移动数据分析

开发指南

为了无法计算的价值 | 阿里云

开发指南

iOS SDK手册

移动数据分析

Mobile Analytics iOS SDK开发指南

1. 前言

本文档介绍了移动数据分析(Mobile Analytics) iOS SDK的使用方式。

Mobile Analytics iOS SDK是阿里云面向移动开发者提供的iOS平台下的数据统计与监控服务。通过该SDK，开发者可以在自己的APP中便捷地进行数据埋点，监控日常的业务数据与性能数据，并通过阿里云控制台界面观察对应的数据报表展现。另外，用户可以通过设定自定义的数据解析规则实现定制化的数据图表展现。

您可以通过获取alicloud-ios-demo工程源码获得移动数据分析服务的使用例程。

2. 安装Mobile Analytics iOS SDK

2.1 注意

使用1.0.7及之前版本请在**【Crash分析】**板块查看crash信息。

使用1.0.8版本及之后的版本，请在**【新版Crash分析】**板块查看crash信息。推荐使用1.0.8及之后的版本，crash数据更加准确，丢包率更小。

删除- (ALBBMANTracker *)getTracker:(NSString) trackerId方法，请使用- (ALBBMANTracker *)getDefaultTracker代替。

删除- (void)setCrashCaughtListener:(id<ALBBMANICrashCaughtListener>)aListener方法，1.0.8及之后的版本使用新的crash模块，无需再调用次方法设置参数。

2.2 手动集成SDK

将下载的包加至Link Binary With Libraries，包括：

1. AlicloudMobileAnalytics.framework
2. UTMini.framework
3. UTDID.framework
4. CrashReporter.framework

加载系统必须依赖的包：

1. libz.tbd
2. libresolv.tbd
3. libsqlite3.tbd
4. CoreTelephony.framework
5. SystemConfiguration.framework

2.3 Pod集成

- 指定Master仓库和阿里云仓库：

```
source 'https://github.com/CocoaPods/Specs.git'
source 'https://github.com/aliyun/aliyun-specs.git'
```

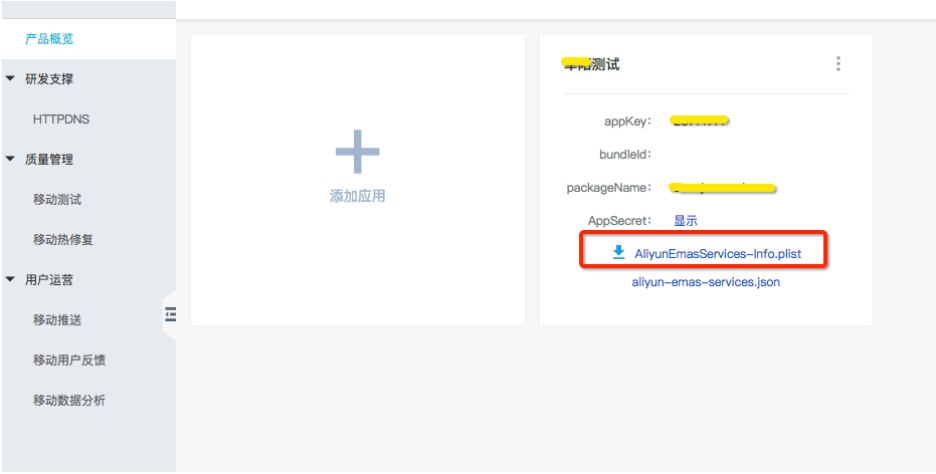
- 添加依赖：

```
pod 'AlicloudMAN', '~> 1.0.12'
```

（~>为模糊指定版本号方式，~> 1.0.12表明引用版本位于1.0.12 <= version < 1.1之间的最新版本SDK，用户可参考Podfile Syntax Reference，根据项目需要指定SDK版本。）

2.4 下载SDK统一配置文件

在控制台产品概览页面，下载App的配置文件AliyunEmasServices-Info.plist，如下图所示：



Xcode中，把下载的AliyunEmasServices-Info.plist文件拖入对应App Target下即可，在弹出框勾选Copy items if needed。

【附】AliyunEmasServices-Info.plist配置文件，包含SDK初始化所需的配置信息，用户只需要调用无需手动输入配置信息的autoInit初始化接口，参考第3节描述。

2.5 引用头文件

```
#import <AlicloudMobileAnalitics/ALBBMAN.h>
```

特别说明：应用的targets->Build Settings->linking->Other Linker Flags,请加上-ObjC这个属性。

3. 获取Mobile Analytics服务

在您使用Mobile Analytics iOS SDK进行数据统计与监控前，您需要首先获取Mobile Analytics服务，然后可以进行版本和渠道的配置。

- 手动输入配置信息，初始化接口：

```
- (void)initWithAppKey:(NSString *)appKey secretKey:(NSString *)secretKey;
```

- 自动初始化接口：

```
- (void)autoInit;
```

- MAN SDK初始化实例：

```
// 获取MAN服务
```

```
ALBBMANAnalytics *man = [ALBBMANAnalytics getInstance];
// 打开调试日志，线上版本建议关闭
// [man turnOnDebug];
// 初始化MAN，无需输入配置信息
[man autoInit];
// appVersion默认从Info.list的CFBundleShortVersionString字段获取，如果没有指定，可在此调用setAppversion设定
// 如果上述两个地方都没有设定，appVersion为"-"
[man setAppVersion:@"2.3.1"];
// 设置渠道（用以标记该app的分发渠道名称），如果不关心可以不设置即不调用该接口，渠道设置将影响控制台【渠道分析】栏目的报表展现。
[man setChannel:@"50"];
```

4. 业务数据统计

数据统计的准确性依赖被监控APP的常规生命轨迹，比如应用启动次数依赖于用户正常退出应用触发的上报策略。

4.1 登录/注册会员

4.1.1 登录会员

接口：

```
- (void)updateUserAccount:(NSString *)pNick userid:(NSString *)pUserId;
```

功能: 获取登录会员，然后会给每条日志添加登录会员字段

是否必须调用: 否

调用时机: 登录时调用

备注: 阿里云 平台上的登录会员 UV 指标依赖该接口

4.1.2 注册会员

接口:

```
- (void)userRegister:(NSString *)pUsernick;
```

功能: 产生一条注册会员事件日志

是否必须调用: 否

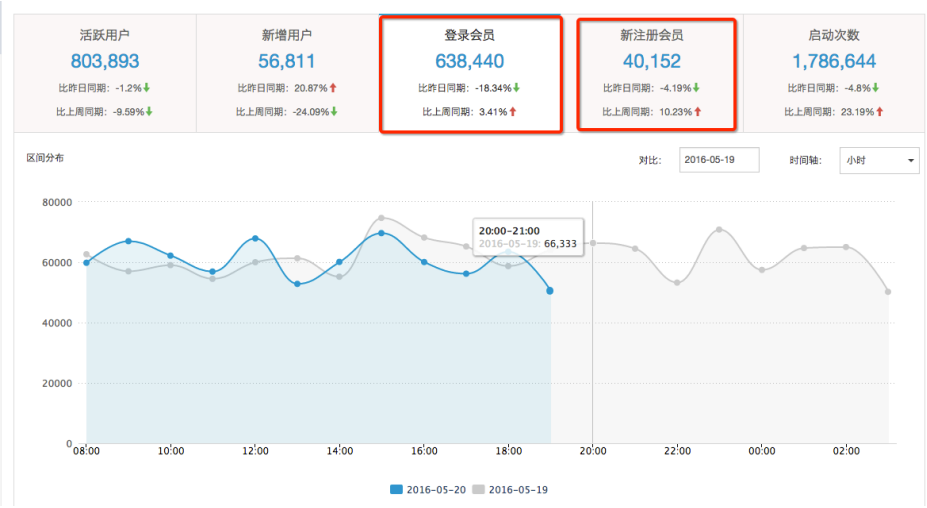
调用时机: 注册时调用

备注: 阿里云 平台上注册会员指标依赖该接口

4.1.3 代码示例

```
ALBBMANAnalytics *man = [ALBBMANAnalytics getInstance];
[man userRegister:@"userNick"];
[man updateUserAccount:@"userNick" userid:@"userId"];
```

