

移动开发平台

崩溃监控服务 (b ugly)

产品文档



腾讯云

【版权声明】

©2013-2019 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或暗示的承诺或保证。

文档目录

崩溃监控服务 (bugly)

产品介绍

控制台使用手册

异常上报

Android 文档

Android 快速入门

高级功能

上传符号表

定制服务

测试是否成功接入了 Crash 上报服务

自定义上报数据

iOS 文档

iOS 快速入门

上传符号表

定制服务

测试是否成功接入 Crash 上报服务

自定义上报数据

常见问题

崩溃监控服务 (bugly)

产品介绍

最近更新时间：2018-04-03 18:01:05

简介

崩溃监控服务 (bugly) 为移动开发者提供专业的异常上报和运营统计，帮助开发者快速发现异常，并通过丰富的现场信息帮您快速定位和解决问题。

。

功能

全面监控

24 小时实时监控和告警，让您及时发现异常问题，支持 Android/IOS 两大智能手机平台，ANR、Crash 和卡顿一网打尽。

智能分析

成熟的大数据分析能力，及时掌握当天 Top 异常，了解引入异常的应用版本，排查机型/系统适配原因以及快速定位内存问题，帮您智能定位和排除异常原因，快速解决问题。

丰富现场

全面的出错堆栈信息，详细的应用运行时数据，自定义的日志和参数，辅助您快速定位异常问题，提升解决效率。

精准搜索

业界领先的多维度搜索服务，通过指定版本、异常类型、渠道等，让您快速找到关键异常，及时定位问题。

客户案例

QQ、微信、QQ 音乐、58 同城、携程、迅雷、美丽说

控制台使用手册

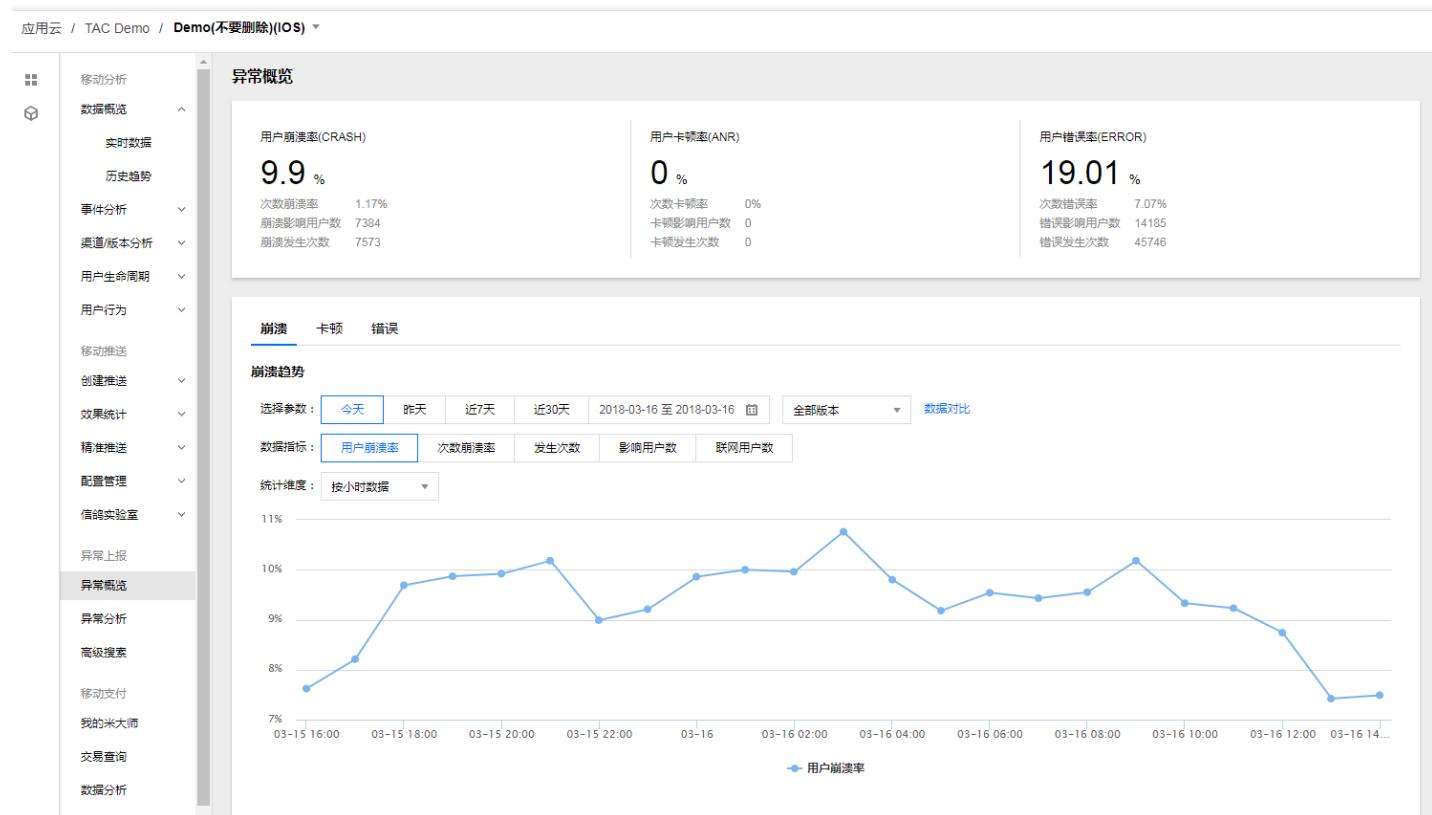
异常上报

最近更新时间：2018-04-04 10:52:05

应用集成 SDK 后，即可在 Web 站点查看应用上报的崩溃数据和联网数据。

概览数据

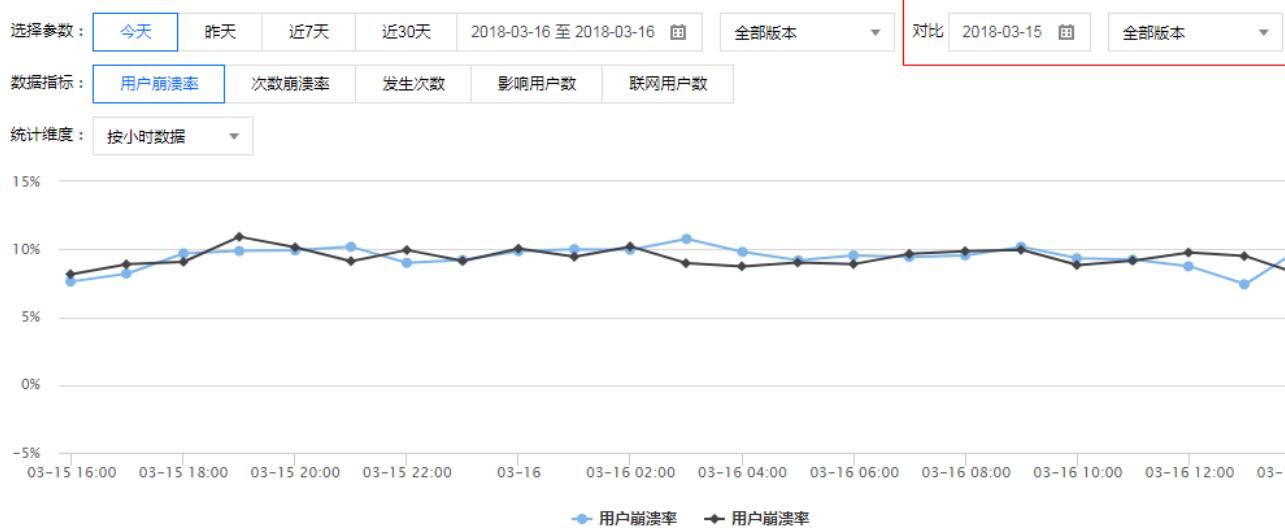
概览数据显示今日实时统计、异常趋势曲线、昨日 TOP 20 异常问题。



其中异常趋势曲线可以选择两个时间段做曲线对比。

崩溃 卡顿 错误

崩溃趋势



异常分析

异常分析可以查看上报问题的列表。页面中可以通过搜索关键字、选择异常类型、选择应用版本来过滤列表。点击异常名称可以进入异常详情页面。

移动分析	异常分析			
	崩溃	卡顿	错误	
数据概览	全部版本			ID/崩溃名称/堆栈
实时数据				🔍
历史趋势				
事件分析				
渠道/版本分析	#1147 SIGABRT	2018-03-16 23:05:56	104978	104847
用户生命周期	BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewContro...			
用户行为	#1178 SIGABRT	2018-03-16 23:05:53	626778	622082
移动推送	libsystem_kernel.dylib __pthread_kill			
创建推送	#1145 SIGILL	2018-03-16 23:05:42	104570	104443
效果统计	BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewContro...			
精准推送	#1200 SIGSEGV	2018-03-16 23:05:03	104783	104657
配置管理	BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewContro...			
信鸽实验室	#1176 SIGSEGV	2018-03-16 23:04:58	105175	105028
异常上报	BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewContro...			
异常概览	#1182 SIGSEGV	2018-03-16 23:04:57	103985	103852
异常分析	BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewContro...			
高级搜索	#1204 SIGTRAP	2018-03-16 23:04:53	104322	104186
移动支付	BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewContro...			
我的米大师	#1184 SIGSEGV	2018-03-16 23:04:02	103748	103621
交易查询				
数据分析				

异常详情

异常详情页展示了当前异常总上报次数和总影响用户数，以及显示该异常的上报列表。

移动分析

数据概览

事件分析

渠道/版本分析

用户生命周期

用户行为

移动推送

创建推送

效果统计

精准推送

配置管理

信鸽实验室

异常上报

异常概览

异常分析

高级搜索

移动支付

我的米大师

交易查询

数据分析

异常详情

#1147 SIGABRT

异常描述 : BuglyDemo -[CRLDetailViewController doCrash] (CRLDetailViewController.m:)

