ALIBABA CLOUD

阿里云

开发指南

文档版本：20220406

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.iOS SDK

1.1. iOS SDK发布说明

V1.0.0.1（2019-10-16）

- 远程日志首次发布。

1.2. iOS SDK接入（Pod集成）

本文介绍如何通过Pod集成方式添加依赖接入远程日志服务 。

> 说明
>
> - 接入远程日志服务的iOS SDK可采用Pod集成和手动集成2种方式添加依赖。推荐使用Pod集成方式添加依赖，可大幅简化接入操作。
> - 如需使用手动集成方式添加依赖，操作方法参见：iOS SDK接入（手动集成）。

前提条件

- 已创建工作空间/应用。
- 已下载应用配置文件。

  具体内容参见：**移动研发平台 EMAS** > 快速入门。

使用限制

- 仅支持iOS 8.0及以上的App。
- 推荐使用CocoaPods管理依赖的Xcode项目。
- 日志在手机端上最多存储7天 。

接入概述

1. 添加依赖：采用Pod集成方式添加依赖。
2. 接入服务：添加iOS配置文件；引入头文件；初始化SDK ；设置日志拉取级别。
3. 执行编译：添加编译设置。
4. 打印日志：在业务代码中打印日志信息。

添加依赖

1：指定官方仓库和阿里云仓库。

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

2：添加依赖。

```
pod 'AlicloudTLog', '~> 1.0.0.2'
```

> **说明**
>
> 执行 `pod search AlicloudTLog` 命令，查询 `AlicloudTLog` 最新版本。

3：执行 `pod install` 或者 `pod update` 命令，获取SDK到项目中。

接入服务

1：将iOS配置文件 `AliyunEmasServices-Info.plist` 拷贝至项目根目录。

iOS配置文件获取方式参见：下载配置文件。

2：在 `AppDelegate.m` 文件中引入头文件：

```
#import <AlicloudTLog/AlicloudTlogProvider.h>
#import <AlicloudHAUtil/AlicloudHAProvider.h>
#import <TRemoteDebugger/TRDManagerService.h>
```

3：在 `AppDelegate.m` 文件的 `application:didFinishLaunchingWithOptions` 方法中，添加代码段，初始化SDK。

```
NSString *appVersion = @"x.x"; //配置项：App版本
NSString *channel = @"xx"; //配置项：渠道标记
NSString *nick = @"xx"; //配置项：用户昵称
[[AlicloudTlogProvider alloc] autoInitWithAppVersion:appVersion channel:channel nick:nick];
[AlicloudHAProvider start];
[TRDManagerService updateLogLevel:TLogLevelXXX]; //配置项：控制台可拉取的日志级别
```

参数说明：

参数	说明

参数	说明
appVersion	用于指定App的版本，上报至服务端，进行版本区分。 【数据类型】字符串 【格式要求】自定义 【取值范围】任意长度 **说明** 该参数值将在控制台显示为下拉列表选项，建议短小凝练。 【是否必选】是 【是否可为空】否 【默认值】无 【大小写敏感】是。例如，vx.x和Vx.x不是一个版本。 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。 【示例】`NSString *appVersion = @"1.0";`
channel	用于指定渠道标识，上报至服务端，进行渠道区分。 【数据类型】字符串 【取值范围】任意长度 【是否必选】是 【是否可为空】否 【默认值】无 【字符类型】英文大小写、数字。**说明**：不支持中文字符、特殊字符。 【示例】`NSString *channel = @"appstore";`

