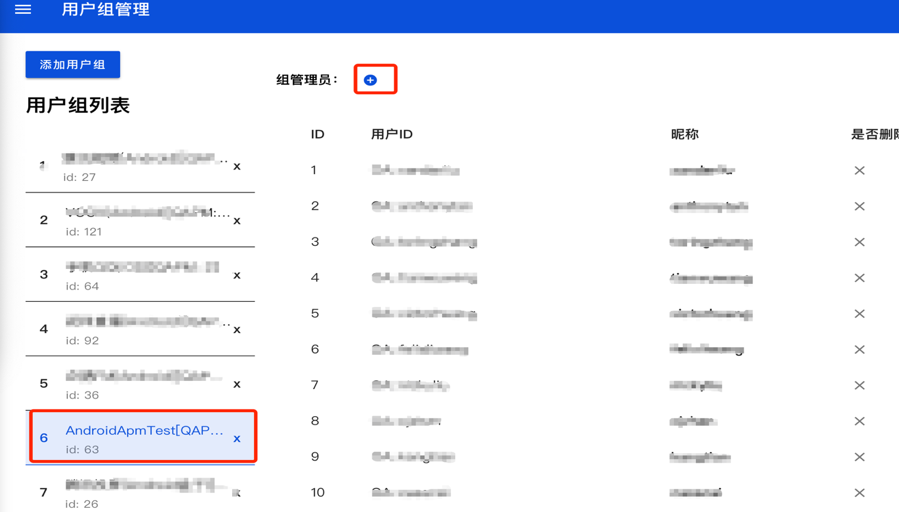
## 2.6. 添加用户组权限(只针对公有云用户分享权限的方式)

1.产品管理员需要通过点击链接[https://tencent.qq.com](https://tencent.qq.com/) 登入腾讯公共服务平台（TPS）->选择用户组管理，如下图所示



2.进入上述页面之后，**内网用户请用iOA或企业微信登陆，外网用户请用QQ登录**。

3.选择要赋予用户权限的产品（产品名为您在自助接入时填写的产品英文名）



4.点击对应 “+”生成对应权限链接，链接24小时内有效，建议为各职能负责人添加管理员权限



1. 将生成的链接发送给需要加入权限的同学