发生次数 104,979

影响用户数 104,848

上报记录列表

上报ID	版本	异常上报时间	设备	系统版本	操作
24746142	1.0(1)	2018-03-16 22:45:02 1...	iphone 6	8.1.3(12B466)	详情
24741173	1.0(2)	2018-03-16 22:12:35 3...	ipad 3	7.1.1(11D201)	详情
24748124	2.0(0)	2018-03-16 22:54:12 1...	iphone 6	8.1(12B411)	详情
24734305	1.0(1)	2018-03-16 22:53:10 0...	iphone 6	8.0.2(12A405)	详情
24724764	1.0(2)	2018-03-16 22:52:18 0...	iphone 6	7.0.4(11B554a)	详情
24747152	2.0(0)	2018-03-16 22:54:05 5...	iphone 6	6.1.6(10B500)	详情
24716011	1.0(0)	2018-03-16 22:14:26 6...	iphone 6	8.1.3(12B466)	详情
24741208	1.0(0)	2018-03-16 22:43:05 0...	ipad air	8.1.3(12B466)	详情
24735306	2.0(0)	2018-03-16 22:34:17 4...	iphone 6	8.1(12B411)	详情
24735346	2.0(0)	2018-03-16 22:59:34 7...	iphone 6	8.1(12B411)	详情

上报详情

上报详情页展示单个上报的详细信息，可以通过上传符号表解析上报数据。

上报数据里包含出错堆栈、跟踪数据、跟踪日志以及一些其他日志信息。

出错堆栈可以通过切换“解析”和“原始”来显示代码。

移动分析

数据概览

事件分析

渠道/版本分析

用户生命周期

用户行为

移动推送

创建推送

效果统计

精准推送

配置管理

信鸽实验室

异常上报

异常概览

异常分析

高级搜索

移动支付

我的米大师

交易查询

数据分析

基础信息

异常进程#线程: BuglyDemo#Thread 0

应用版本: 1.0(1)

用户ID: Unknown

设备机型: iPhone 6

发生时间: 2018-03-16 22:45:02

应用包名: com.tencent.buglydemo

上报时间: 2018-03-16 22:45:02

系统版本: 8.1.3(12B466)

出错堆栈 跟踪数据 跟踪日志 符号表

#Thread 0 SIGABRT

1 libsystem_kernel.dylib __pthread_kill + 8
2 libsystem_pthread.dylib pthread_kill + 58
3 libsystem_c.dylib abort + 72
4 CrashLibiOS-[CRLCrashAbort crash] (CRLCrashAbort.m:37)
5 BuglyDemo-[CRLDetailViewController doCrash] (CRLDetailViewController.m:53)
6 UIKit-[UIApplication sendAction:to:from:forEvent:] + 66
7 UIKit-[UIControl sendAction:to:forEvent:] + 40
8 UIKit-[UIControl _sendActionsForEvents:withEvent:] + 580
9 UIKit-[UIControl touchesEnded:withEvent:] + 580
10 UIKit-[UIGestureRecognizerUpdate + 10154]
11 UIKit-[UIWindow _sendGesturesForEvent:] + 780
12 UIKit-[UIWindow sendEvent:] + 516
13 UIKit-[UIApplication sendEvent:] + 192
14 UIKit-[UIApplicationHandleEventFromQueueEvent + 14534]
15 UIKit-[UIApplicationHandleEventQueue + 1346]
16 CoreFoundation-[__CFRunLoop_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION + 12]
17 CoreFoundation-[__CFRunLoopDoSources0 + 218]
18 CoreFoundation-[__CFRunLoopRun + 764]

高级搜索

通过各种条件快速查找需要定位分析的异常。

Android 文档

Android 快速入门

最近更新时间：2018-08-16 16:28:43

准备工作

您首先需要一个 Android 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。

第一步：创建项目和应用（已完成请跳过）

在使用我们的服务前，您必须先在 MobileLine 控制台上 [创建项目和应用](#)。

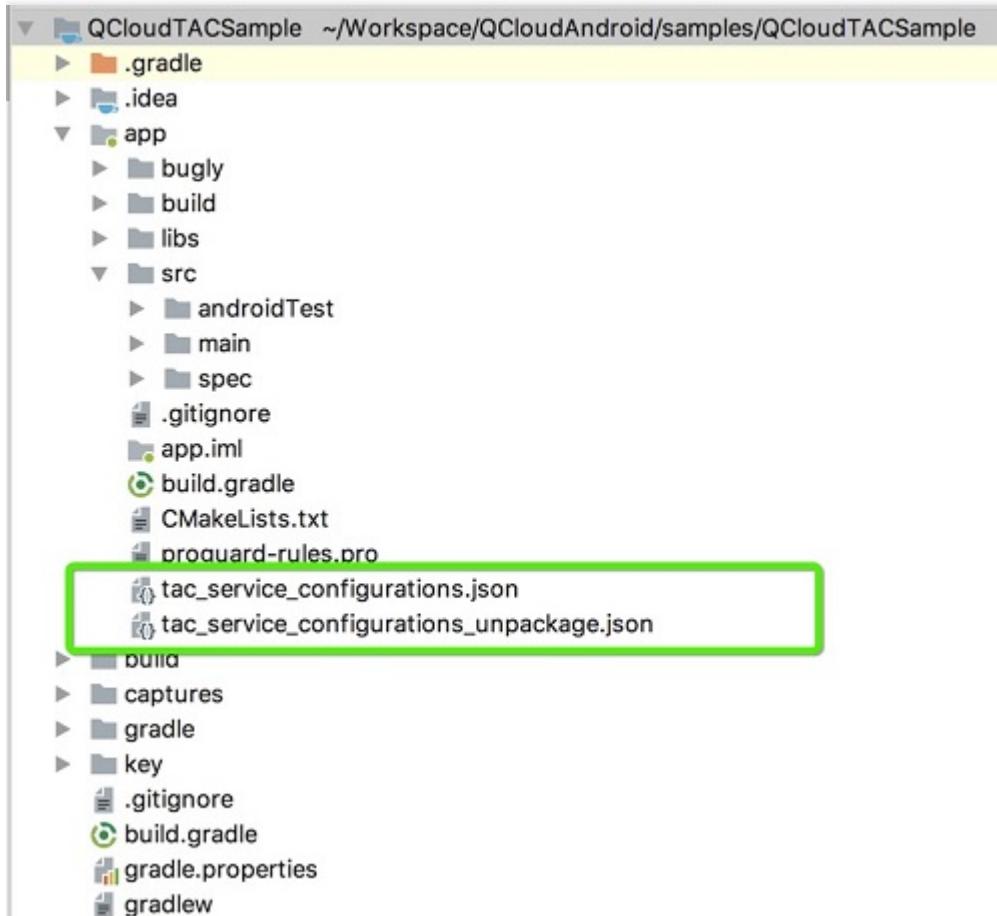
第二步：添加配置文件（已完成请跳过）

在您创建好的应用上单击【下载配置】按钮来下载该应用的配置文件的压缩包：



The screenshot shows the 'Application Management' section of the MobileLine control panel. On the left, there is a sidebar with '应用管理' (Application Management) selected. The main area displays an application named 'AndroidDemo (Android)' with the ID 'AppId : 102719' and package name '应用包名 : com.android.demo'. To the right of the application name are four metrics: '日活跃用户' (0), '月活跃用户' (0), and '遇到崩溃的用户比率' (0%). A red arrow points to the '快速入门 下载配置' (Quick Start Download Configuration) button, which is highlighted in blue. At the top right of the interface, there are links for '应用设置' (Application Settings) and '查看文档' (View Documentation).

解压该压缩包，您会得到 `tac_service_configurations.json` 和 `tac_service_configurations_unpackage.json` 两个文件，请您如图所示添加到您自己的工程中去。



注意：

请您按照图示来添加配置文件，`tac_service_configurations_unpackage.json` 文件中包含了敏感信息，请不要打包到 APK 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

您需要在工程级 `build.gradle` 文件中添加 SDK 插件的依赖：

```
buildscript {  
    ...  
    dependencies {  
        classpath 'com.android.tools.build:gradle:3.0.1'  
        // 添加这行  
        classpath 'com.tencent.tac:tac-services-plugin:1.3.+'  
    }  
}
```

在应用级 build.gradle 文件 (通常是 app/build.gradle) 中添加 crash 服务依赖 , 并使用插件 :

```
dependencies {  
    // 增加这两行  
    compile 'com.tencent.tac:tac-core:1.3.+'  
    compile 'com.tencent.tac:tac-crash:1.3.+'  
}  
  
...  
  
// 在文件最后使用插件  
apply plugin: 'com.tencent.tac.services'
```