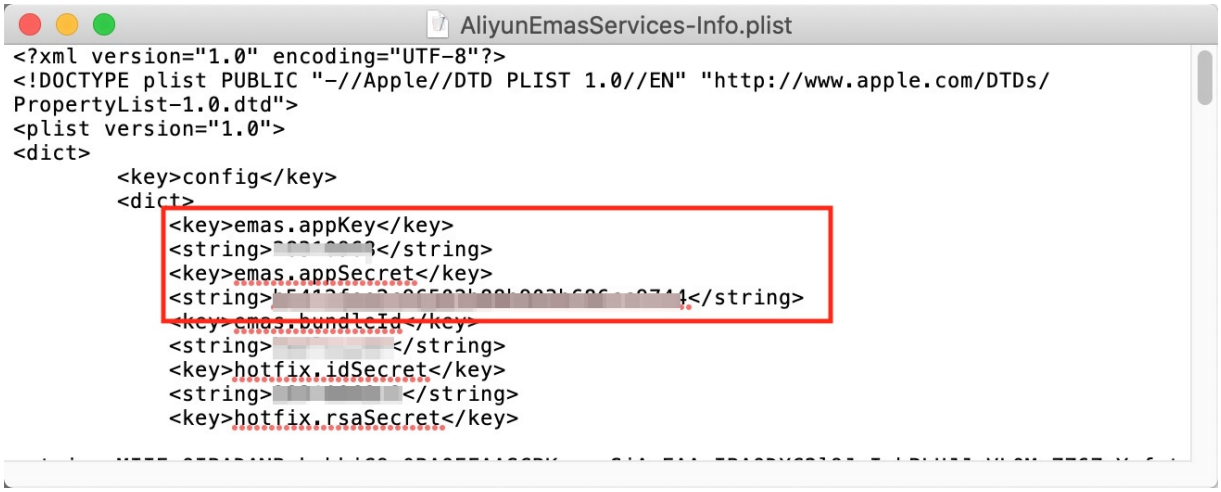
参数	说明
nick	用于指定用户昵称，上报至服务端，进行用户区分。后续可能依据该参数，进行数据检索。 【数据类型】字符串 【取值范围】任意长度 【是否必选】是 【是否可为空】否 【默认值】无 【字符类型】英文大小写、数字 **说明** 该参数不支持中文字符、特殊字符。 【命名规范】自定义 【示例】`NSString *nick = @"wldtest";`
TLogLevelXXX	用于设置控制台可拉取的日志级别。日志级别说明参见：术语解释。 【数据类型】枚举型 【取值范围】 • TLogLevelError：拉取Error级别的日志。 • TLogLevelWarn：拉取Warn/Error级别的日志。 • TLogLevelInfo：拉取Warn/Error/Info级别的日志。 • TLogLevelDebug：拉取Warn/Error/Info/Debug级别的日志。 【是否必选】否 【默认取值】TLogLevelInfo 【示例】`[TRDManagerService updateLogLevel:TLogLevelInfo];`

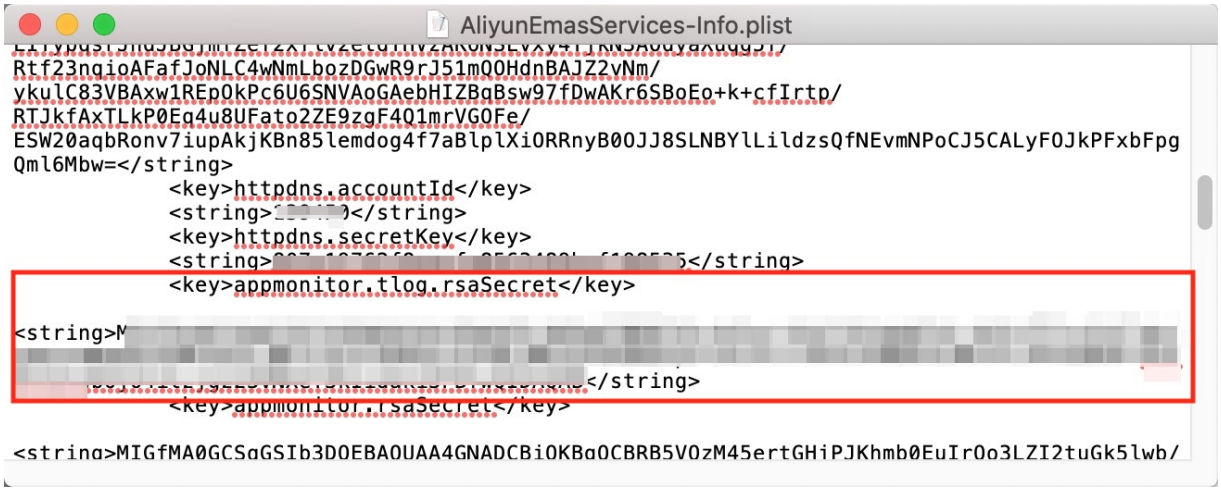
说明

推荐使用`autoInitWithAppVersion`接口接入服务。如需使用`initWithAppKey`接口接入服务，须手动配置`appKey` / `secret` / `tlogRsaSecret`参数。

