MTA 接口说明

本文档提供了 MTA.h 头文件中大部分接口的使用方法。全部接口及使用方法请查看头文件 MTA.h。若是第一次使用 MTA 请先查看快速接入指南;

头文件 MTA.h

启动 MTA

使用统计功能以前，需要先启动MTA。在UIApplicationDelegate的

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions;
```

回调中调用 MTA 的 startWithAppkey 方法即可启动MTA。

接口

```
/**
 启动MTA

 @param appkey 从网页申请的appKey
 */
+ (void)startWithAppkey:(NSString *)appkey;

/**
 检测版本，并启动MTA。
 如果当前MTA的版本小于ver参数，则MTA不启动。否则启动MTA。

 @param appkey 从网页申请的appKey
 @param ver 最低允许启动的版本
 @return 如果MTA成功启动，返回YES，否则返回NO
 */
+ (BOOL)startWithAppkey:(NSString *)appkey checkedSdkVersion:(NSString
*)ver;
```

示例

```
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [MTA startWithAppkey:@"ABCDEFG"];
}
```

统计页面时长

页面时长可以统计某个页面的访问时长

接口

```
/**
 标记一次页面访问的开始
 此接口需要跟trackPageViewEnd配对使用
 多次开始以第一次开始的时间为准

 @param page 页面名
 */
+ (void)trackPageViewBegin:(NSString *)page;


/**
 标记一起页面访问的开始
 并且指定上报方式
 此接口需要跟trackPageViewEnd配对使用
 多次开始以第一次开始的时间为准

 @param page 页面名
 @param appkey 若此参数不为nil，则上报到此appkey。否则，上报到startWithAppkey中传入的appkey
 */
+ (void)trackPageViewBegin:(NSString *)page appkey:(NSString *)appkey;


/**
 标记一次页面访问的结束
 此接口需要跟trackPageViewBegin配对使用
 多次结束以第一次结束的时间为准

 @param page 页面名字
 */
+ (void)trackPageViewEnd:(NSString *)page;

/**
 标记一起页面访问的结束
 并且指定上报方式
 此接口需要跟trackPageViewBegin配对使用
 多次结束以第一次结束的时间为准

 @param page 页面名
 @param appkey 若此参数不为nil，则上报到此appkey。否则，上报到startWithAppkey中传入的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackPageViewEnd:(NSString *)page
    appkey:(NSString *)appkey
    isRealTime:(BOOL)isRealTime;
```

示例

```
-(void) viewDidAppear:(BOOL)animated {
    [MTA trackPageViewBegin:@"Page1"];
    [super viewDidAppear:animated];
}

- (void)viewWillDisappear:(BOOL)animated {
    [MTA trackPageViewEnd:@"Page1"];
    [super viewWillDisappear:animated];
}
```

自定义事件

自定义事件分为两类

1. 次数统计
2. 时长统计

两类自定义事件都可以带参数，参数的类型有两种

1. NSDictionary 类型的参数
2. NSArray 类型的参数

因为NSDictionary类型的参数能表达的内容更丰富，因此推荐优先使用NSDictionary的参数上报。

NSDictionary为参数的自定义事件

接口

```
/**
 上报自定义事件

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
 */
+ (void)trackCustomKeyValueEvent:(NSString *)event_id props:(NSDictionary
*)kvs;

/**
 上报自定义事件
 并且指定上报方式

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策
略上报。
 */
+ (void)trackCustomKeyValueEvent:(NSString *)event_id
    props:(NSDictionary *)kvs
    appkey:(NSString *)appkey
```

```
    isRealTime:(BOOL)isRealTime;

/**
 开始统计自定义时长事件
 此接口需要跟trackCustomKeyValueEventEnd配对使用
 多次调用以第一次开始时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
 */
+ (void)trackCustomKeyValueEventBegin:(NSString *)event_id props:
(NSDictionary *)kvs;

/**
 开始统计自定义时长事件
 并指定上报方式
 此接口需要跟trackCustomKeyValueEventEnd配对使用
 多次调用以第一次开始时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 */
+ (void)trackCustomKeyValueEventBegin:(NSString *)event_id
    props:(NSDictionary *)kvs
    appkey:(NSString *)appkey;