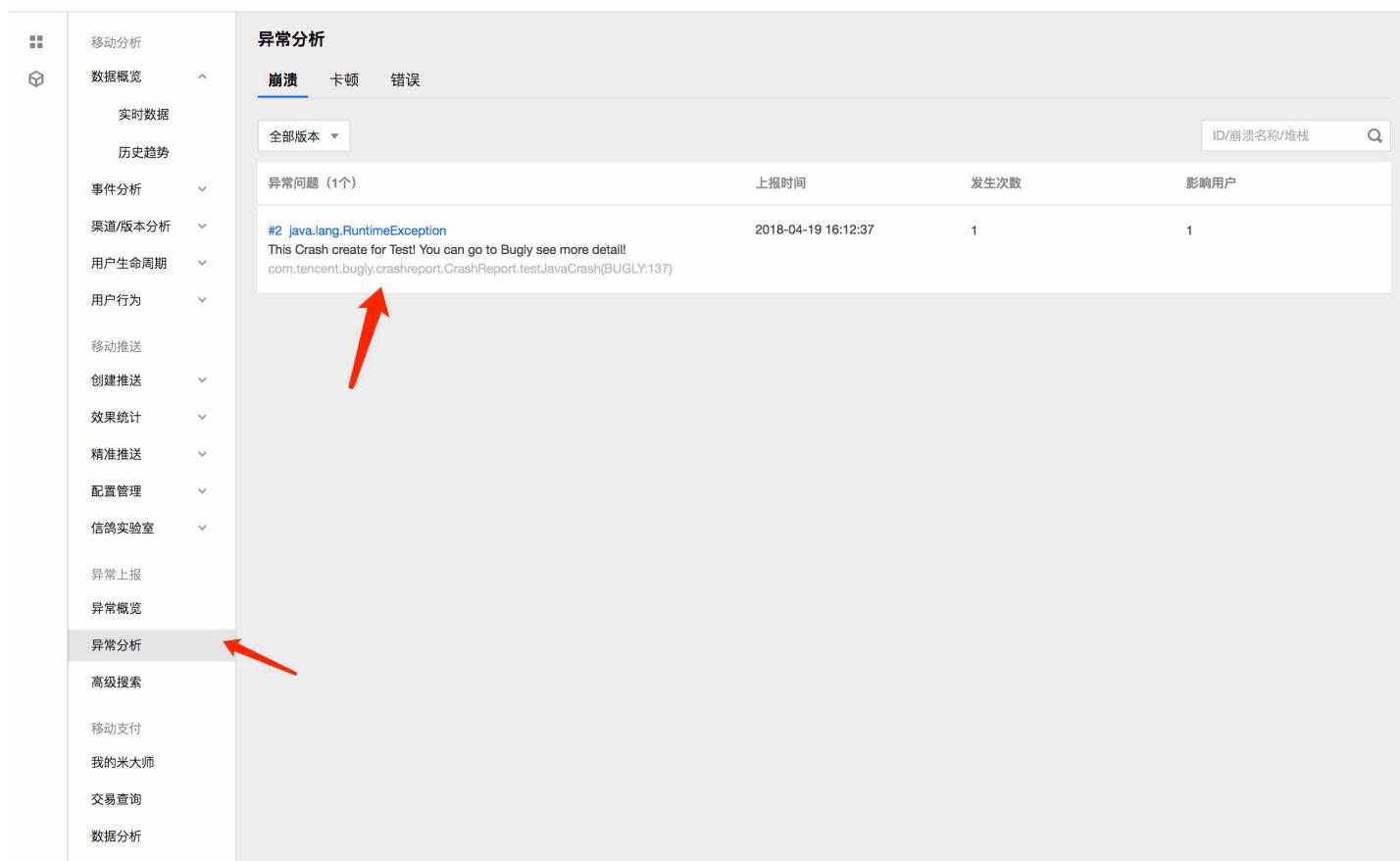
至此您已成功接入了 MobileLine 崩溃监控服务。

验证服务

为了让您测试是否正确的接入了 Crash 服务 , 您可以调用如下代码来主动产生 Crash :

```
TACCrashSimulator.testJavaCrash();
```

应用 Crash 后 , 您可以登录 [MobileLine 控制台](#) , 然后单击【异常上报】下的【异常分析】 , 即可查看上报到控制台的异常 , 如果没有上报 , 您可以查看 [常见问题](#)



移动分析

数据概览

实时数据

历史趋势

事件分析

渠道/版本分析

用户生命周期

用户行为

移动推送

创建推送

效果统计

精准推送

配置管理

信鸽实验室

异常上报

异常概览

异常分析 (highlighted)

高级搜索

移动支付

我的米大师

交易查询

数据分析

崩溃 (highlighted)

卡顿

错误

全部版本

ID/崩溃名称/堆栈

异常问题 (1个)	上报时间	发生次数	影响用户
#2 java.lang.RuntimeException This Crash create for Test! You can go to Bugly see more detail! com.tencent.bugly.crashreport.CrashReport.testJavaCrash(BUGLY:137)	2018-04-19 16:12:37	1	1

Proguard 配置

如果您的代码开启了混淆，为了 SDK 可以正常工作，请在 `proguard-rules.pro` 文件中添加如下配置：

```
# MobileLine Core

-keep class com.tencent.qcloud.core.** { *; }
-keep class bolts.** { *; }
-keep class com.tencent.tac.** { *; }
-keep class com.tencent.stat.*{*}
-keep class com.tencent.mid.*{*}
-dontwarn okhttp3.**
-dontwarn okio.**
-dontwarn javax.annotation.**
-dontwarn org.conscrypt.**
```

```
# MobileLine Crash
```

```
-keep class com.tencent.bugly.** { *;}
```

高级功能

最近更新时间：2019-04-17 16:32:39

Javascript 的异常捕获功能

Crash 提供了 Javascript 的异常捕获和上报能力，以便开发者可以感知到 WebView 中发生的 Javascript 异常。

```
TACCrashService.getInstance().setJavascriptMonitor(Webview);
```

由于 Android 4.4 以下版本存在反射漏洞，接口默认只对 Android 4.4 及以上版本有效。

如果您是采用自动集成 SDK 的方式，可以选择自动注入或者手动注入两种方式。

自动注入

建议在 WebChromeClient 的 onProgressChanged 函数中调用接口，例子如下：

```
WebView webView = new WebView(this);
// 设置WebChromeClient
webView.setWebChromeClient(new WebChromeClient() {
    @Override
    public void onProgressChanged(WebView webView, int progress) {
        // 增加 Javascript 异常监控
        TACCrashService.getInstance().setJavascriptMonitor(webView, true);
        super.onProgressChanged(webView, progress);
    }
});
// 加载 HTML
webView.loadUrl(url);
```

手动注入

将下载的 SDK 资源包中的 Buggy.js 文件添加到需要监控 Javascript 异常的 HTML 中：

```
<html>
<script src="bugly.js"></script>
<body>
...
</body>
</html>
```

在 WebView 加载完该 HTML 后设置 Javascript 的异常捕获功能：

```
WebView webView = new WebView(this);
```

```
// 加载 HTML
webView.loadUrl(url);
// 增加 Javascript 异常监控
TACCrashService.getInstance().setJavascriptMonitor(webView, false);
```

在捕获到 Javascript 异常后，默认会上报以下信息：

- Android 设备的相关信息；
- Javascript 异常堆栈和其他信息；
- Java 堆栈；
- WebView 的信息，目前只包括 ContentDescription。

MultiDex 注意事项

如果使用了 MultiDex，建议通过 Gradle 的 "multiDexKeepFile" 配置等方式把 Crash 的类放到主 Dex，另外建议在 Application 类的 "attachBaseContext" 方法中主动加载非主 dex：

```
public class MyApplication extends SomeOtherApplication {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(context);
        Multidex.install(this);
    }
}
```

增加上报进程控制

如果 App 使用了多进程且各个进程都会初始化 Crash（例如在 Application 类 onCreate() 中初始化 Crash），那么每个进程下都会进行数据上报，造成不必要的资源浪费。

您可以通过以下方法控制只在主线程上报数据，首先通过以下方法获取当前进程是否为主进程：

```
public boolean isMainProcess() {
    ActivityManager am = ((ActivityManager) getSystemService(Context.ACTIVITY_SERVICE));
    List<ActivityManager.RunningAppProcessInfo> processInfos = am.getRunningAppProcesses();
    String mainProcessName = getPackageName();
    int myPid = android.os.Process.myPid();
    for (ActivityManager.RunningAppProcessInfo info : processInfos) {
        if (info.pid == myPid && mainProcessName.equals(info.processName)) {
            return true;
        }
    }
}
```

```
    }  
    return false;  
}
```

然后，在您启动服务的地方，加上进程的判断：

```
if (isMainProcess()) {  
    crashService.start(context);  
}
```

上传符号表

最近更新时间：2018-07-10 13:13:43

如果您的项目使用了 Native 工程或者对代码进行了混淆，那么我们需要您上传符号表对 App 发生 Crash 的程序堆栈进行解析和还原；如果您的项目既没有使用 Native 工程也没有对代码进行混淆，那么您无需上传符号表。

自动上传：使用 Android Studio 插件

如果您已经按照入门文档在 build.gradle 中使用了 SDK 插件，那么符号表会自动上传，不需要其他配置：

```
// 在文件最后使用插件
apply plugin: 'com.tencent.tac.services'
```

手动上传

1. 下载 [符号表工具](#)。
2. 根据 UUID 定位 Debug SO 文件，具体可参考工具包中的使用文档。
3. 使用工具生成符号表文件（zip 文件），具体的使用方法可参考工具包中的使用文档。
4. 在移动开发平台（MobileLine）的控制台上传符号表文件。

如果您的项目只使用了混淆代码（Proguard），而没有 Native 工程，只需要直接上传 Proguard 生成的 Mapping 文件即可。

定制服务

最近更新时间：2019-04-26 11:01:42

我们提供了一些高级配置项，您可以通过这些配置项定制化您的 Crash 服务。

注意：

您需要在启动服务前完成配置。

获取 Options

```
final TACApplicationOptions tacApplicationOptions = TACApplication.options();
final TACCrashOptions tacCrashOptions = tacApplicationOptions.sub("crash");
```

设置上报延时时间

Crash 会在启动 10s 后联网同步数据。如果您有特别需求，可以修改这个时间。

```
final TACCrashOptions tacCrashOptions = tacApplicationOptions.sub("crash");
tacCrashOptions.setReportDelay(20000); //改为 20s
```

设置 Crash 回调

您可以设置 Crash 发生时的回调，回调返回的数据将伴随 Crash 一起上报到 Crash 平台，并展示在附件中：

Crash 回调类的定义如下：

```
public interface TACCrashHandleCallback {

    public static final int CRASHTYPE_JAVA_CRASH = 0; // Java crash
    public static final int CRASHTYPE_JAVA_CATCH = 1; // Java caught exception
    public static final int CRASHTYPE_NATIVE = 2; // Native crash
    public static final int CRASHTYPE_U3D = 3; // Unity error
    public static final int CRASHTYPE_ANR = 4; // ANR
    public static final int CRASHTYPE_COCOS2DX_JS = 5; // Cocos JS error
}
```

```
public static final int CRASHTYPE_COCOS2DX_LUA = 6; // Cocos Lua error

    /**
     * 处理 Crash，并上传自定义 key-values
     *
     * @param crashType Crash 类型
     * @param errorCode 错误码
     * @param errorMessage 错误信息
     * @param errorStack 错误堆栈
     * @return 需要上传给控制台的信息
     */
    Map<String, String> onCrashUploadKeyValues(int crashType, String errorCode, String errorMessage,
                                                String errorStack);

    /**
     * 处理 Crash，并上传自定义二进制文件
     *
     * @param crashType Crash 类型
     * @param errorCode 错误码
     * @param errorMessage 错误信息
     * @param errorStack 错误堆栈
     * @return 需要上传给控制台的信息
     */
    byte[] onCrashUploadBinary(int crashType, String errorCode, String errorMessage, String errorStack);

}
```