打开iOS配置文件，查询参数取值：

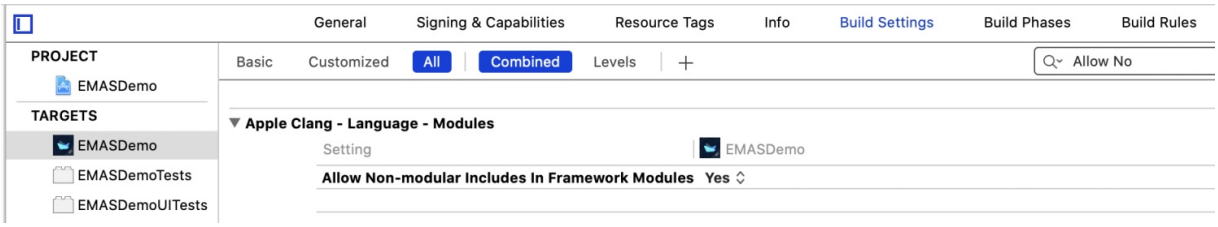
参数	配置文件字段	说明
appKey	emas.appKey	App标识。
secret	emas.appSecret	App认证信息。
tlogRsaSecret	appmonitor.tlog.rsaSecret	远程日志公钥。





执行编译

1：在项目的Build Setting中，将 `Allow Non-modular Includes In Framework Modules` 设置为 `YES` 。



2：执行编译。

> **说明**
>
> - 编译过程中如出现 `duplicate symbol` 类型错误，确认本地依赖与CocoaPods管理的依赖是否重复；如是，则删除本地依赖。
> - 如同时使用其他阿里云产品，可能会因为依赖中存在UTDID冲突，造成编译失败。解决办法参见：SDK UTDID冲突解决方案。

打印日志

1：如业务流程触发日志输出，需引入头文件：

```
#import <TRemoteDebugger/TLogBiz.h>
#import <TRemoteDebugger/TLogFactory.h>
#import <TRemoteDebugger/TRDManagerService.h>
```

2：在适当位置添加代码，输出日志信息。示例代码：

```
TLogBiz *log = [TLogFactory createTLogForModuleName:@"YourModuleName"];
[log error:@"error message"];
[log warn:@"warn message"];
[log debug:@"debug message"];
[log info:@"info message"];
```

选项	说明
YourModuleName	指定保存日志信息的模块的名称。
error/warn/debug/info message	根据实际场景，区分级别输出日志信息，便于后续按照级别进行日志信息查询。日志级别说明参见：术语解释。

上报日志

1：如业务流程触发日志输出，需引入头文件：

```
#import <AlicloudTLog/AlicloudTlogProvider.h>
```

2：在适当位置添加代码，主动上报当日日志信息。示例代码：

```
[AlicloudTlogProvider uploadTLog:@"COMMENT"]
```

> 说明
>
> - 函数用于主动上报当日缓存的日志信息。
> - 主动上报时系统会删除历史缓存日志，仅上报最新缓存的日志信息，避免相同日志重复上报。
> - 该接口需要使用最新版本的SDK，版本号ALICLOUDTLOG_VERSION @"1.0.1.1"。

参数	说明
COMMENT	上报日志时设置的备注信息。可用于记录待定位的问题。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】否

样例代码

远程日志服务iOS SDK接入工程样例参见：Demo工程。

1.3. iOS SDK接入（手动集成）

本文介绍如何通过手动集成方式添加依赖接入远程日志服务。

> 说明
>
> - 接入远程日志服务的iOS SDK可采用Pod集成和手动集成2种方式添加依赖。推荐使用Pod集成方式添加依赖，可大幅简化接入操作。
> - 如需使用Pod集成方式添加依赖，操作方法参见：iOS SDK接入（Pod集成）