完成上述埋点后，您就可以在阿里云控制台看到相应统计信息，例如下图显示实时统计里的登录会员和新注册会员。



4.2 页面埋点

使用ALBBMANPageHitHelper可以进行ViewController级别的页面埋点，ALBBMANPageHitHelper为页面埋点辅助类，可以自动完成页面名称（默认获取ViewController名称并去除后缀Controller）、来源页面和页面停留时间的统计，如需使用更为灵活的页面埋点方案，可使用ALBBMANPageHitBuilder页面打点基础类进行页面埋点，详情见4.3节。

【注意】：如果App中没有进行页面埋点，活跃用户参数不能正常统计。

4.2.1 页面进入

接口：

```
- (void)pageAppear:(UIViewController *)pViewController;
```

功能： 记录页面进入时的一些状态信息，但不发送日志，和 pageDisappear 配合使用

入参： UIViewController 或者其子类指针

是否必须调用； 需要对页面埋点时调用

调用时机： viewDidAppear

备注： 必须和 pageDisappear 搭配使用

4.2.2 页面离开

接口：

```
- (void)pageDisAppear:(UIViewController *)pViewController;
```

功能： 页面离开发送页面事件日志，和 pageAppear 配合使用

入参： UIViewController 或者其子类指针

是否必须调用： 当调用了 pageAppear 后，必须调用 pageDisappear

调用时机： viewDidDisAppear

备注： 必须和 pageAppear 搭配使用

4.2.3 设置页面扩展参数

接口：

```
- (void)updatePageProperties:(UIViewController *)pViewController properties:(NSDictionary *)pProperties;
```

功能：设置页面扩展参数

是否必须调用：否

调用时机：调用pageDisAppear之前

4.2.4 代码示例

```
// 进入页面
[[ALBBMANPageHitHelper getInstance] pageAppear:self];

// 设置页面事件扩展参数
NSDictionary *properties = [NSDictionary dictionaryWithObject:@"pageValue" forKey:@"pageKey"];
[[ALBBMANPageHitHelper getInstance] updatePageProperties:self properties:properties];

// 离开页面
[[ALBBMANPageHitHelper getInstance] pageDisAppear:self];
```

4.3 页面事件

ALBBMANPageHitBuilder 类也是用来产生页面事件日志的，虽然 4.2 节中可以通过 pageAppear 和 pageDisAppear 来进行页面埋点，但是它针对的只是 UIViewController 级别的页面，并不能满足一些场景的页面事件，例如 UIView，如果用户需要将一个UIView 当做一个页面，那么就可以通过 ALBBMANPageHitBuilder 来进行页面事件的埋点，即用户自己采集页面事件相关的信息(如页面的 refer、页面停留时间等)，通过 ALBBMANPageBuilder 构造出一条页面事件的日志 map，最后通过某个 ALBBMANTracker 埋点实例的 send API发送上传。

ALBBMANTracker是一个用于对埋点数据进行上报的工具，下文提到的ALBBMANPageHitBuilder等事件类都是通过ALBBMANTracker进行事件件上报的。其中ALBBMANTracker的获取如下所示：

```
// 获取默认ALBBMANTracker实例
ALBBMANTracker *tracker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
```

埋点数据上报都是通过ALBBMANTracker进行，我们可以设定/删除上报数据的全局字段，全局字段设定后在所有的上报日志中都可以查看，如下所示：

```
ALBBMANTracker *tracker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
// 设定全局字段
[tracker setGlobalProperty:@"globalKey1" value:@"globalValue1"];
// 删除全局字段
[tracker removeGlobalProperty:@"globalKey1"];
```

4.3.1 设置页面名称

接口：

```
- (void)setPageName:(NSString *)pPageName;
```

功能： 设置页面名称

入参： 页面名称，pPageName不能为空

是否必须调用： 是，在已经创建了 ALBBMANPageHitBuilder 实例后，必须调用该方法

调用时机： 创建 ALBBMANPageHitBuilder 后，必须调用 setPageName，否则调用 build 后返回为 nil

备注： 页面名称是页面事件的基础，必须设置

4.3.2 设置页面 refer

接口：

```
- (void)setReferPage:(NSString *)pReferPageName;
```

功能： 设置页面的 refer，即当前页面的来源页面

入参： refer 页面名称，需要在 setPageName 的 PageName 集合中，也可以为空

是否必须调用： 否

4.3.3 设置页面停留时间

接口：

```
- (void)setDurationOnPage:(long long)durationTimeOnPage;
```

功能： 记录页面从展现到页面离开的停留时间

入参： 页面停留时间

是否必须调用 否

调用时机： 需要记录当前页面停留时间

4.3.4 设置页面事件扩展参数

接口：

```
- (void)setProperty:(NSString *)pKey value:(NSString *)pValue;
```

功能：给单条日志添加一个扩展参数

入参：key 和 value 都不能为 nil，其中 key 不能为PAGE/EVENTID/ARG1/ARG2/ARG3/ARGS，否则 build 返回 nil

是否必须调用：否

调用时机：需要给ALBBMANPageHitBuilder实例添加扩展参数时

4.3.5 组装单条日志 map

接口：

```
- (NSDictionary *)build;
```

功能： 将塞入的参数组成 map 返回

入参： 无

是否必须调用： 否

调用时机： 创建 ALBBMANPageHitBuilder 或者 ALBBMANCustomHitBuilder 实例，塞入一些业务字段后，调用 build 组装业务字段生成日志 map，通过某个 ALBBMANTracker 埋点实例 send 接口发送

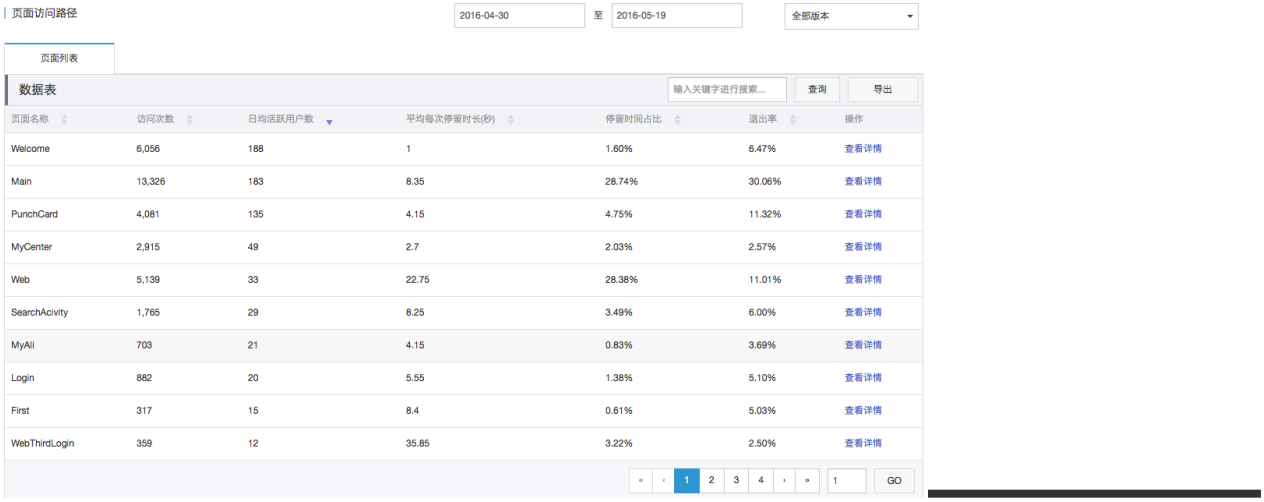
备注： build 函数返回的日志 map，必须通过ALBBMANTracker埋点上报工具的send接口上传数据完成打点