设置方法如下：

```
final TACCrashOptions tacCrashOptions = tacApplicationOptions.sub("crash");
tacCrashOptions.setHandleCallback(new TACCrashHandleCallback() {
    @Override
    public Map<String, String> onCrashUploadKeyValues(int crashType, String errorCode, String errorMessage,
                                                       String errorStack) {
        LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
        map.put("Key", "Value");
        return null;
    }

    @Override
    public byte[] onCrashUploadBinary(int crashType, String errorCode, String errorMessage, String errorStack) {
        try {
            return "Extra data.".getBytes("UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
});
```

```
} catch (Exception e) {  
    return null;  
}  
}  
}  
});
```

关闭 Native Crash 上报

默认会上报 Native Crash，如果您需要，可以关闭该功能。

```
final TACCrashOptions tacCrashOptions = tacApplicationOptions.sub("crash");  
tacCrashOptions.enableNativeCrashMonitor(false);
```

关闭 ANR 上报

默认会上报 ANR，如果您需要，可以关闭该功能。

```
final TACCrashOptions tacCrashOptions = tacApplicationOptions.sub("crash");  
tacCrashOptions.enableANRCrashMonitor(false);
```

测试是否成功接入了 Crash 上报服务

最近更新时间：2018-04-24 18:11:01

您可以通过以下方法主动触发异常，以测试 SDK 是否正常工作，不必花费时间等待应用出现崩溃。

调用测试代码产生 Crash

您可以在任意位置调用如下代码产生 Java Crash 异常。

```
// 模拟java异常
TACCrashSimulator.testJavaCrash();
```

在控制台上查看异常是否上报成功

应用 Crash 后，您可以登录 [MobileLine 控制台](#)，然后点击【异常上报】下的【异常分析】，即可查看上报到控制台的异常，如果没有上报，可以查看 [常见问题](#)

移动分析

数据概览

实时数据

历史趋势

事件分析

渠道/版本分析

用户生命周期

用户行为

移动推送

创建推送

效果统计

精准推送

配置管理

信鸽实验室

异常上报

异常概览

异常分析

高级搜索

移动支付

我的米大师

交易查询

数据分析

异常分析

崩溃 卡顿 错误

全部版本 ▾

ID/崩溃名称/堆栈 🔍

异常问题 (1个)	上报时间	发生次数	影响用户
#2 java.lang.RuntimeException This Crash create for Test! You can go to Bugly see more detail! com.tencent.bugly.crashreport.CrashReport.testJavaCrash(BUGLY:137)	2018-04-19 16:12:37	1	1

自定义上报数据

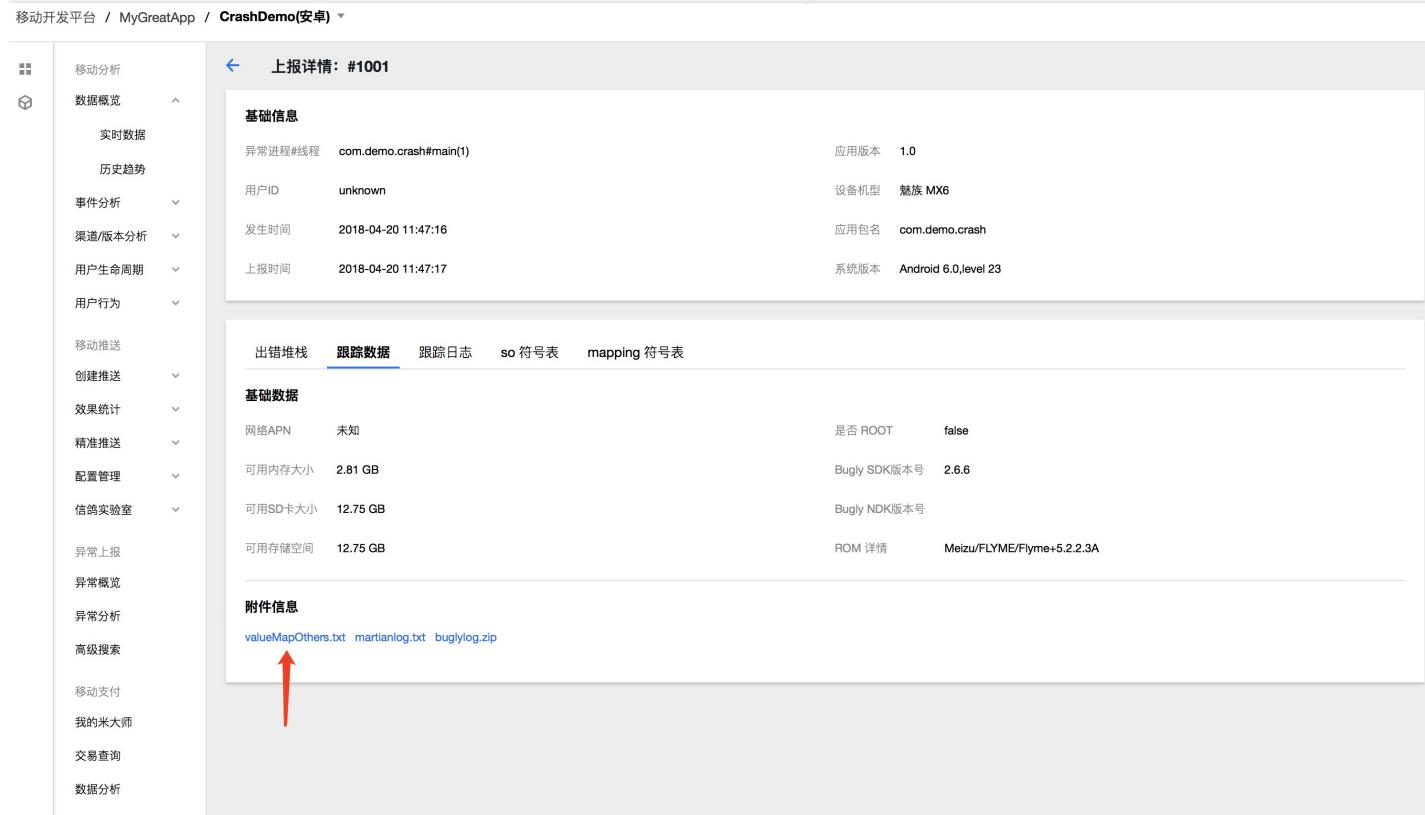
最近更新时间：2018-04-24 18:14:59

自定义 Map 参数

自定义 Map 参数可以保存发生 Crash 时的一些自定义的环境信息。在发生 Crash 时会随着异常信息一起上报并在页面展示。

```
TACCrashService.getInstance().putUserData(context, "userkey", "uservalue");
```

异常上报后，您可以找到对应的 Crash 后，查看设置的 Map 参数。



The screenshot shows the 'Crash Detail' page for a specific crash report. In the 'Attachments' section, there is a file named 'valueMapOthers.txt'. A red arrow points to this file, indicating where custom Map parameters are stored.

基础信息

异常进程#线程	com.demo.crash#main(1)	应用版本	1.0
用户ID	unknown	设备机型	魅族 MX6
发生时间	2018-04-20 11:47:16	应用包名	com.demo.crash
上报时间	2018-04-20 11:47:17	系统版本	Android 6.0.level 23

基础数据

网络APN	未知	是否 ROOT	false
可用内存大小	2.81 GB	Bugly SDK版本号	2.6.6
可用SD卡大小	12.75 GB	Bugly NDK版本号	
可用存储空间	12.75 GB	ROM 详情	Meizu/FLYME/Flyme+5.2.2.3A

附件信息

- valueMapOthers.txt
- martianlog.txt
- buglylog.zip

注意：

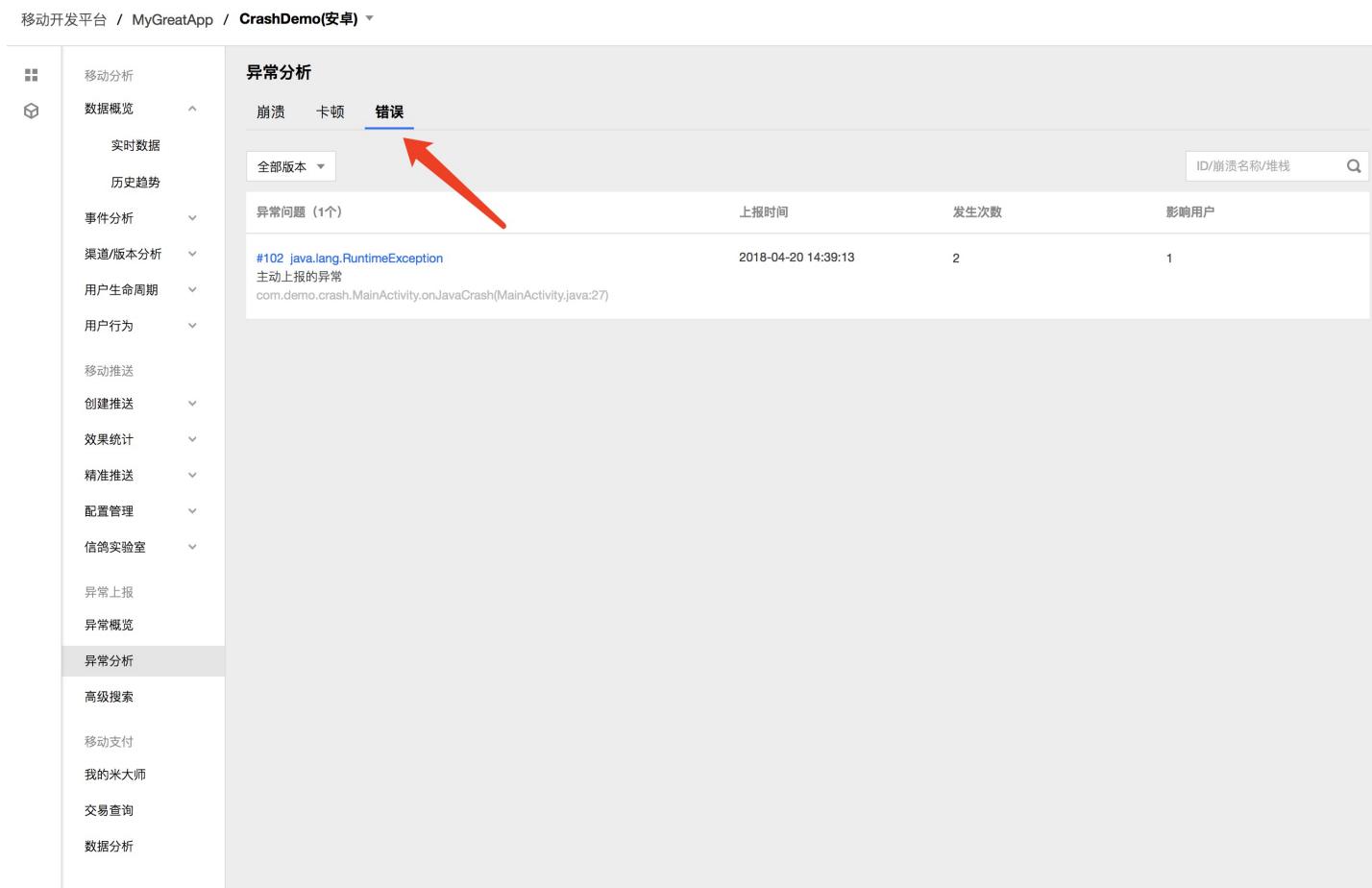
最多可以有 9 对自定义的 key-value (超过则添加失败) ；
key 限长 50 字节，value 限长 200 字节，过长截断；
key 必须匹配正则：[a-zA-Z[0-9]]+。