前提条件

- 已创建工作空间/应用。
- 已下载iOS应用配置文件。
- 已下载SDK包，并确保包文件完整。SDK包文件列表如下：
  - AlicloudHAUtil.framework
  - AlicloudTLog.framework
  - AlicloudUtils.framework
  - AliHACore.framework
  - AliHALogEngine.framework
  - AliHAMethodTrace.framework
  - AliHAProtocol.framework
  - AliHASecurity.framework

- AliyunOSSiOS.framework
- RemoteDebugChannel.framework
- TBJSONModel.framework
- TBRest.framework
- TRemoteDebugger.framework
- UTDID.framework
- UTMini.framework
- ZipArchive.framework

具体内容参见：**移动研发平台 EMAS** > 快速入门

使用限制

- 仅支持iOS 8.0及以上的App。
- 日志在手机端上最多存储7天。

接入概述

1. 添加依赖：采用手动集成方式。
2. 接入服务：添加iOS配置文件；引入头文件；初始化SDK；设置日志拉取级别。
3. 执行编译：添加编译设置。
4. 打印日志：在业务代码中输出日志信息。

添加依赖

1：在Xcode中，将SDK目录中的framework文件拖入Target目录，在弹出框勾选 `Copy items if needed` 选项。

2：打开**Build Phases** > **Link Binary With Libraries**，添加Xcode自带的公共包文件：

- libc++.tbd
- libresolv.tbd
- SystemConfiguration.framework

接入服务

1：将iOS配置文件 `AliyunEmasServices-Info.plist` 拷贝至项目根目录。

iOS配置文件获取方式参见：下载配置文件。

2：在 `AppDelegate.m` 文件中引入头文件：

```
#import <AlicloudTLog/AlicloudTlogProvider.h>
#import <AlicloudHAUtil/AlicloudHAProvider.h>
#import <TRemoteDebugger/TRDManagerService.h>
```

3：在 `AppDelegate.m` 文件的 `application:didFinishLaunchingWithOptions` 方法中，添加代码段，初始化SDK。

```
NSString *appVersion = @"x.x"; //配置项：App版本
NSString *channel = @"xx"; //配置项：渠道标记
NSString *nick = @"xx"; //配置项：用户昵称
[[AlicloudTlogProvider alloc] autoInitWithAppVersion:appVersion channel:channel nick:nick];
[AlicloudHAProvider start];
[TRDManagerService updateLogLevel:TLogLevelXXX]; //配置项：控制台可拉取的日志级别
```

参数说明：

参数	说明
appVersion	用于指定App的版本，上报至服务端，进行版本区分。 【数据类型】字符串 【格式要求】自定义 【取值范围】任意长度。 **说明** 该参数值将在控制台显示为下拉列表选项，建议短小凝练。 【是否必选】是 【是否可为空】否 【默认值】无 【大小写敏感】是。例如，vx.x和Vx.x不是一个版本。 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。 【示例】 `NSString *appVersion = @"1.0";`

参数	说明
channel	用于指定渠道标识，上报至服务端，进行渠道区分。 【数据类型】字符串 【取值范围】任意长度 【是否必选】是 【是否可为空】否 【默认值】无 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。 【示例】`NSString *channel = @"appstore";`
nick	用于指定用户昵称，上报至服务端，进行用户区分。后续可能依据该参数，进行数据检索。 【数据类型】字符串 【取值范围】任意长度 【是否必选】是 【是否可为空】否 【默认值】无 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。 【命名规范】自定义 【示例】`NSString *nick = @"wldtest";`

参数	说明
TLogLevelXXX	用于设置控制台可拉取的日志级别。日志级别说明参见：术语解释 【数据类型】枚举型 【取值范围】 • TLogLevelError：拉取Error级别的日志。 • TLogLevelWarn：拉取Warn/Error级别的日志。 • TLogLevelInfo：拉取Warn/Error/Info级别的日志。 • TLogLevelDebug：拉取Warn/Error/Info/Debug级别的日志。 【是否必选】否 【默认取值】TLogLevelInfo 【示例】`[TRDManagerService updateLogLevel:TLogLevelInfo];`