4.3.6 代码示例

```
ALBBMANPageHitBuilder *pageHitBuilder = [[ALBBMANPageHitBuilder alloc] init];
// 设置页面refer
[pageHitBuilder setReferPage:@"pageRefer"];
// 设置页面名称
[pageHitBuilder setPageName:@"pageName"];
// 设置页面停留时间
[pageHitBuilder setDurationOnPage:100];
// 设置页面事件扩展参数
[pageHitBuilder setProperty:@"pagePropertyKey1" value:@"pagePropertyValue1"];
```

```
[pageHitBuilder setProperty:@"pagePropertyKey2" value:@"pagePropertyValue2"];
ALBBMANTracker *tracker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
// 组装日志并发送
[tracker send:[pageHitBuilder build]];
```

上述的页面埋点与页面事件将影响控制台【页面路径分析】、【关键漏斗】、【控件点击】、【页面留存】等指标的报表展现，页面路径如下图所示。



5. 性能数据统计

APP性能直接决定了用户体验效果，同时也一直是业务开发人员容易忽视的问题。Mobile Analytics提供了丰富的性能监控API方便用户全方位监控自己的APP运行状态。

5.1 网络性能统计

Mobile Analytics针对传统APP的网络性能统计进行了封装，方便用户“一键埋点”，获取网络层面的关键技术指标，这些指标包括：

- 请求的整体响应时间
- 网络异常统计

5.1.1 接口及统计

从客户端角度来看，一个正常的网络请求分为以下几个阶段：tcp建连、请求发送、开始接收响应数据、接受数据完成。分阶段性能指标有：TCP建连时间、首字节时间、整体的响应时间。那么针对这几个阶段，我们分别提供有对应的API接口。

```
- (instancetype)initWithHost:(NSString *)host method:(NSString *)method;

/*
* 用户可以自行设置属性，但是不要包含“/Host/Method/EVENTID/PAGE/ARG1/ARG2/ARG3/ARGS/COMPRESS”
*/
- (void)setProperty:(NSString *)pKey value:(NSString *)pValue;
```

```
- (void)setproperties:(NSMutableDictionary *)properties;

/*
* 网络请求打点开始
*/
- (void)requestStart;
/*
* 打点标记建连成功
*/
- (void)connectFinished;

/*
* 打点标记首字节到达
*/
- (void)requestFirstBytes;

/*
* 打点正常结束
*/
- (void)requestEndWithBytes:(long)loadBytes;

/*
* 出错的时候上报
*/
- (void)requestEndWithError:(ALBBMANNetworkError *)error;

/*
* 整合打点日志
*/
- (NSDictionary *)build;
```

初始化ALBBMANNetworkHitBuilder的时候Host/Method是必须要的,如果传入Nil,数据将无法上报。下面是以NSURLConnection的实例:

```
 #import <AlicloudMobileAnalitics/ALBBMAN.h>

/**
* @brief
同步请求网络性能埋点
*/
- (void)syncNetworkHit {
NSURL *URL = [NSURL URLWithString:@"http://www.taobao.com/"];
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:URL
cachePolicy:NSURLRequestReloadIgnoringLocalCacheData
timeoutInterval:30];
// 由于NSURLConnection同步接口，封装了TCP连接、首字节的过程，所以同步接口上面我们统计不到TCP建连时间跟首字节时间
ALBBMANNetworkHitBuilder *builder = [[ALBBMANNetworkHitBuilder alloc] initWithHost:URL.host
method:[request HTTPMethod]];

// 开始请求打点
[builder requestStart];

NSHTTPURLResponse *response;
NSError *error;
```

```
NSData *data = [NSURLConnection sendSynchronousRequest:request returningResponse:&response error:&error];
if (error) {
// 自定义网络异常，见文档5.1.2
ALBBMANNetworkError *networkError = [[ALBBMANNetworkError alloc] initWithErrorCode:1001];
// 自定义网络异常附加信息
[networkError setProperty:@"IP" value:@"1.2.3.4"];
[builder requestEndWithError:networkError];
}
if (response.statusCode >= 400 && response.statusCode < 600) {
// 默认网络异常，见文档5.1.2
ALBBMANNetworkError *networkError;
if (response.statusCode < 500) {
networkError = [ALBBMANNetworkError ErrorWithHttpException4];
} else {
networkError = [ALBBMANNetworkError ErrorWithHttpException5];
}
// 默认网络异常附加信息
[networkError setProperty:@"IP" value:@"1.2.3.4"];
[builder requestEndWithError:networkError];
}
else {
//结束请求打点，bytes是下载的数据量大小
[builder requestEndWithBytes:data.length];
}
// 组装日志并发送
ALBBMANTracker *tracker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
[tracker send:[builder build]];
}

// 异步接口，相对同步接口的情况下,可以多统计到首字节,其他跟同步接口一样
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {
// 首字节响应埋点
[_builder requestFirstBytes];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
[_mutableData appendData:data];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
// 结束请求打点
[_builder requestEndWithBytes:_mutableData.length];
// 组装日志并发送
ALBBMANTracker *tracker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
[tracker send:[_builder build]];
}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {
ALBBMANNetworkError *networkError = [[ALBBMANNetworkError alloc] initWithErrorCode:[NSString
stringWithFormat:@"%ld", (long)error.code]];
[_builder requestEndWithError:networkError];
// 组装日志并发送
ALBBMANTracker *tracker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
[tracker send:[_builder build]];
}
```

```
//由于NSURLConnection封装性,已然看不到TCP连接过程,所以统计不到建连
//时间,如果是自实现的Socket的情况下,可以额外在TCP连接成功的时刻调用如下代码,统计tcp建连时间
[builder connectFinished];
```

注意，目前Method字段的value只支持POST和GET（不区分大小写）。

网络性能的地域分析如下图所示，

5.1.2 异常统计

网络请求无法避免异常情况，在埋点时，如果抛出异常打断了整个网络请求的流程，那么您是需要对此做额外处理的，否则将导致打点数据紊乱。处理的办法也很简便，您只需要在抓获异常以后选择网络默认异常类型和自定义网络异常类型的任意一种方式来标记这次请求发生了异常：

默认网络异常类型

异常类型	对应接口	说明
400<=HttpResponseCode<500	ErrorWithHttpException4	Http状态码4XX的错误
500<=HttpResponseCode	ErrorWithHttpException5	Http状态码5XX的错误
IO错误	ErrorWithIOInterrupted	请求的IO错误
Timeout错误	ErrorWithSocketTimeout	连接/请求超时的错误

请参考以上事例。

使用说明：

```
ALBBMANNetworkError *networkError = [ALBBMANNetworkError ErrorWithHttpException4];
```

```
// 网络异常附加信息
[networkError setProperty:@"IP" value:@"1.2.3.4"];
[builder requestEndWithError:networkError];
```

自定义网络异常类型

如果默认网络异常类型无法满足您的需求，您可以自定义网络异常类型，但是网络异常类型编码要在1001 <= YouErrorCode <= 1010范围内，如果不进行前端配置将以错误编码(10xx)的形式进行展示，如果在前端配置错误编码对应的信息，就会展示您所配置的信息。

异常类型	对应接口	说明
自定义错误	initWithErrorCode	默认网络错误不可描述时,可自定义

使用说明：

```
ALBBMANNetworkError *networkError = [[ALBBMANNetworkError alloc] initWithErrorCode:1001];
// 自定义网络异常附加信息
[networkError setProperty:@"IP" value:@"1.2.3.4"];
[builder requestEndWithError:networkError];
```

5.2 自定义性能事件埋点

自定义性能事件埋点可满足用户对定制化的性能事件进行实时监控的需求(比如，Cache的读写耗时，私有网络协议的RPC耗时，客户端算法的耗时等）。在进行正确的客户端埋点后，您可通过阿里云控制台进行数据的多维度实时监控。