主动上报异常

主动上报开发者 Catch 的异常您可能会关注某些重要异常的 Catch 情况，我们提供了上报这类异常的接口。

```
try {  
    //...  
} catch (Throwable thr) {  
    TACCrashService.getInstance().postCaughtException(thr);  
}
```

您可以在【异常上报】中点击【异常分析】，并在【错误】栏下查看主动上报的异常：



移动开发平台 / MyGreatApp / CrashDemo(安卓) ▾

异常分析

崩溃 卡顿 错误 全部版本 ID/崩溃名称/堆栈

异常问题 (1个)	上报时间	发生次数	影响用户
#102 java.lang.RuntimeException 主动上报的异常 com.demo.crash.MainActivity.onJavaCrash(MainActivity.java:27)	2018-04-20 14:39:13	2	1

iOS 文档

iOS 快速入门

最近更新时间：2019-03-04 16:05:26

移动开发平台 (MobileLine) 使用起来非常容易，只需要简单的 4 步，您便可快速接入移动崩溃监测。接入后，您即可获得我们提供的各项能力，减少您在开发应用时的重复工作，提升开发效率。

准备工作

为了使用移动开发平台 (MobileLine) iOS 版本的 SDK，您首先需要一个 iOS 工程，这个工程可以是您现有的工程，也可以是您新建的一个空的工程。

第一步：创建项目和应用

在使用我们的服务前，您必须先在 MobileLine 控制台上 [创建项目和应用](#)。

如果您已经在 MobileLine 控制台上创建过了项目和应用，请跳过此步。

第二步：添加配置文件

如果您已经添加过配置文件，请跳过此步。

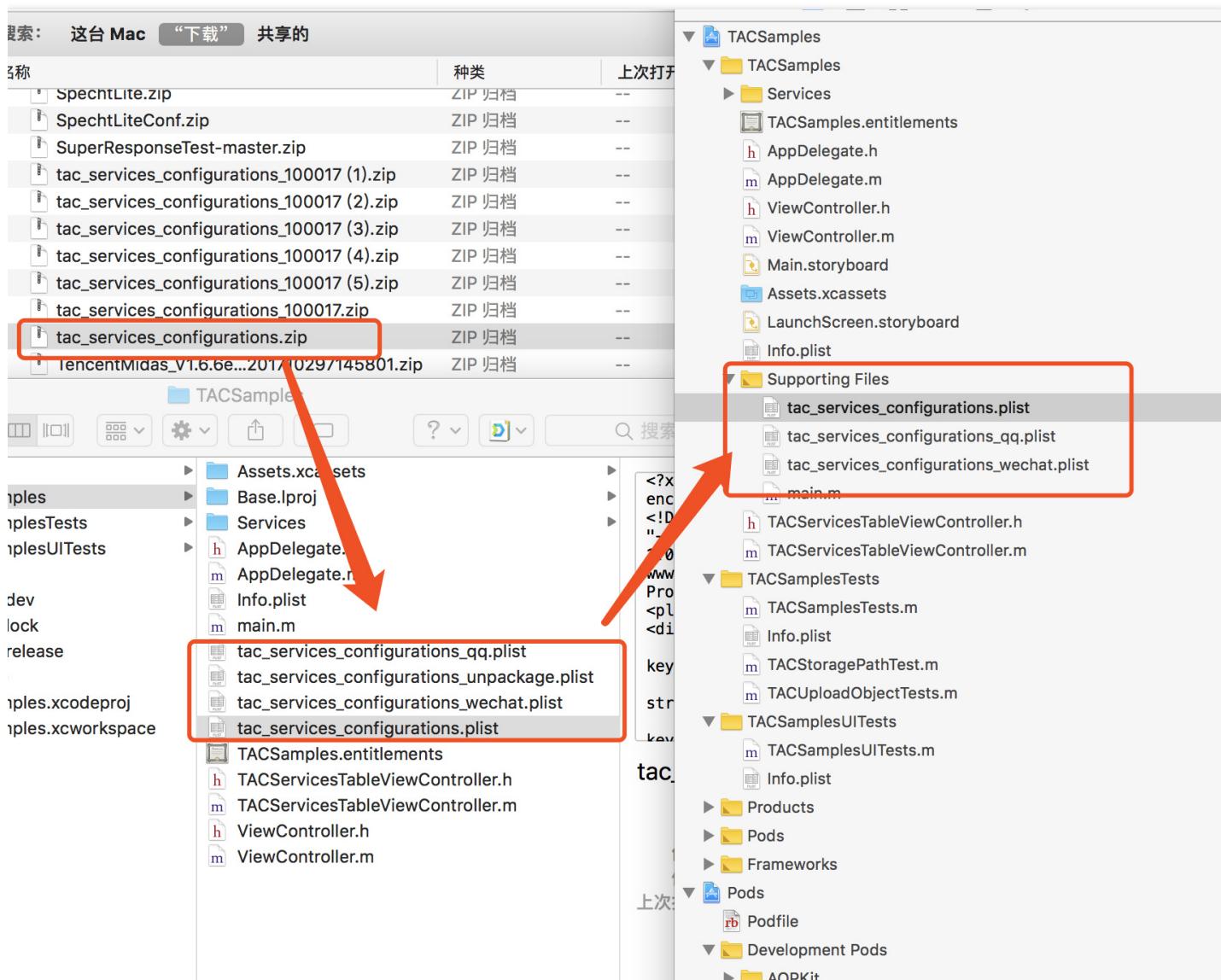
创建好应用后，您可以点击红框中的【下载配置】来下载该应用的配置文件的压缩包：



The screenshot shows the 'Application Management' section of the MobileLine control panel. It lists an application named 'FastNote (iOS)' with AppId: 100188 and package name: com.dzpqbz.chatdaily. The interface includes tabs for 'Create Application', 'Quick Start', 'Integration Guide', and 'Download Configuration'. A red box and arrow point to the 'Download Configuration' button. Below the application list, there are statistics for daily and monthly active users, and a crash rate percentage.

日活跃用户	月活跃用户	遇到崩溃的用户比率
0	2	0%

解压后将 tac_services_configurations.plist 文件集成进项目中。其中有一个 tac_services_configurations_unpackage.plist 文件，请将该文件放到您工程的根目录下面(切记不要将改文件添加进工程中)。添加好配置文件后，继续单击【下一步】。



注意：

请您按照图示来添加配置文件，`tac_service_configurations_unpackage.plist` 文件中包含了敏感信息，请不要打包到 apk 文件中，MobileLine SDK 也会对此进行检查，防止由于您误打包造成的敏感信息泄露。

第三步：集成 SDK

如果还没有 Podfile，请创建一个。

```
$ cd your-project directory  
$ pod init
```

并在您的 Podfile 文件中添加移动开发平台 (MobileLine) 的私有源：

```
source "https://git.cloud.tencent.com/qcloud_u/cocopoads-repo"  
source "https://github.com/CocoaPods/Specs"
```