/**
 结束统计自定义时长事件
 此接口需要跟trackCustomKeyValueEventBegin配对使用
 多次调用以第一次结束时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
            参数中的key和value必须跟开始统计时传入的参数一样才能正常配对
 */
+ (void)trackCustomKeyValueEventEnd:(NSString *)event_id props:(NSDictionary
*)kvs;

/**
 结束上报自定义时长事件
 并指定上报方式
 此接口需要跟trackCustomKeyValueEventBegin配对使用
 多次调用以第一次结束时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
            参数中的key和value必须跟开始统计时传入的参数一样才能正常配对
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
```

```
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackCustomKeyValueEventEnd:(NSString *)event_id
    props:(NSDictionary *)kvs
    appkey:(NSString *)appkey
    isRealTime:(BOOL)isRealTime;


/**
 直接统计自定义时长事件
 这个方法用于上报统计好的时长事件

 @param seconds 自定义事件的时长，单位秒
 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
 */
+ (void)trackCustomKeyValueEventDuration:(uint32_t)seconds
    withEventid:(NSString *)event_id
    props:(NSDictionary *)kvs;

/**
 直接上报自定义时长事件
 并指定上报方式
 这个方法用于上报统计好的时长事件

 @param seconds 自定义事件的时长，单位秒
 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param kvs 事件的参数，参数需要先在MTA前台配置好才能生效
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackCustomKeyValueEventDuration:(uint32_t)seconds
    withEventid:(NSString *)event_id
    props:(NSDictionary *)kvs
    appKey:(NSString *)appkey
    isRealTime:(BOOL)isRealTime;
```

示例

```
// 次数统计
- (IBAction)clickKVButton:(id)sender {
    [MTA trackCustomKeyValueEvent:@"KVEvent"
        props:[NSDictionary dictionaryWithObject:@"Value" forKey:@"Key"]];
}

// 时长统计
- (IBAction)clickStartKvButton:(id)sender {
    [MTA trackCustomKeyValueEventBegin:@"KVEvent"
        props:[NSDictionary dictionaryWithObject:@"Value"
forKey:@"TimeKey"]];
}

- (IBAction)clickEndKvButton:(id)sender {
    [MTA trackCustomKeyValueEventEnd:@"KVEvent"
        props:[NSDictionary dictionaryWithObject:@"Value"
forKey:@"TimeKey"]];
}
```

NSArray为参数的自定义事件

接口

```
/**
 上报自定义事件

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param array 事件的参数，参数需要先在MTA前台配置好才能生效
 */
+ (void)trackCustomEvent:(NSString *)event_id args:(NSArray *)array;


/**
 上报自定义事件
 并指定上报方式

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param array 事件的参数，参数需要先在MTA前台配置好才能生效
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackCustomEvent:(NSString *)event_id
    args:(NSArray *)array
    appkey:(NSString *)appkey
    isRealTime:(BOOL)isRealTime;

/**
 开始统计自定义时长事件
```

```
 此接口需要跟trackCustomEventEnd配对使用
 多次调用以第一次开始时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param array 事件的参数，参数需要先在MTA前台配置好才能生效
 */
+ (void)trackCustomEventBegin:(NSString *)event_id args:(NSArray *)array;

/**
 开始统计自定义时长事件
 并指定上报方式
 此接口需要跟trackCustomEventEnd配对使用
 多次调用以第一次开始时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param array 事件的参数，参数需要先在MTA前台配置好才能生效
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 */
+ (void)trackCustomEventBegin:(NSString *)event_id
    args:(NSArray *)array
    appkey:(NSString *)appkey;

/**
 结束统计自定义时长事件
 此接口需要跟trackCustomKeyValueEventBegin配对使用
 多次调用以第一次结束时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param array 事件的参数，参数需要先在MTA前台配置好才能生效
                参数中的各项必须跟开始统计时传入的参数一样才能正常配对
 */
+ (void)trackCustomEventEnd:(NSString *)event_id args:(NSArray *)array;