自定义性能事件可包含以下几部分内容：

1.事件名称（event_label）,只能为字母、数字和下划线组成【必选】

2.事件从开始到完成消耗的时长【必选】

3.事件所携带的属性【可选】

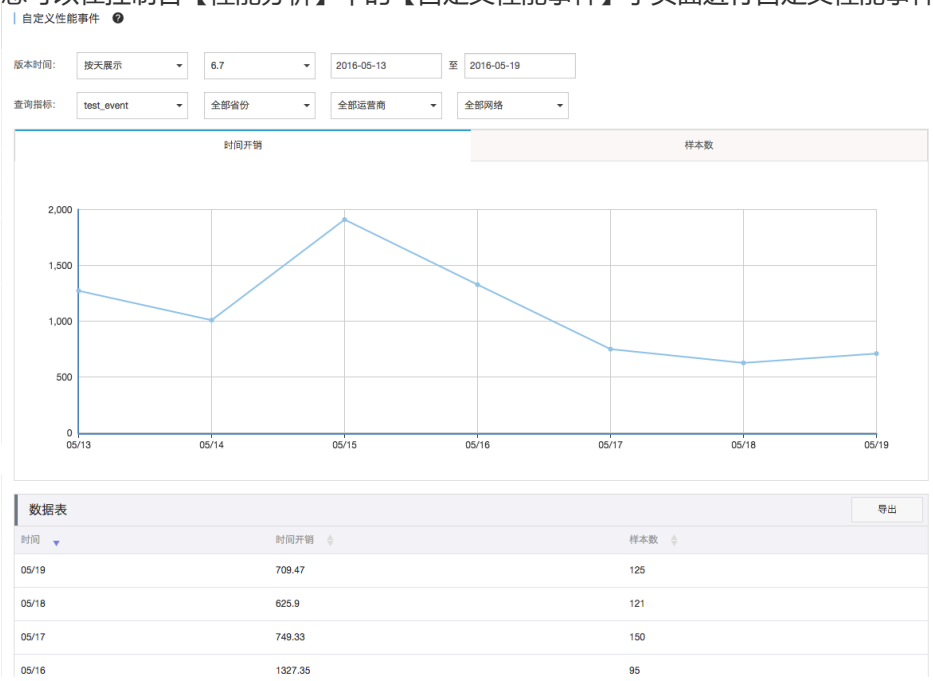
【注意】 自定义性能事件用于监控用户事件的时间开销，请参考下述代码实例正确传递参数。

例子：

```
ALBBMANCustomPerformanceHitBuilder *customPerfBuilder = [[ALBBMANCustomPerformanceHitBuilder alloc]
init:@"HomeActivityInit"];
// 记录事件时间方式1，自定义性能事件开始
[customPerfBuilder hitStart];
...
// 自定义性能事件结束
[customPerfBuilder hitEnd];
// 设置定义性能事件持续时间，方式2
// [customPerfBuilder setDurationIntervalInMillis:1234];
// 设置扩展参数
[customPerfBuilder setProperty:@"Page" value:@"Home"];
```

```
// 组装日志并发送
ALBBMANTracker *traker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
[traker send:[customPerfBuilder build]];
```

完成上述埋点后您可以在控制台【性能分析】中的【自定义性能事件】子页面进行自定义性能事件的实时监控，如下图所示。



5.3 crashHandler

Mobile Analytics 初始化的时候默认会打开 crashHandler，即自动捕获应用 crash，然后发送 crash 日志事件。

【注意】如果您的程序中设置了crash捕获，您的crash捕获设置会和Mobile Analytics中的crashHandler相互覆盖，覆盖结果因两者运行的先后顺序的不同而异。

5.3.1 关闭 crashHandler

接口:

```
- (void)turnOffCrashHandler;
```

功能: 关闭 Mobile Analytics 的自动捕获应用 crash

是否必须调用: 否

调用时机: 初始化SDK时调用

6. 自定义事件

自定义事件埋点可用于满足用户的定制化需求。

6.1 设置自定义事件的标签

接口：

```
- (void)setEventLabel:(NSString *)pEventId;
```

功能： 区分不同自定义事件的标签，同一种自定义事件的 pEventId 相同

入参： 自定义事件标签，相当于自定义事件的业务 ID，不能为空

是否必须调用： 是

调用时机： 创建 ALBBMANCustomHitBuilder 后，必须调用 setEventLabel，否则调用 build 后返回为 nil

备注： 对于自定义事件，必须设置标签，pEventId需要事先在wdm上申请

6.2 设置自定义事件的页面名称

接口：

```
- (void)setEventPage:(NSString *)pPageName;
```

功能： 设置该自定义事件发生在哪个页面

入参： 自定义事件的页面名称，可以为空，这种情况日志中默认页面名称为 “UT”

是否必须调用： 否

调用时机： 需要明确该自定义事件发生时的页面，不调用情况下默认为 “UT”

6.3 设置自定义事件停留时间

接口：

```
- (void)setDurationOnEvent:(long long)durationOnEvent;
```

功能： 设置自定义事件持续时间，跟 ALBBMANPageHitBuilder 中的页面停留时间类似

入参： 自定义事件停留时间

是否必须调用： 否

调用时机： 需要记录自定义事件的停留时间

6.4 设置自定义事件扩展参数

接口：

```
- (void)setProperty:(NSString *)pKey value:(NSString *)pValue;
```

功能：给单条日志添加一个扩展参数

入参：key 和 value 都不能为 nil，其中 key 不能为PAGE/EVENTID/ARG1/ARG2/ARG3/ARGS，否则 build 返回 nil

是否必须调用：否

调用时机：需要给ALBBMANCustomHitBuilder实例添加扩展参数时

6.5 代码示例

```
ALBBMANCustomHitBuilder *customBuilder = [[ALBBMANCustomHitBuilder alloc] init];
// 设置自定义事件标签
[customBuilder setEventLabel:@"test_event_label"];
// 设置自定义事件页面名称
[customBuilder setEventPage:@"test_Page"];
// 设置自定义事件持续时间
[customBuilder setDurationOnEvent:12345];
// 设置自定义事件扩展参数
[customBuilder setProperty:@"ckey0" value:@"value0"];
[customBuilder setProperty:@"ckey1" value:@"value1"];
[customBuilder setProperty:@"ckey2" value:@"value2"];
ALBBMANTracker *traker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
// 组装日志并发送
NSDictionary *dic = [customBuilder build];
[traker send:dic];
```

自定义事件扩展参数在控制台【自定义事件】-【详细数据】-【参数分析】中可查看，但查看之前请在【管理设置】-【自定义事件管理】中添加要在控制台显示的事件ID。 如果您需要对自定义事件进行实时监控，请参考【5.3 自定义性能事件】章节。

7. 如何实时验证数据是否正常上报

您当前可以通过以下两种方式进行验证：

- 打开移动数据分析log，查看是否会出现如***** UTMCEngine <info> ***** UTMCUploader:upload :: upload response:{"t":1464532326870,"ret":"","success":"success"}的日志；

  注意：SDK的日志上报会有缓存和聚合，因此上报时机会比API调用时机滞后一些，可耐心等待30-60s或将应用切到后台查看。

- 登录控制台查看活跃用户等实时报表；

  注意：活跃用户的统计依赖页面埋点，移动数据分析后台会将零星的页面埋点处理为噪点，进行过滤，因此请确保您有足量的页面埋点事件上报（>5），另外控制台的实时报表大概会有

5min的延迟。

8. H5页面数据的采集

H5页面采集并没有单独的SDK，依赖native进行上传，通过JSBridge通知给native，然后调用MAN的相应方法，进行数据的上报。可运行demo请参考：alicloud-ios-demo