在 Podfile 中添加依赖：

```
pod 'TACCrash'
```

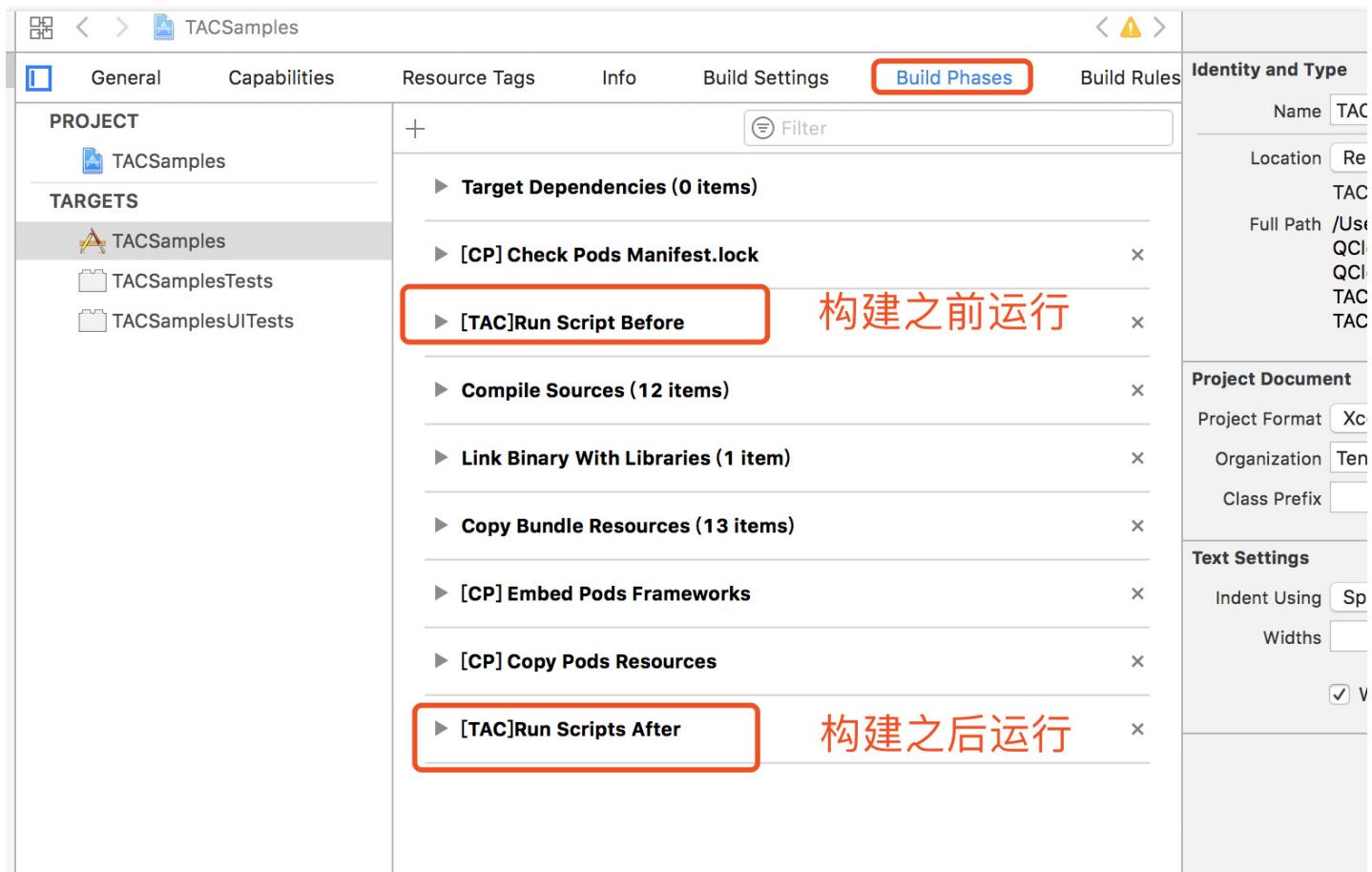
配置程序需要脚本

如果您在其他模块中完成了此步骤，请不要重复执行。

为了简化 SDK 的接入流程，我们使用 shell 脚本，帮助您自动化的去执行一些繁琐的操作，比如 crash 自动上报，在 Info.plist 里面注册各种第三方 SDK 的回调 scheme。因而，需要您添加以下脚本来使用我们自动化的加入流程。

脚本主要包括两个：

1. 在构建之前运行的脚本，该类型的脚本会修改一些程序的配置信息，比如在 Info.plist 里面增加 qqwallet 的 scheme 回调。
2. 在构建之后运行的脚本，该类型的脚本在执行结束后做一些动作，比如 Crash 符号表上报。



自动添加所有程序需要脚本

自动添加脚本目前仅支持通过 Cocoapods 方式进行集成的用户。如果使用 Cocoapods 集成的话，在 Podfile 的最后一行后面新起一行，并且将以下代码粘贴进去以后，运行 `pod install` 即可，就完成了配置程序需要脚本这一步。

```
pre_install do |installer|
  puts "[TAC]-Running post installer"
  xcodeproj_file_name = "placeholder"
  Dir.foreach("./") do |file|
    if file.include?("xcodeproj")
      xcodeproj_file_name = file
    end
  end
  puts "[TAC]-project file is #{xcodeproj_file_name}"
  project = Xcodeproj::Project.open(xcodeproj_file_name)
  project.targets.each do |target|
    shell_script_after_build_phase_name = "[TAC] Run After Script"
    shell_script_before_build_phase_name = "[TAC] Run Before Script"
```

```
puts "[TAC]-target.product_type is #{target.product_type}"
if target.product_type.include?("application")
  should_insert_after_build_phases = 0
  should_insert_before_build_phases = 0
  after_build_phase = nil
  before_build_phase = nil
  target.shell_script_build_phases.each do |bp|
    if !bp.name.nil? and bp.name.include?(shell_script_after_build_phase_name)
      should_insert_after_build_phases = 1
      after_build_phase = bp
    end
    if !bp.name.nil? and bp.name.include?(shell_script_before_build_phase_name)
      should_insert_before_build_phases = 1
      before_build_phase = bp
    end
  end

if should_insert_after_build_phases == 1
  puts "[TAC]-Build phases with the same name--#{shell_script_after_build_phase_name} has already existed"
else
  after_build_phase = target.new_shell_script_build_phase
  puts "[TAC]-installing run after build phases-- #{after_build_phase}"

end
after_build_phase.name = shell_script_after_build_phase_name
after_build_phase.shell_script = "
if [ -f \"${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.after.sh\" ]; then
  bash \"${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.after.sh\"
fi
"
after_build_phase.shell_path = '/bin/sh'
if should_insert_before_build_phases == 1
  puts "[TAC]-Build phases with the same name--#{shell_script_before_build_phase_name} has already existed"
else
  before_build_phase = target.new_shell_script_build_phase
  target.build_phases.insert(0,target.build_phases.pop)
  puts "[TAC]-installing run before build phases-- #{before_build_phase}"

end
before_build_phase.name = shell_script_before_build_phase_name
before_build_phase.shell_script = "
if [ -f \"${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.before.sh\" ]; then
```

```
bash \"${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.before.sh\"  
fi  
"  
before_build_phase.shell_path = '/bin/sh'  
end  
end  
puts "[TAC]-Saving projects"  
project.save()  
end
```

注：运行 pod install 以后，可以按照上面的图片打开项目里的 Build Phases 确认是否有 [TAC] 开头，与图上类似的 Build phases。如果没有的话，可再次运行 pod install 后检查即可。

手动添加程序需要脚本

请按照以下步骤来添加脚本：

添加构建之前运行的脚本

1. 在导航栏中打开您的工程。
2. 打开 Tab Build Phases。
3. 单击 Add a new build phase，并选择 New Run Script Phase，您可以将改脚本命名 TAC Run Before

注意：

请确保该脚本在 Build Phases 中排序为第二。

4. 根据自己集成的模块和集成方式将代码粘贴入 Type a script... 文本框:。

需要黏贴的代码

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以 "tac.run.all.before.sh" 命名的脚本，并执行，如果您不需要自定义不用动这里]  
${TAC_CORE_FRAMEWORK_PATH}/Scripts/tac.run.all.before.sh
```

其中 THIRD_FRAMEWORK_PATH 变量的取值根据您的安装方式而不同：

- 如果您使用 Cocoapods 来集成的则为 \${PODS_ROOT}/TACCore，您需要黏贴的代码实例如下：

```
 ${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.before.sh
```

- 如果您使用手工集成的方式则为 [您存储 TACCore 库的地址]，即您 TACCore framework 的引入路径，您需要黏贴的代码实例如下：

```
export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]  
[您存储 TACCore 库的地址]/TACCore.framework/Scripts/tac.run.all.before.sh
```

添加构建之后运行的脚本

1. 在导航栏中打开您的工程。
2. 打开 Tab Build Phases。
3. 点击 Add a new build phase，并选择 New Run Script Phase，您可以将改脚本命名 TAC Run Before。

注意：

请确保该脚本在 Build Phases 中排序需要放到最后。

4. 根据自己集成的模块和集成方式将代码粘贴入 Type a script... 文本框：

需要黏贴的代码

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]  
${TAC_CORE_FRAMEWORK_PATH}/Scripts/tac.run.all.after.sh
```

其中 THIRD_FRAMEWORK_PATH 变量的取值根据您的安装方式而不同：

- 如果您使用 Cocoapods 来集成的则为 \${PODS_ROOT}/TACCore，您需要黏贴的代码实例如下：

```
 ${SRCROOT}/Pods/TACCore/Scripts/tac.run.all.after.sh
```

- 如果您使用手工集成的方式则为 [您存储 TACCore 库的地址]，即您 TACCore framework 的引入路径，您需要黏贴的代码实例如下：

```
#export TAC_SCRIPTS_BASE_PATH=[自定义执行脚本查找路径，我们会在该路径下寻找所有以“tac.run.all.after.sh”命名的脚本，并执行，如果您不需要自定义不用动这里]  
[您存储 TACCore 库的地址]/TACCore.framework/Scripts/tac.run.all.after.sh
```

第四步：启动服务

移动崩溃监测 服务无需启动，到此您已经成功接入了 MobileLine 移动崩溃监测服务。

后续步骤

了解 MobileLine：

- 查看 [MobileLine 应用示例](#)

向您的应用添加 MobileLine 功能：

- 借助 [Analytics](#) 深入分析用户行为。
- 借助 [messaging](#) 向用户发送通知。
- 借助 [crash](#) 确定应用崩溃的时间和原因。
- 借助 [storage](#) 存储和访问用户生成的内容（如照片或视频）。
- 借助 [authorization](#) 来进行用户身份验证。
- 借助 [payment](#) 获取微信和手 Q 支付能力

上传符号表

最近更新时间：2018-04-24 18:07:16

解析崩溃堆栈需要符号表，您可通过下面两种方式进行符号表的上传。

通过配置运行后脚本自动上传

如果您已经完成了快速入门步骤中的【配置程序需要脚本】操作，那么每次在程序编译完成后会将符号表自动上传到后台服务，无需额外步骤。

手动上传

1. 下载 [符号表工具](#)。
2. 根据 UUID 定位 DSYM 文件，具体可参考工具包中的使用文档。
3. 使用工具生成符号表文件（zip 文件），具体的使用方法可参考工具包中的使用文档。
4. 在移动开发平台（MobileLine）的控制台上传符号表文件。

定制服务

最近更新时间：2019-04-17 15:55:08

我们提供了一些高级配置项，您可以通过这些配置项定制您的 Crash 服务。

注意：

您需要在启动服务前完成配置。

获取 Options

Objective-C 代码示例：

```
TACApplicationOptions* options = [TACApplicationOptions defaultApplicationOptions];
TACCrashOptions* crashOptions = options.crashOptions;
```

Swift 代码示例：

```
let options = TACApplicationOptions.default()
let crashOptions = options?.crashOptions
```