/**
 结束统计自定义时长事件
 并指定上报方式
 此接口需要跟trackCustomKeyValueEventBegin配对使用
 多次调用以第一次结束时间为准

 @param event_id 事件的ID，ID需要先在MTA前台配置好才能生效
 @param array 事件的参数，参数需要先在MTA前台配置好才能生效
                参数中的各项必须跟开始统计时传入的参数一样才能正常配对
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackCustomEventEnd:(NSString *)event_id
    args:(NSArray *)array
    appkey:(NSString *)appkey
```

```
    isRealTime:(BOOL)isRealTime;
```

示例

```
// 次数统计
- (IBAction)clickNormaltButton:(id)sender {
    [MTA trackCustomEvent:@"NormalEvent" args:[NSArray
arrayWithObject:@"arg0"]];
}

// 时长统计
- (IBAction)clickStartButton:(id)sender {
    [MTA trackCustomEventBegin:@"TimeEvent" args:[NSArray
arrayWithObject:@"arg0"]];
}

- (IBAction)clickEndButton:(id)sender {
    [MTA trackCustomEventEnd:@"TimeEvent" args:[NSArray
arrayWithObject:@"arg0"]];
}
```

上报当前缓存的事件

接口

```
/**
 上报当前缓存的数据
 若当前有缓存的事件（比如上报策略不为实时上报，或者有事件上报失败）时
 调用此方法可以上报缓存的事件

 @param maxStatCount 最大上报事件的条数
 */
+ (void)commitCachedStats:(int32_t)maxStatCount;
```

使用时长统计

在UIApplicationDelegate的

```
- (void)applicationDidBecomeActive:(UIApplication *)application;
- (void)applicationWillResignActive:(UIApplication *)application;
```

两个回调中，分别添加对应的打点代码，即可上报app的使用时长。

接口

```
/**
 开始统计使用时长
 建议在App进入前台时调用
 */
+ (void)trackActiveBegin;

/**
 结束统计使用时长
 建议在App离开前台时调用
 */
+ (void)trackActiveEnd;
```

示例

```
// 开始打点
- (void)applicationDidBecomeActive:(UIApplication *)application {
    [MTA trackActiveBegin];
}

// 结束打点
- (void)applicationWillResignActive:(UIApplication *)application {
    [MTA trackActiveEnd];
}
```

接口统计

统计应用对某个外部接口（特别是网络类的接口，如连接、登陆、下载等）的调用情况。当开发者用到某个外部接口，可调用该函数将一些指标进行上报，MTA 将统计出每个接口的 调用情况，并在接口可用性发生变化时进行告警通知; 对于调用量很大的接口，也可以采样上报，云监控统计将根据 sampling 参数在展现页面进行数量的还原。

接口

```
/**
 接口统计的枚举值
 */
typedef enum {
    /**
     接口调用成功
     */
    MTA_SUCCESS = 0,

    /**
     接口调用失败
     */
    MTA_FAILURE = 1,

    /**
```

```
     接口调用出现逻辑错误
     */
    MTA_LOGIC_FAILURE = 2
} MTAAppMonitorErrorType;


/**
 接口统计的数据结构
 */
@interface MTAAppMonitorStat : NSObject

/**
 监控业务接口名
 */
@property (nonatomic, retain) NSString *interface;

/**
 上传请求包量，单位字节
 */
@property uint32_t requestPackageSize;

/**
 接收应答包量，单位字节
 */
@property uint32_t responsePackageSize;

/**
 消耗的时间，单位毫秒
 */
@property uint64_t consumedMilliseconds;

/**
 业务返回的应答码
 */
@property int32_t returnCode;

/**
 业务返回类型
 */
@property MTAAppMonitorErrorType resultType;

/**
 上报采样率，默认0含义为无采样
 */
@property uint32_t sampling;
@end

/**
 对网络接口的调用情况进行统计
```

```
 参数的详细信息请看接口统计数据结构中的相关说明

 @param stat 接口统计的数据，详情请看接口统计数据结构的相关说明
 */
+ (void)reportAppMonitorStat:(MTAAppMonitorStat *)stat;