8.1 代码示例

H5进行自定义事件的上报：

JavaScript代码：

```
// 通过iframe发起请求，然后在native端进行捕获。
var iframe = document.createElement("IFRAME");
// 通过自定义scheme，来判断是正常请求还是H5通信。
// 参数可以放在url中，然后在native解析。
iframe.setAttribute("src", "jsbridge://custom");
document.documentElement.appendChild(iframe);
iframe.parentNode.removeChild(iframe);
iframe = null;
```

Objective-c

```
- (BOOL) webView:(UIWebView *)webView
shouldStartLoadWithRequest:(NSURLRequest *)request
navigationType:(UIWebViewNavigationType)navigationType
{
NSURL *url = [request URL];
NSString *scheme = [url scheme];

// js通信
if ( [@"jsbridge" isEqualToString:scheme] ) {
if ( [@"custom" isEqualToString:url.host] ) {
ALBBMANCustomHitBuilder *customBuilder = [[ALBBMANCustomHitBuilder alloc] init];
// 设置自定义事件标签
[customBuilder setEventLabel:@"test_event_label"];
// 设置自定义事件页面名称
[customBuilder setEventPage:@"test_Page"];
// 设置自定义事件持续时间
[customBuilder setDurationOnEvent:12345];
// 设置自定义事件扩展参数
[customBuilder setProperty:@"ckey0" value:@"value0"];
[customBuilder setProperty:@"ckey1" value:@"value1"];
[customBuilder setProperty:@"ckey2" value:@"value2"];
ALBBMANTracker *traker = [[ALBBMANAnalytics getInstance] getDefaultTracker];
// 组装日志并发送
NSDictionary *dic = [customBuilder build];
[traker send:dic];
```

```
}
return NO;
}

return YES;
}
```

Android SDK手册

移动数据分析

Mobile Analytics Android SDK开发指南

1. 前言

本文档介绍了移动数据分析(Mobile Analytics) Android SDK的使用方式。

Mobile Analytics Android SDK是阿里云面向移动开发者提供的Android平台下的数据统计与监控服务。通过该SDK，开发者可以在自己的APP中便捷地进行数据埋点，监控日常的业务数据与网络性能数据，并通过阿里云控制台界面观察对应的数据报表展现。另外，用户后续可以通过设定自定义的数据解析规则实现定制化的数据图表展现。

您可以通过获取alicloud-android-demo工程源码获得移动数据分析服务的使用例程。

2. 安装Mobile Analytics Android SDK

2.1 注意

使用1.1.5及之前版本请在**【Crash分析】**板块查看crash信息。

使用1.1.6版本及之后的版本，请在**【新版Crash分析】**板块查看crash信息。

推荐使用1.1.6及之后的版本，crash数据更加准确，丢包率更小。

1.1.6之后的版本如果用手动设置channel的方式，请在manService.getMANAnalytics().init方法之前调用，具体请看下方初始化代码。通过mainfest.xml设置的话不影响。

2.2 手动集成SDK

2.2.1 SDK目录结构

```
OneSDK
|-- libs
|-- |-- jniLibs
| | |-- armeabi
| | | |-- libMotu.so -crash捕获的so包
| | |-- armeabi-v7a
| | | |-- libMotu.so
| | |-- x86
| | | |-- libMotu.so
| |-- alicloud-android-sdk-man-1.1.6.jar -移动数据分析主功能包
| |-- alicloud-android-ut-2.6.0.jar -UT基础包
| |-- utdid4all-1.1.5.3_proguard.jar -设备Id生成包
```

2.2.2 SDK集成

- 手动拷贝OneSDK目录下的jniLibs到以下目录：src - > main
- 在build.gradle配置中添加如下配置项：

```
android {
...
defaultConfig {
...
ndk {
moduleName "jniLibs"
abiFilters "armeabi", "armeabi-v7a", "x86"
}
}
}
```

2.3 Maven依赖

- build.gradle中添加Maven仓库地址：

```
allprojects {
repositories {
maven {
url 'http://maven.aliyun.com/nexus/content/repositories/releases/'
}
}
}
```

- gradle添加依赖：

```
```

```
dependencies {
compile 'com.aliyun.ams:alicloud-android-man:1.2.0'
}
```

（开发时可以如上所述指定完整的版本号，也可以指定模糊版本号，gradle自动拉取满足条件的最新版本SDK，如compile 'com.aliyun.ams:alicloud-android-man:1.+'）

2.4 EMAS产品统一接入

要求sdk版本>=1.2.2版本，使用统一接入方式后，将无需在AndroidManifest中指定appKey / appSecret，并且初始化时，可直接使用manService.getMANAnalytics().init(this, getApplicationContext());即可。具体请参考：Emas统一接入文档(Android)

3. 应用程序初始化

在您使用Mobile Analytics Android SDK进行数据统计与监控前，您需要对SDK的上下文进行一些初始化配置，如权限声明、传递应用上下文、访问控制等。其中权限声明在AndroidManifest.xml文件中进行。

3.1 权限声明及配置AppKey,AppSecret

以下是Mobile Analytics Android SDK所需要的Android权限及配置AppKey,AppSecret，请把这些权限配置到您的AndroidManifest.xml文件，否则，SDK将无法正常工作。

```
...
<!-- 若您使用上述2.3中"统一接入的方式，则无需在AndroidManifest中配置appKey / appSecret" -->
<meta-data android:name="com.alibaba.app.appkey" android:value="YourAppKey"></meta-data>
<meta-data android:name="com.alibaba.app.appsecret" android:value="YourAppSecret"></meta-data>
</application>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
<uses-permission android:name="android.permission.GET_TASKS"></uses-permission>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
<uses-permission android:name="android.permission.READ_SETTINGS"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

3.2 初始化及参数设置示例

在Application的实现类中，添加初始化SDK的代码。

Mobile Analytics Android SDK初始化部分的接口如下：

```
public class YourApplication extends Application {
```

```
@Override
public void onCreate() {
super.onCreate();

/* 【注意】建议您在Application中初始化MAN，以保证正常获取MANService*/

// 获取MAN服务
MANService manService = MANServiceProvider.getService();

// 打开调试日志，线上版本建议关闭
// manService.getMANAnalytics().turnOnDebug();

// 若需要关闭 SDK 的自动异常捕获功能可进行如下操作(如需关闭crash report，建议在init方法调用前关闭crash),详见文档5.4
manService.getMANAnalytics().turnOffCrashReporter();

// 设置渠道（用以标记该app的分发渠道名称），如果不关心可以不设置即不调用该接口，渠道设置将影响控制台【渠道分析】栏目的报表展现。如果文档3.3章节更能满足您渠道配置的需求，就不要调用此方法，按照3.3进行配置即可；1.1.6版本及之后的版本，请在init方法之前调用此方法设置channel.
manService.getMANAnalytics().setChannel("某渠道");

// MAN初始化方法之一，从AndroidManifest.xml中获取appKey和appSecret初始化，若您采用上述 2.3中"统一接入的方式"，则使用当前init方法即可。
manService.getMANAnalytics().init(this, getApplicationContext());

// MAN另一初始化方法，手动指定appKey和appSecret
// 若您采用上述2.3中"统一接入的方式"，则无需使用当前init方法。
// String appKey = "******";
// String appSecret = "******";
// manService.getMANAnalytics().init(this, getApplicationContext(), appKey, appSecret);

// 通过此接口关闭页面自动打点功能，详见文档4.2
manService.getMANAnalytics().turnOffAutoPageTrack();

// 若AndroidManifest.xml 中的 android:versionName 不能满足需求，可在此指定
// 若在上述两个地方均没有设置appversion，上报的字段默认为null
manService.getMANAnalytics().setAppVersion("3.1.1");
}
}
```

3.3 配置渠道信息

您可以在AndroidManifest.xml中配置您的渠道信息，您只需要将<YOUR CHANNEL ID>替换您的渠道信息即可。

【注意】SDK执行初始化时会自动获取AndroidManifest.xml中的字段，并到填充渠道字段；初始化完成后。若同时调用了setChannel方法，则以setChannel方法中的参数为准。