过滤崩溃数据

如果不希望崩溃数据中包含某个第三方库的数据（例如搜狗输入法，SogouInputiPhone.dylib）的话，您可通过设置过滤关键字的形式将其去掉。

Objective-C 代码示例：

```
TACApplicationOptions* options = [TACApplicationOptions defaultApplicationOptions];
TACCrashOptions* crashOptions = options.crashOptions;
NSArray* excludeModuleFilters = @[@"SogouInputiPhone.dylib"];
crashOptions.excludeModuleFilters = excludeModuleFilters;
```

Swift 代码示例：

```
let options = TACApplicationOptions.default()
let crashOptions = options?.crashOptions
```

```
let excludeModuleFilters = ["SogouInputPhone.dylib"]
crashOptions?.excludeModuleFilter = excludeModuleFilters
```

设置 Crash 委托回调

您可以通过设置 delegate 来提供更多的信息以辅助定位分析问题：

Objective-C 代码示例：

```
[TACCrashService shareService].delegate = <#the instance of TACCrashServiceDelegate#>
```

Swift 代码示例：

```
TACCrashService.share().delegate = <#the instance of TACCrashServiceDelegate#>
```

其中协议 TACCrashServiceDelegate 实现拥有以下接口：

```
@optional
**
当抛出异常的时候，需要附带上的环境信息。您可以通过该接口提供更多的信息以辅助您定位问题。

@param service 当前的Crash服务实例
@param exception 异常信息
**
- (NSString*) crashService:(TACCrashService*)service attachmentForException:(NSEException*)exception;
```

关闭 Crash 上报

如果您引入了多个崩溃检测服务，可能不希望启动该模块，请在程序中设置为 NO (默认为 YES)。

Objective-C 代码示例：

```
TACApplicationOptions* options = [TACApplicationOptions defaultApplicationOptions];
TACCrashOptions* crashOptions = options.crashOptions;
crashOptions.enable = NO;
```

Swift 代码示例：

```
let options = TACApplicationOptions.default()
let crashOptions = options?.crashOptions
crashOptions?.enable = false
```

开启或者关闭卡顿监控

卡顿监控可以监控应用内的卡顿情况，并进行上报。如果希望开启或者关闭的话，可以直接设置对应的属性实现：

Objective-C 代码示例：

```
TACApplicationOptions* options = [TACApplicationOptions defaultApplicationOptions];
TACCrashOptions* crashOptions = options.crashOptions;
//设置为 YES 开启卡顿监控
crashOptions.blockMonitorEnable = YES;
```

Swift 代码示例：

```
let options = TACApplicationOptions.default()
let crashOptions = options?.crashOptions
//设置为 YES 开启卡顿监控
crashOptions?.blockMonitorEnable = true
```

更多设置

更多设置信息可以直接参考 TACCrashOptions.h 头文件中的注释。

测试是否成功接入 Crash 上报服务

最近更新时间：2018-04-24 17:40:14

您可以通过以下方法主动触发异常，以测试 SDK 是否正常工作，不必花费时间等待应用出现崩溃。

调用测试代码产生 Crash

您可以在任意位置调用如下代码产生异常。

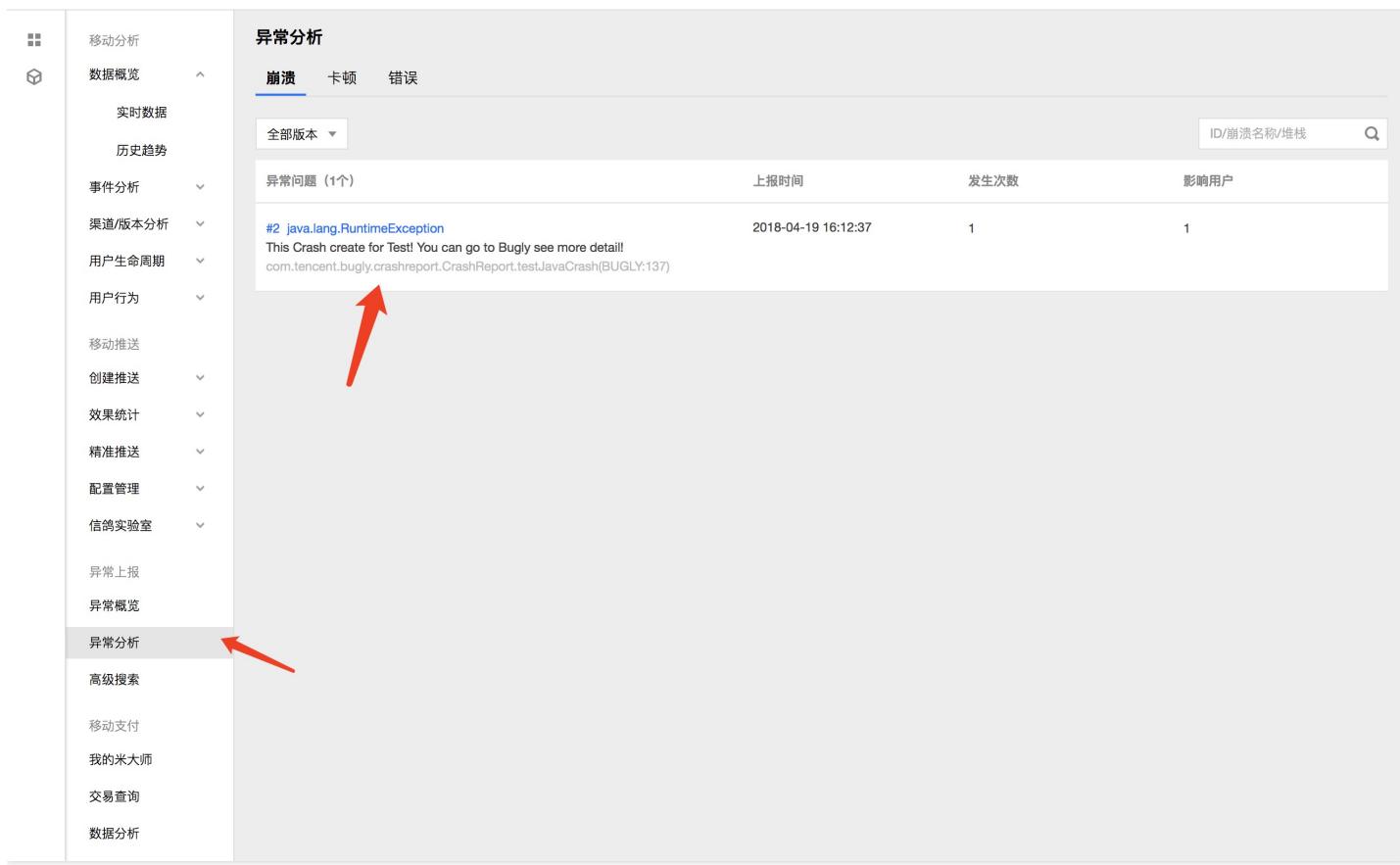
注意：

需要在 Crash 服务启动，也就是进行了配置以后模拟。

```
// 模拟越界访问异常
NSArray* testArray = [NSArray array];
testArray[100];
```

在控制台上查看异常是否上报成功

应用 Crash 后，您可以登录 [MobileLine 控制台](#)，单击【异常上报】>【异常分析】，即可查看上报到控制台的异常，如果没有上报，可以查看 [常见问题](#)



移动分析

数据概览

实时数据

历史趋势

事件分析

渠道/版本分析

用户生命周期

用户行为

移动推送

创建推送

效果统计

精准推送

配置管理

信鸽实验室

异常上报

异常概览

异常分析

高级搜索

移动支付

我的米大师

交易查询

数据分析

异常分析

崩溃 卡顿 错误

全部版本 ▾

ID/崩溃名称/堆栈 Q

异常问题 (1个)	上报时间	发生次数	影响用户
#2 java.lang.RuntimeException This Crash create for Test! You can go to Bugly see more detail! com.tencent.bugly.crashreport.CrashReport.testJavaCrash(BUGLY:137)	2018-04-19 16:12:37	1	1

自定义上报数据

最近更新时间：2018-04-24 18:07:51

自定义参数

自定义参数可以保存发生 Crash 时的一些自定义的环境信息。在发生 Crash 时会随着异常信息一起上报并在页面展示。

```
[[TACCrashService shareService] setUserValue:@"内容" key:@"key"]
```

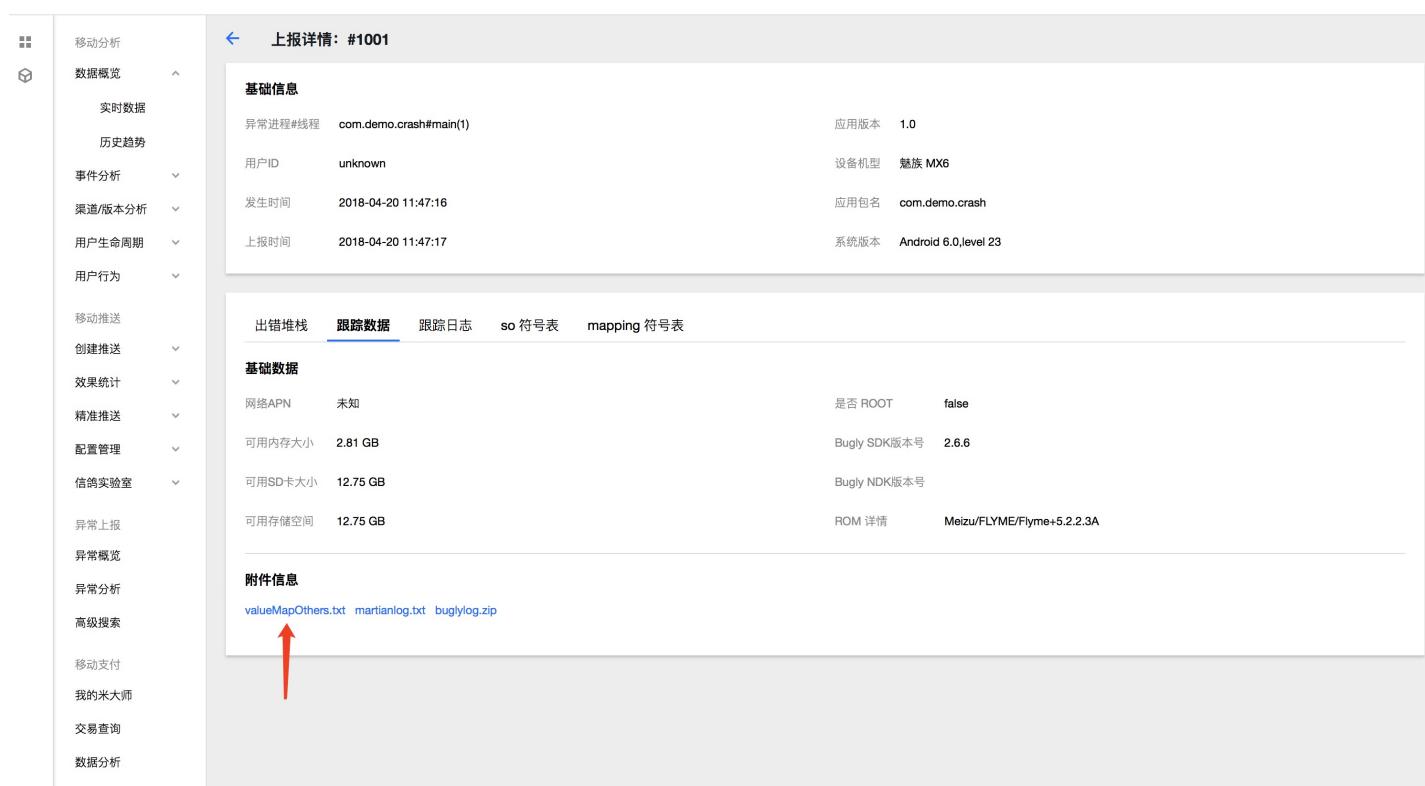
异常上报后，您可以找到对应的 Crash 后，查看设置的 key-value 参数。