> **说明**
>
> 推荐使用 `autoInitWithAppVersion` 接口接入服务。如需使用 `initWithAppKey` 接口接入服务，须手动配置 `appKey` / `secret` / `tlogRsaSecret` 参数。

打开iOS配置文件，查询参数取值：

参数	配置文件字段	说明
appKey	emas.appKey	App标识。
secret	emas.appSecret	App认证信息。
tlogRsaSecret	appmonitor.tlog.rsaSecret	远程日志公钥。

AliyunEmasServices-Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>config</key>
        <dict>
            <key>emas.appKey</key>
            <string>[redacted]</string>
            <key>emas.appSecret</key>
            <string>[redacted]</string>
            <key>emas.bundleId</key>
            <string>[redacted]</string>
            <key>hotfix.idSecret</key>
            <string>[redacted]</string>
            <key>hotfix.rsaSecret</key>
```

AliyunEmasServices-Info.plist

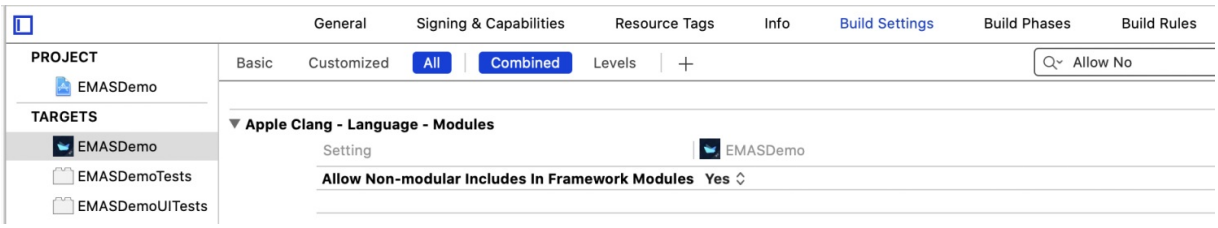
```
[redacted]
Rtf23ngioAFafJoNLC4wNmLbozDGwR9rJ51mQOHdnBAJZ2vNm/
ykulC83VBAxw1REpOkPc6U6SNVAoGAebHIZBqBsw97fDwAKr6SBoEo+k+cfIrtp/
RTJkfAxTLkP0Eq4u8UFato2ZE9zgF4Q1mrVGOFe/
ESW20aqbRonv7iupAkjKBn85lemdog4f7aBlplXiORRnyB0OJJ8SLNBYlLildzsQfNEvmNPoCJ5CALyFOJkPFxbFpg
Qml6Mbw=</string>
            <key>httpdns.accountId</key>
            <string>[redacted]</string>
            <key>httpdns.secretKey</key>
            <string>[redacted]</string>
            <key>appmonitor.tlog.rsaSecret</key>

<string>M[redacted]</string>
            <key>appmonitor.rsaSecret</key>

<string>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCBRB5VOzM45ertGHiPJKhmb0EuIrOo3LZI2tuGk5lwb/
```

执行编译

1：在项目的Build Setting中，将 `Allow Non-modular Includes In Framework Modules` 设置为 `YES` 。



2：执行编译。

> **说明**
>
> - 编译过程中如出现 `duplicate symbol` 类型错误，确认本地依赖与CocoaPods管理的依赖是否重复；如是，则删除本地依赖。
> - 如同时使用其他阿里云产品，可能会因为依赖中存在UTDID冲突，造成编译失败。解决办法参见：SDK UTDID冲突解决方案

打印日志

1：如业务流程触发日志输出，需引入头文件：

```
#import <TRemoteDebugger/TLogBiz.h>
#import <TRemoteDebugger/TLogFactory.h>
#import <TRemoteDebugger/TRDManagerService.h>
```

2：在适当位置添加代码，输出日志信息。示例代码：

```
TLogBiz *log = [TLogFactory createTLogForModuleName:@"YourModuleName"];
[log error:@"error message"];
[log warn:@"warn message"];
[log debug:@"debug message"];
[log info:@"info message"];
```

选项	说明
YourModuleName	指定保存日志信息的模块的名称。
error/warn/debug/info message	根据实际场景，区分级别输出日志信息，便于后续按照级别进行日志信息查询。日志级别说明参见：术语解释

上报日志

1：如业务流程触发日志输出，需引入头文件：