```
<application ...
<meta-data
android:name="ALIYUN_MAN_CHANNEL"
android:value="<YOUR CHANNEL ID>" >
</meta-data>
</application>
```

3.4 SDK调试说明

在控制台中观察到的【今日实时】、【系统质量】-【实时Crash信息】、【新版crash分析】、【系统质量】-【性能分析】部分均为实时数据，调试时可参考该数据，验证环境配置及初始化是否正确。数据统计的准确性依赖APP的常规生命轨迹，比如应用启动次数依赖于用户正常退出应用触发的上报策略。

4. 业务数据统计

4.1 会员账号信息埋点

4.1.1 用户注册埋点

在用户注册成功之后，可使用userRegister 完成用户注册埋点。

```
MANService manService = MANServiceProvider.getService();
// 注册用户埋点
manService.getMANAnalytics().userRegister("usernick");
```

4.1.2 用户登录及注销埋点

用户登录埋点：

```
MANService manService = MANServiceProvider.getService();
// 用户登录埋点
manService.getMANAnalytics().updateUserAccount("usernick", "userid");
```

用户注销埋点：

```
// 用户注销埋点
manService.getMANAnalytics().updateUserAccount("", "");
```

如果不进行 4.1 会员账号信息埋点，此时不能在今日实时里看到【登录会员】和【新注册会员】的统计信息，而设备相关的统计【活跃用户】和【新增用户】可以看到。

完成上述埋点后，您就可以在阿里云控制台看到相应统计信息，例如下图所示为用户周活跃度。

4.2 页面埋点

说明：Mobile Analytics SDK默认会自动采集Android 4.0及以上系统的Activity 页面，如果不需要自动采集可使用下面方法关闭自动页面打点。打开页面自动埋点时，默认页面名称为class.getSimpleName()并去除Activity后缀。

```
// 关闭自动打点
MANService manService = MANServiceProvider.getService();
manService.getMANAnalytics().turnOffAutoPageTrack();
```

为了满足Android 4.0以下系统的页面采集需求，您可按照如下的说明进行手动埋点。

【注意】：如果App中没有进行页面埋点，活跃用户参数不能正常统计。

4.2.1 手动Activity页面埋点

在Activity 的onReasume 以及onPause 中分别加入pageAppear和pageDisAppear代码，建议下述代码可以在一个基类中做，让其它所有的activity类都继承这个基类，就完成了所有子类页面埋点。

代码示例如下：

```
public class BaseActivity extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
}

@Override
protected void onResume() {
super.onResume();
MANService manService = MANServiceProvider.getService();
manService.getMANPageHitHelper().pageAppear(this);
}

@Override
protected void onPause() {
```

```
super.onPause();
MANService manService = MANServiceProvider.getService();
manService.getMANPageHitHelper().pageDisAppear(this);
}
}
```

页面埋点将影响控制台【页面路径分析】、【关键漏斗】、【页面留存】等指标的报表展现。

4.2.2 给Activity页面增加属性统计

场景例子：

我们要开发一个购物app，在购物的app的宝贝展示页面，我们可能想要知道，此页面都展示了哪些商品？可以简单理解为，我们可以对采集的页面（GoodsDetails 页面）记录上增加一个宝贝属性就可以，如item_id=xxxxxx。

代码示例如下：

```
// 这句话要在一个页面的onPause 之前任何位置调用都可以
Map<String, String> lMap = new HashMap<String, String>();
lMap.put("item_id", "xxxxxx");
MANService manService = MANServiceProvider.getService();
manService.getMANPageHitHelper().updatePageProperties(lMap);
```

结果Log：

```
11-06 16:10:32.088: I/cache_log(10879): UT:...||2001||-||-||3191||item_id=xxxxxx
```

这里：2001 是页面事件的ID，3191 是页面展示的时长，item_id=xxxxxx 就是属性。

4.2.3 页面基础埋点使用

上述是针对Activity级别的页面埋点，只需简单地调用pageAppear和pageDispear即可完成埋点对相应的数据上报，上报数据包括：页面名称、来源页面名称、页面停留时间和额外属性设置等。如果需要对非Activity页面，如Fragment进行埋点，或者需要灵活把控页面埋点上报信息，如修改上报页面信息、对页面停留时长做特殊处理等，可以使用页面基础埋点的方式。通过该方式上报埋点数据同样将影响控制台【页面路径分析】、【关键漏斗】、【页面留存】等指标的报表展现。使用方式如下：

获取基础页面打点对象:

```
// 传入参数为页面名称
MANPageHitBuilder pageHitBuilder = new MANPageHitBuilder(String pageName);
```

设置来源页面名称（注：referPageName需要在pageName的集合中）：

```
pageHitBuilder.setReferPage(String referPageName);
```

设置页面停留时间：

```
pageHitBuilder.setDurationOnPage(long duration);
```

设置页面属性：

```
pageHitBuilder.setProperty(String key, String value);
pageHitBuilder.setProperties(Map<String, String> properties);
```

构造页面埋点log数据：

```
pageHitBuilder.build();
```

数据上报：

```
MANServiceProvider.getService().getMANAnalytics().getDefaultTracker().send(pageHitBuilder.build());
```

5. 性能数据统计

APP性能直接决定了用户体验效果，同时也一直是业务开发人员容易忽视的问题。Mobile Analytics提供了丰富的性能监控API方便用户全方位监控自己的APP运行状态。

5.1 基本网络性能统计

Mobile Analytics针对传统APP的网络性能统计进行了封装，方便用户埋点，获取网络层面的常见指标，这些指标包括：

- 请求的整体响应时间
- APP整体带宽利用率
- 网络异常统计

在进行正确地数据埋点上报后，您可以在阿里云控制台Mobile Analytics监控页面观察您的APP的相应性能指标情况。

5.1.1 开始网络请求

Mobile Analytics对网络性能的统计以一次网络请求为基本单位，您需要在这次网络请求的流程中埋入一些记录点，让Mobile Analytics统计这次请求的相关数据，上报到数据分析中心。具体方法如下：

对某个网络请求的统计通过MANNetworkPerformanceHitBuilder完成，在发起网络请求前，您需要调用如下代码，记录此次网络请求的Host信息和请求Method，并标记网络请求开始：

```
MANNetworkPerformanceHitBuilder networkPerfBuilder = new MANNetworkPerformanceHitBuilder("taobao.com",
"GET");
```

```
/* 打点记录网络请求开始 */
networkPerfBuilder.hitRequestStart();
```

此外，Mobile Analytics Android SDK还提供了可以附加额外信息的接口：

```
/* 附带额外需要上报信息 */
networkPerfBuilder.withExtraInfo("name1", "value1");
```

【注意】Host必须符合标准Hostname格式，否则会被忽略。Method必须是“GET”或者“POST”，否则默认为“GET”，目前只支持这两个值。我们在前台展示时会用这些信息中的’Host’，‘Method’作为维度。

5.1.2 网络请求正常结束，统计请求响应时间

请求结束后，您还需要做一次埋点，来标记该次网络请求的结束。接口如下：