最多可以有 9 对自定义的 key-value (超过则添加失败) ；

key 限长 50 字节，value 限长 200 字节，过长截断；

key 必须匹配正则：[a-zA-Z[0-9]]+。

移动开发平台 / MyGreatApp / CrashDemo(安卓) ▾



基础信息	值
异常进程#线程	com.demo.crash#main(1)
应用版本	1.0
设备机型	魅族 MX6
用户ID	unknown
发生时间	2018-04-20 11:47:16
应用包名	com.demo.crash
上报时间	2018-04-20 11:47:17
系统版本	Android 6.0,level 23

基础数据	值
是否 ROOT	false
网络APN	未知
Bugly SDK版本号	2.6.6
可用内存大小	2.81 GB
可用SD卡大小	12.75 GB
Bugly NDK版本号	
可用存储空间	12.75 GB
ROM 详情	Meizu/FLYME/Flyme+5.2.2.3A

附件信息	值
valueMapOthers.txt	martianlog.txt bugglylog.zip

最多可以有 9 对自定义的 key-value (超过则添加失败) ；

key 限长 50 字节，value 限长 200 字节，过长截断；

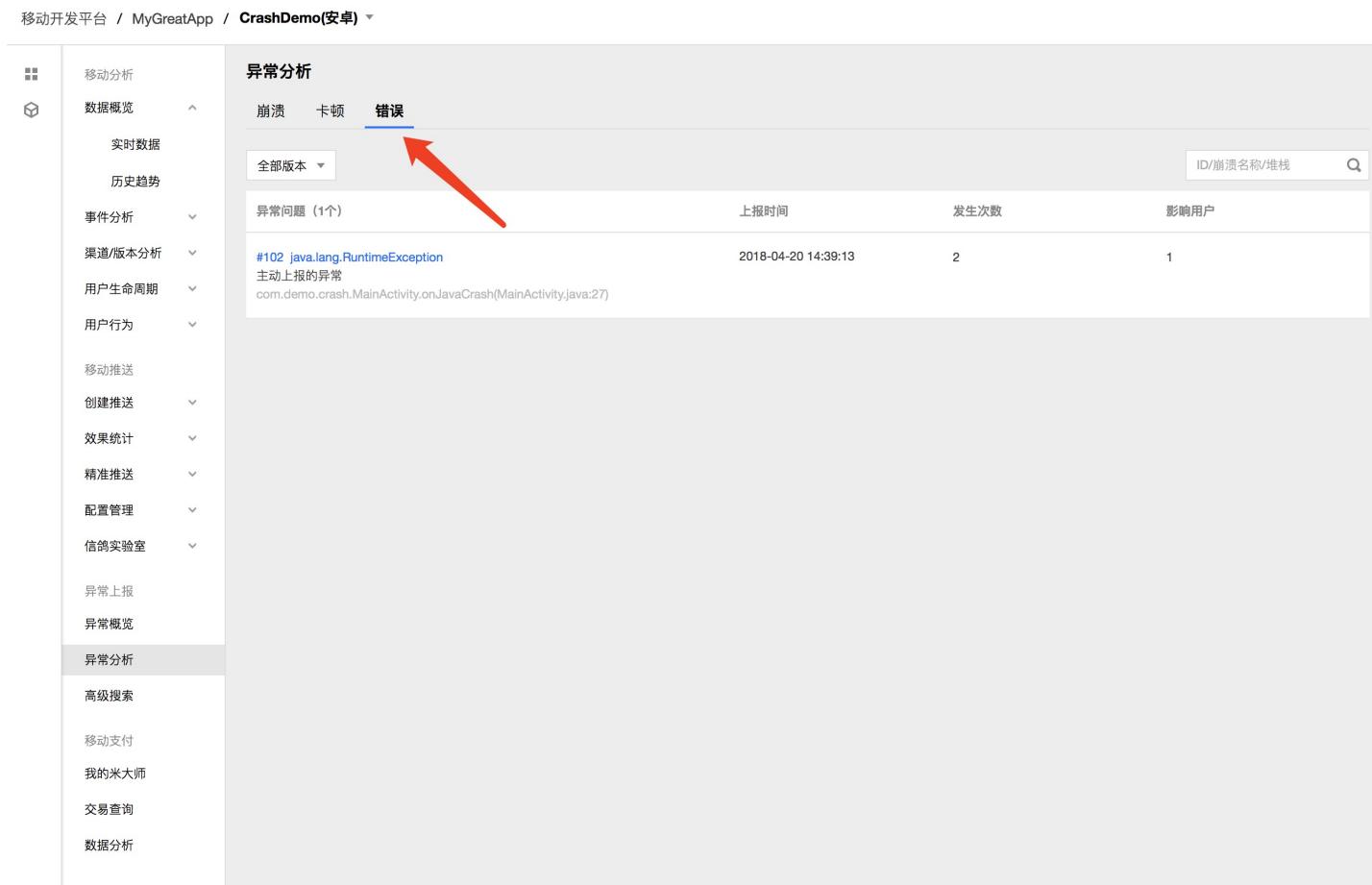
key 必须匹配正则 : [a-zA-Z[0-9]]+。

主动上报异常和错误

主动上报开发者 Catch 的异常您可能会关注某些重要异常的 Catch 情况，我们提供了上报这类异常的接口。

```
//TACCrashService
- (void) reportException:(NSError*)exception;
/** 上报一个错误
 @param error 错误信息
 */
- (void) reportError:(NSError*)error;
```

您可以在【异常上报】中点击【异常分析】，并在【错误】栏下查看主动上报的异常：



ID/崩溃名称/堆栈	上报时间	发生次数	影响用户
#102 java.lang.RuntimeException 主动上报的异常 com.demo.crash.MainActivity.onJavaCrash(MainActivity.java:27)	2018-04-20 14:39:13	2	1

常见问题

最近更新时间：2019-04-23 16:39:32

iOS 常见问题

接入 SDK 后崩溃没有上报？

1. 检查 Crashlytics 是否已经正常启动。
2. 网络是否可用。
3. 在测试时若之前有上报突然不上报了，可能是触发了 Crashlytics 的流量保护机制，请卸载 App 后再测试（并不影响真实用户 Crash 准确率）。
4. 是否有使用具有捕获 Crash 功能的其他第三方组件，若有将 Crashlytics 的初始化放在该组件后面。

检查卡顿的依据和上报时机是什么？

iOS 卡顿检查的依据是监控主线程 Runloop 的执行，观察执行耗时是否超过预定阈值(默认阈值为 3000ms) 在监控到卡顿时会立即记录线程堆栈到本地，在 App 从后台切换到前台时，执行上报。

Android 常见问题

为什么相同的用户一天上报了几百条 Crash？会消耗用户流量吗？

系统在进程发生 Crash 后，有可能会再次将它拉起（系统的、代码逻辑的），导致不停地 Crash。例如 Service、Recevier 等都比较容易出现自动拉起的情况。Crashlytics 有自己的流量保护机制，在保证数据准确性的前提下，会尽量减少用户流量消耗。

Crashlytics 上报 Crash 的时机是？

Crashlytics 会在发生 Crash 时尝试尽量上报。如果失败，会在下次启动选择合适的时机上报。

为什么我完成了 Crashlytics 集成，在页面还是看不到日志？

1. 检查 Crashlytics 是否已经正常启动。
2. 网络是否可用。
3. 在测试时若之前有上报突然不上报了，可能是触发了 Crashlytics 的流量保护机制，请卸载 App 后再测试（并不影响真实用户 Crash 准确率）。

每个版本都要配置符号表吗？

是的。每个 App 版本都需要对应一份符号表 (Java 的 Mapping 文件及 SO 的 Symbol 文件 , appleC++ 的 dsym 文件) , 配置只对设置的版本有效 , 重复配置将会覆盖。

配置了还原符号表 , 为什么显示日志仍然没有行号信息 ?

您的行号信息有可能在编译或混淆 APK 的时候已经丢失了。符号表中是没有行号信息的。

使用 Crashlytics 的库应用启动不了 , 提示发生 UnsatisfiedLinkError 异常 ?

通常是因为安装包中各 CPU 架构目录下所需要的动态库缺少导致的。

Crashlytics 收集了设备哪些信息 ? 有用户隐私吗 ?

Crashlytics 收集的信息都是为了更真实地为开发者还原 Crash 场景服务的 , 并不涉及用户隐私信息 :

- Crash 环境 : Crash 信息及线程堆栈 , ROM/RAM/SD 卡容量、网络/语言等状态
- App 信息 : 包名、版本、所属进程名
- 设备信息 : IMEI 等设备识别 , 用于判断 Crash 设备统计。