```
#import <AlicloudTLog/AlicloudTlogProvider.h>
```

2：在适当位置添加代码，主动上报当日日志信息。示例代码：

```
[AlicloudTlogProvider uploadTLog:@"COMMENT"]
```

> 说明
>
> - 函数用于主动上报当日缓存的日志信息。
> - 主动上报时系统会删除历史缓存日志，仅上报最新缓存的日志信息，避免相同日志重复上报。
> - 该接口需要使用最新版本的SDK，版本号ALICLOUDTLOG_VERSION @"1.0.1.1"。

参数	说明

参数	说明
COMMENT	上报日志时设置的备注信息。可用于记录待定位的问题。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】否

样例代码

远程日志服务iOS SDK接入工程样例参见：Demo工程

1.4. iOS SDK接入验证

iOS SDK接入过程中如出现问题，可按照**编译阶段常见问题排查**所列要点，进行排查；iOS SDK接入之后，可按照**运行阶段业务逻辑验证**进行接入验证。

编译阶段常见问题排查

1：手动集成场景，确认已下载使用最新版本的SDK包，且内容完整无缺失。

2：确认已正确添加依赖，且不存在依赖冲突。

3：确认已添加Xcode自带的公共包文件：

- libc++.tbd
- libresolv.tbd
- SystemConfiguration.framework

4：确认已将iOS配置文件 `AliyunEmasServices-Info.plist` 拷贝至项目根目录。

5：确认已在初始化SDK代码中正确引用头文件。

6：如使用 `initWithAppKey` 接口接入服务，确认已正确设置 `appKey` / `secret` / `tlogRsaSecret` 。

7：确认编译设置正确，即将 `Allow Non-modular Includes In Framework Modules` 设置为 `YES` 。

运行阶段业务逻辑验证

步骤

1：识别设备

【手机端】

1：重新安装并启动App。

2：执行代码，获取设备ID。