```
networkPerfBuilder.hitRequestEndWithLoadBytes(loadBytes);
```

类型为int的loadBytes参数是为了让您传入该次网络请求的总下载数据量，以便Mobile Analytics Android SDK打点记录。

在您调用这个函数以后，整个网络性能埋点流程就结束了，该次网络请求的相关数据会上报到数据分析平台，分析汇总后，您就可以到阿里云控制台Mobile Analytics页面观察您的APP的相应性能指标情况,如下图所示。

日期	次均TCP建连时间(ms)	次均首字节抵达时间(ms)	次均请求完成时间(ms)	次均请求资源大小（byte）	请求样本总数
03/24	106.15	215.84	321.08	1245.36	13011
03/25	111.26	220.36	333.66	1043.49	10288526
03/26	93.92	187.89	281.96	1055.26	8519270
03/27	81.74	171.15	252.73	1708.09	371131
03/28	455.9	1367.24	2025.27	24437.38	11239
03/29	204.69	608.45	776.29	18929	1432

5.1.3 网络请求异常结束，上报网络异常

网络请求无法避免异常情况，在埋点时，如果抛出异常打断了整个网络请求的流程，那么您是需要对此做额外处理的，否则将导致打点数据紊乱。上报的办法也很简便，您只需要在抓获异常以后选择默认网络异常类型或者自定义网络异常类型的任意一种方式来标记这次请求发生了异常：

默认网络异常类型

异常类型或HttpResponseCode	对应接口
400<=HttpResponseCode<500	buildHttpCodeClientError4XX()
500<=HttpResponseCode	buildHttpCodeServerError5XX()
MalformedURLException	buildMalformedURLException()
InterruptedIOException	buildInterruptedIOException()
SocketTimeoutException	buildSocketTimeoutException()
IOException	buildIOException()

使用说明：

```
// 如果您需要上报其它异常，请按照上表替换buildIOException为相应接口
MANNetworkErrorInfo errorInfo = MANNetworkErrorCodeBuilder.buildIOException()
.withExtraInfo("error_url", url.toString())
.withExtraInfo("other_info", "value");
networkPerfBuilder.hitRequestEndWithError(errorInfo);
```

自定义网络异常类型

如果默认网络异常类型无法满足您的需求，您可以自定义网络异常类型，但是网络异常类型编码要在 1001 <= YourErrorCode <= 1010范围内,如果不进行前端配置将以错误编码（10xx）的形式进行展示，如果在前端配置错误编码对应的信息，就会展示您所配置的信息。

使用说明：

```
MANNetworkErrorInfo errorInfo  = MANNetworkErrorCodeBuilder.buildCustomErrorCode(1001)
.withExtraInfo("error_url", url.toString());
networkPerfBuilder.hitRequestEndWithError(errorInfo);
```

5.1.5节中示例代码的最后部分已经展示了具体如何操作。这个接口的errorInfo参数是为了让您上报这次异常的一些您认为有价值的信息。

【注意】 一旦您调用了MANNetworkPerformanceHitBuilder的networkPerfBuilder.hitRequestStart();接口以后，您需要确认代码在正常情况下执行到networkPerfBuilder.hitRequestEndWithLoadBytes(loadBytes);，或者在异常请求下执行到networkPerfBuilder.hitRequestEndWithError(errorInfo);，否则，此次网络请求的相关信息将上报失败。

5.1.4 调用Tracker的send方法上报打点数据

使用builder进行各个阶段的打点以后，需要获取到MANTracker将它的数据上报，如下：

```
manService.getMANAnalytics().getDefaultTracker().send(networkEvent); // 调用Track的send方法把此次打点的数据上报
```

5.1.5 代码示例

以下是一个较为完善的示例代码：

```
MANService manService = MANServiceProvider.getService();
String urlString = "http://www.aliyun.com";
MANNetworkPerformanceHitBuilder networkPerformanceHitBuilder = new
MANNetworkPerformanceHitBuilder("www.aliyun.com", "GET");

try {
URL url = new URL(urlString);
byte[] buf = new byte[64 * 1024];

// 打点记录请求开始
networkPerformanceHitBuilder.hitRequestStart();
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
// 建立连接
conn.connect();
// 开始获取响应内容
int responseCode = conn.getResponseCode();
long totalBytes = 0;
int len = 0;
if (responseCode == 200) {
// 读尽响应内容
InputStream in = conn.getInputStream();
while ((len = in.read(buf)) != -1) {
totalBytes += len;
}
in.close();
}
// 附带额外需要上报信息
networkPerformanceHitBuilder.withExtraInfo("name1", "value1");
// 打点标记请求结束
networkPerformanceHitBuilder.hitRequestEndWithLoadBytes(totalBytes);
} catch (IOException e) {
e.printStackTrace();
// 按照文档5.1.3的说明选择默认网络异常类型或者自定义网络异常类型来上报网络异常
MANNetworkErrorInfo errorInfo = MANNetworkErrorCodeBuilder.buildIOException()
.withExtraInfo("error_url", urlString)
.withExtraInfo("other_info", "value");
// MANNetworkErrorInfo errorInfo = MANNetworkErrorCodeBuilder.buildCustomErrorCode(1001)
// .withExtraInfo("error_url", urlString)
// .withExtraInfo("other_info", "value");
// 打点，记录出错情况
networkPerformanceHitBuilder.hitRequestEndWithError(errorInfo);
}
// 上报网络性能事件打点数据
manService.getMANAnalytics().getDefaultTracker().send(networkPerformanceHitBuilder.build());
```

5.2 高级网络性能统计

如果您对网络指标有更细粒度的要求，并且已经按照5.1的文档说明进行了网络性能埋点，此时如果按照本章进行操作，便可以增加TCP建连时间及首字节响应时间的埋点。

- TCP建连时间（5.2 高级网络指标统计）
- 请求的首字节响应时间（5.2 高级网络指标统计）
- 请求的整体响应时间（见5.1 基本网络性能统计）
- APP整体带宽利用率（见5.1 基本网络性能统计）
- 网络异常统计（见5.1 基本网络性能统计）

5.2.1 TCP建连时间统计

如果您希望统计该次网络请求TCP建连的耗时，那么在发起网络请求以后，建连成功时，您需要调用以下接口标记建连成功：

```
/*打点记录建连成功*/
networkPerfBuilder.hitConnectFinished();
```

这个函数一般在httpURLConnection.connect()方法或者httpClient.execute()方法执行完毕时调用。

5.2.2 请求首字节响应时间统计

如果您希望统计该次网络请求的首字节响应时间，您需要调用以下接口标记首字节到达：

```
networkPerfBuilder.hitRecievedFirstByte();
```

这个时间点一般是在httpURLConnection.getResponseCode()方法或者HttpResponse.getStatusLine.getStatusCode()方法执行完毕以后。

5.2.3 代码示例

下面是增加TCP建连时间及首字节响应时间埋点后的示例代码：

```
MANService manService = MANServiceProvider.getService();
String urlString = "http://www.aliyun.com";
MANNetworkPerformanceHitBuilder networkPerformanceHitBuilder = new
MANNetworkPerformanceHitBuilder("www.aliyun.com", "GET");

try {
URL url = new URL(urlString);
byte[] buf = new byte[64 * 1024];

// 打点记录请求开始
networkPerformanceHitBuilder.hitRequestStart();
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
// 建立连接
conn.connect();
```

```
// 打点记录建连时间
networkPerformanceHitBuilder.hitConnectFinished();
// 开始获取响应内容
int responseCode = conn.getResponseCode();
// 打点记录首包时间
networkPerformanceHitBuilder.hitRecievedFirstByte();
long totalBytes = 0;
int len = 0;
if (responseCode == 200) {
// 读尽响应内容
InputStream in = conn.getInputStream();
while ((len = in.read(buf)) != -1) {
totalBytes += len;
}
in.close();
}
// 附带额外需要上报信息
networkPerformanceHitBuilder.withExtraInfo("name1", "value1");
// 打点标记请求结束
networkPerformanceHitBuilder.hitRequestEndWithLoadBytes(totalBytes);
} catch (IOException e) {
e.printStackTrace();
// 按照文档5.1.3的说明选择默认网络异常类型或者自定义网络异常类型来上报网络异常
MANNetworkErrorInfo errorInfo = MANNetworkErrorCodeBuilder.buildIOException()
.withExtraInfo("error_url", urlString)
.withExtraInfo("other_info", "value");
// MANNetworkErrorInfo errorInfo = MANNetworkErrorCodeBuilder.buildCustomErrorCode(1001)
// .withExtraInfo("error_url", urlString)
// .withExtraInfo("other_info", "value");
// 打点，记录出错情况
networkPerformanceHitBuilder.hitRequestEndWithError(errorInfo);
}
// 上报网络性能事件打点数据
manService.getMANAnalytics().getDefaultTracker().send(networkPerformanceHitBuilder.build());
```

5.3 自定义性能事件埋点

自定义性能事件埋点可满足用户针对定制化的性能事件进行实时监控的需求（比如，Cache的读写耗时，私有网络协议的RPC耗时，客户端算法的耗时等）。在进行正确的客户端埋点后，您可通过阿里云控制台进行数据的多维度实时监控。

自定义性能事件可包含以下几部分内容：

1.事件名称（event_label）,只能为字母、数字和下划线组成【必选】

2.事件从开始到完成消耗的时长【必选】

3.事件所携带的属性【可选】

【注意】 自定义性能事件用于监控用户事件的时间开销，请参考下述代码实例正确传递参数。

例子：