/**
 对网络接口的调用情况进行统计
 并指定上报方式
 参数的详细信息请看接口统计数据结构中的相关说明

 @param stat 接口统计的数据，详情请看接口统计数据结构的相关说明
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)reportAppMonitorStat:(MTAAppMonitorStat *)stat appkey:(NSString *)appkey isRealTime:(BOOL)isRealTime;
```

示例

```
-(IBAction) clickNormaltButton:(id)sender{
    MTAAppMonitorStat* stat = [[MTAAppMonitorStat alloc] init];
    [stat setInterface:@"interface1"];
    // ...
    [stat setRetsultType: SUCCESS];
    [MTA reportAppMonitorStat:stat];
}
```

用户画像

MTA 的用户画像功能需要开发者上报用户的 QQ 号码。上报 QQ 号码以后，MTA 后台会自动生成 APP 的用户画像。

接口

```
/**
 上报QQ号
 上报QQ号以后可以使用MTA提供的用户画像功能

 @param qq QQ号
 */
+ (void)reportQQ:(NSString *)qq;

/**
 上报QQ号
 并指定上报方式
 上报QQ号以后可以使用MTA提供的用户画像功能

 @param qq QQ号
 @param appkey 需要上报的appKey，若传入nil，则上报到启动函数中的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)reportQQ:(NSString *)qq appkey:(NSString *)appkey isRealTime:(BOOL)isRealTime;
```

示例

```
- (void)loginSuccess:(BOOL)animated {
    NSString *qq = @"45284547";
    [MTA reportQQ:qq];
}
```

错误统计

错误统计既可以统计 APP 的 crash。也可以统计 APP 中的逻辑错误。APP 的 crash 由 MTA 自动捕获并且上报，无需开发者调用额外的接口。而 APP 中的逻辑错误需要开发者手动调用相关接口上报。

接口

```
/**
 统计程序逻辑错误
 逻辑错误只有描述，没有堆栈信息

 @param error 错误描述
 */
+ (void)trackError:(NSString *)error;

/**
 统计程序逻辑错误
 并且指定上报方式
 逻辑错误只有描述，没有堆栈信息

 @param error 错误描述
 @param appkey 若此参数不为nil，则上报到此appkey。否则，上报到startWithAppkey中传入的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackError:(NSString *)error appkey:(NSString *)appkey isRealTime:(BOOL)isRealTime;

/**
 统计异常
 异常信息包括了异常的原因和堆栈

 @param exception 异常信息
 */
+ (void)trackException:(NSException *)exception;

/**
 统计异常
 并且指定上报方式
 异常信息包括了异常的原因和堆栈

 @param exception 异常信息
 @param appkey 若此参数不为nil，则上报到此appkey。否则，上报到startWithAppkey中传入的appkey
 @param isRealTime 是否实时上报，若传入YES，则忽略全局上报策略实时上报。否则按照全局策略上报。
 */
+ (void)trackException:(NSException *)exception appkey:(NSString *)appkey isRealTime:(BOOL)isRealTime;
```

在崩溃发生时候，MTA 会自动捕获崩溃的堆栈以及基本的上下文信息，并且在下次启动时候上报。除此之外，开发者还可以调用特定的 API 来储存额外的上下文信息，这些信息会在崩溃发生时，跟随崩溃报告一起上报，以便Debug。

接口

```
/**
 设置自定义的tag
 崩溃发生时，会将已经设置的tag上报，以便定位问题

 @param tagKey tag的Key，若key已经设置，则新的value会覆盖旧的value
 @param tagValue tag的value
 */
+ (void)setCustomTag:(NSString *)tagKey value:(NSString *)tagValue;


/**
 输出诊断信息
 崩溃发生时，会将最后输出的50条诊断信息上报，以便定位问题

 @param log 诊断信息
 */
+ (void)traceLog:(NSString *)log;
```

头文件MTAConfig.h

头文件 MTAConfig.h 提供了一些方法来自定义MTA的上报行为，比如上报策略，一次上报的条数等等。若有需要，可以查看 MTAConfig.h 头文件，做适当修改。

注：必须在调用 MTA 启动函数之前修改 MTAConfig 中的配置。否则配置可能不生效并且可能会引发 SDK 的一些未定义行为。