```
String labelKey = "fibonacci";
```

```
MANCustomPerformanceHitBuilder performanceHitBuilder = new MANCustomPerformanceHitBuilder(labelKey);
// 记录自定义性能事件起始时间
performanceHitBuilder.hitStart();
fibonacci(1);
// 记录自定义性能事件结束时间
performanceHitBuilder.hitEnd();
// 设置时长方法2
// long timeCost = 0;
// performanceHitBuilder.setDuration(timeCost);
performanceHitBuilder.withExtraInfo("EXTRA_INFO_KEY1", "EXTRA_INFO_VALUE")
.withExtraInfo("EXTRA_INFO_KEY2", "EXTRA_INFO_VALUE");
// 上报自定义性能事件打点数据
manService.getMANAnalytics().sendCustomPerformance(performanceHitBuilder.build());
```

5.4 Crash 异常捕获埋点

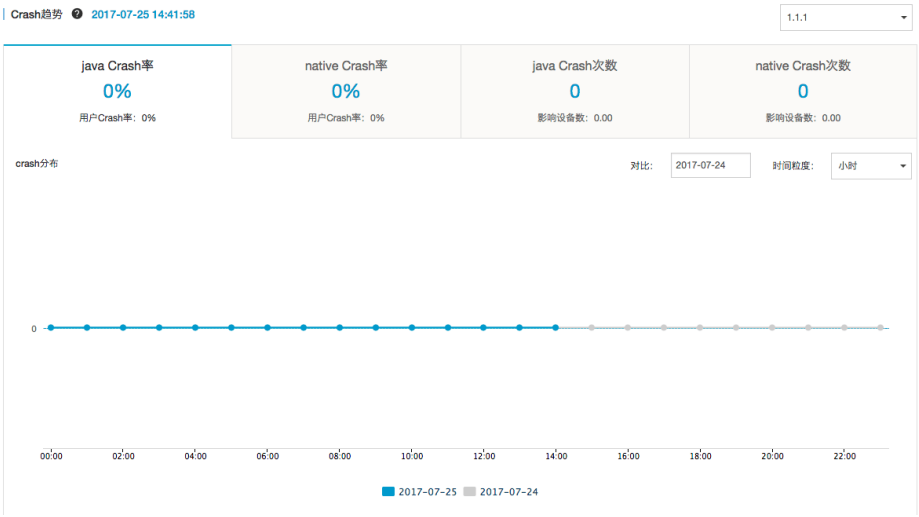
由于CrashHandler（app crash数据采集）是默认开启的，如果您不需要，可以通过如下方式关闭(建议在init()方法前，调用turnoff方法关闭)

```
MANService manService = MANServiceProvider.getService();
// 关闭crash自动采集功能
manService.getMANAnalytics().turnOffCrashReporter();
```

【注意】 这段代码建议与初始化SDK代码放置在一起，在程序启动部分执行。

【注意】 如果您的代码中使用了Thread.setDefaultUncaughtExceptionHandler(handler)，请在初始化MAN SDK之前调用Thread.setDefaultUncaughtExceptionHandler(handler)。这样MAN收到异常通知以后，会再次通知到您的handler,

如果使用了crash采集功能，可在前端界面实时看到crash相关信息，如下图所示。



6 自定义事件埋点

自定义事件埋点可用于满足用户的定制化需求。

自定义事件可包含以下几个部分内容：

1.事件名称（event_label）

2.事件从开始到完成消耗的时长

3.事件所携带的属性

4.事件对应的页面

例子：

```
// 事件名称：play_music
MANCustomHitBuilder hitBuilder = new MANHitBuilders.MANCustomHitBuilder("playmusic");

// 可使用如下接口设置时长：3分钟
hitBuilder.setDurationOnEvent(3 * 60 * 1000);

// 设置关联的页面名称：聆听
hitBuilder.setEventPage("Listen");
// 设置属性：类型摇滚
hitBuilder.setProperty("type", "rock");
// 设置属性：歌曲标题
hitBuilder.setProperty("title", "wonderful tonight");
// 发送自定义事件打点
MANService manService = MANServiceProvider.getService();
manService.getMANAnalytics().getDefaultTracker().send(hitBuilder.build());
```

自定义事件扩展参数在控制台【自定义事件】-【详细数据】-【参数分析】中可查看，但查看之前请在【管理设置】-【自定义事件管理】中添加要在控制台显示的事件ID。如果您需要对自定义事件进行实时监控，请参考【5.3 自定义性能事件】章节。

7. 如何实时验证数据是否正常上报

您当前可以通过以下两种方式进行验证：

- 打开移动数据分析log，查看logcat是否会出现如
  UT:{"t":1464532680574,"ret":"","success":"success"}的日志；

  注意：SDK的日志上报会有缓存和聚合，因此上报时机会比API调用时机滞后一些，可耐心等待30-60s或将应用切到后台查看。

- 登录控制台查看活跃用户等实时报表；

  注意：活跃用户的统计依赖页面埋点，移动数据分析后台会将零星的页面埋点处理为噪点，进行过滤，因此请确保您有足量的页面埋点事件上报（>5），另外控制台的实时报表大概会有5min的延迟。

8. H5数据的上报

H5页面采集并没有单独的SDK，依赖native进行上传，通过JSBridge通知给native，然后调用MAN的相应方法，进行数据的上报。可运行demo请参考：alicloud-android-demo

8.1 代码示例

JavaScript代码：

```
// 这里通过在JS alert消息，然后在native端进行捕获通信
alert( "jsbridge://custom" );
```

Java

重写WebChromeClient的onAlert方法，对于约定好的scheme，进行拦截，然后根据相应内容执行不同的方法，从而达到调用native埋点的目的。

```
WebView webView = (WebView) findViewById(R.id.h5DemoWebview);
webView.setWebChromeClient(new WebChromeClient(){
@Override
public boolean onJsAlert(WebView view, String url, String message, JsResult result) {
// 这里只要自己约定好方式就行，uri、json、xml等等。
Uri uri = Uri.parse(message);

if ( uri.getScheme().equals( "jsbridge" ) ) {
if ( uri.getHost().equals( "custom" ) ) {
// 事件名称：play_music
MANHitBuilders.MANCustomHitBuilder hitBuilder = new MANHitBuilders.MANCustomHitBuilder("playmusic");
// 可使用如下接口设置时长：3分钟
hitBuilder.setDurationOnEvent(3 * 60 * 1000);
```

```
// 设置关联的页面名称：聆听
hitBuilder.setEventPage("Listen");
// 设置属性：类型摇滚
hitBuilder.setProperty("type", "rock");
// 设置属性：歌曲标题
hitBuilder.setProperty("title", "wonderful tonight");
// 发送自定义事件打点
MANService manService = MANServiceProvider.getService();
manService.getMANAnalytics().getDefaultTracker().send(hitBuilder.build());
}
return true;
}
return super.onJsAlert(view, url, message, result);
}
});
```

9. 混淆配置

```
-keep class com.alibaba.sdk.android.**{*;}
-keep class com.ut.**{*;}
-keep class com.ta.**{*;}
```