```
NSLog(@"\n ==== %@",[UTDevice utdid]);
```

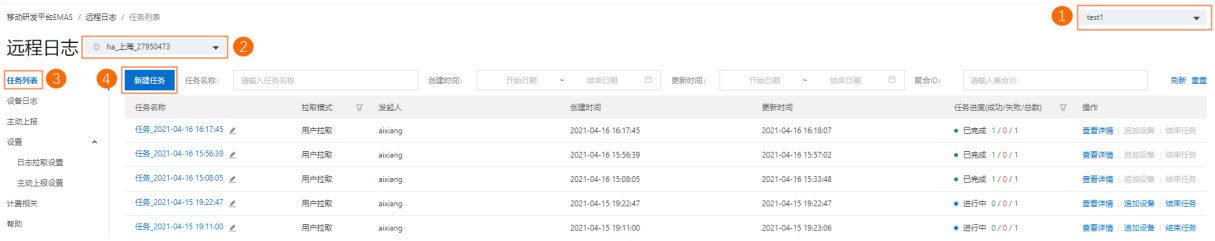
【控制台】确认识别手机端设备。

1：打开远程日志控制台。具体操作参见：打开控制台页面。

2：在下拉列表选择产品和应用。

3：从左侧导航栏，选择**任务列表**页签。

4：在**任务列表**页签，单击**新建任务**按钮，打开**新建任务**页面。



5：在左侧导航栏，分别选择**设备日志**和**主动上报**，输入**设备ID/名称**，单击**刷新**按钮，查看手机端设备是否已被识别。

如未识别，可能的原因是：

- 网络未连接/延迟：确认手机端已联网，稍作等待，刷新再试。
- SDK接入失败/SDK未获取数据/数据发送失败/后端问题：联系技术支持解决。

步骤2：拉取日志

【手机端】确认日志上传成功。

1：操作App进行前后台切换，触发日志输出。

2：查看客户端日志，搜索关键字 `tlog` ，查看相关信息，确认日志是否上传成功。

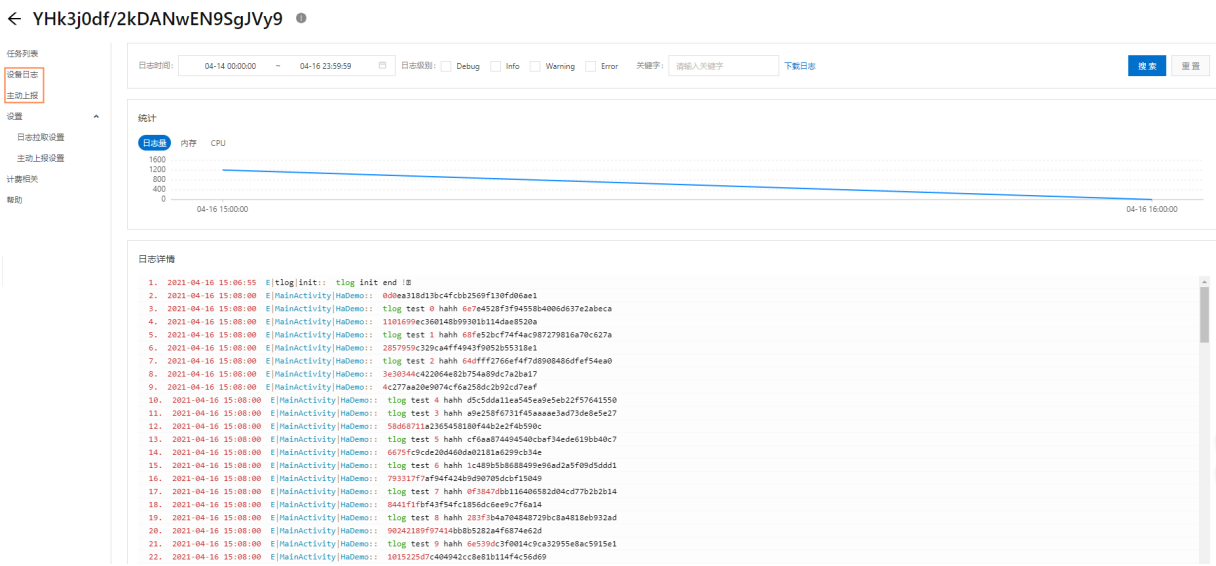
【控制台】确认正确拉取日志。

1：创建日志拉取任务，拉取指定手机端设备的日志。具体操作参见：新建任务。

2：查看任务，确认日志拉取任务已完成。具体操作参见：查看/管理任务。

3：在左侧导航栏，分别选择**设备日志**和**主动上报**，确认指定手机端设备的日志已拉取成功。

- **设备日志**的具体操作参见：查看日志。
- **主动上报**的具体操作请参见：查看主动上报日志。



如预期日志未被成功拉取，可能的原因是：SDK接入失败/SDK未获取数据/数据发送失败/后端问题。联系技术支持解决。

2.Android SDK

2.1. Android SDK发布说明

V1.1.4.4（2022-03-28）

- 提升兼容性。

V1.1.4.3（2022-02-14）

- 支持安装包名称包含下划线（_）。

V1.1.3.2（2021-04-01）

- 支持X86_64。

V1.1.3.1（2020-12-18）

- 增加更新昵称。

V1.1.2.3（2020-02-25）

- 支持arm64-v8a。
- 修复Android 9及以上版本的OSS上传失败问题。

V1.1.2.1（2019-10-23）

- 首次发布。

2.2. Android SDK接入（Maven集成）

本文介绍如何通过Maven集成方式添加依赖接入远程日志服务。

> 说明
>
> - 接入远程日志服务的Android SDK可采用Maven集成和本地集成2种方式添加依赖。推荐使用Maven集成方式添加依赖，可大幅简化接入操作。
> - 如需使用本地集成方式添加依赖，操作方法参见：Android SDK接入（本地集成）。

前提条件

- 已创建工作空间/应用。具体操作参见：**移动研发平台 EMAS** > 快速入门。
- 已打开控制台页面。具体操作参见：打开控制台页面。

使用限制

- 仅支持Android 4.2及以上版本。
- 仅支持armeabi-v7a/arm64-v8a/X86/X86_64架构。
- 日志在手机端最多存储7天。

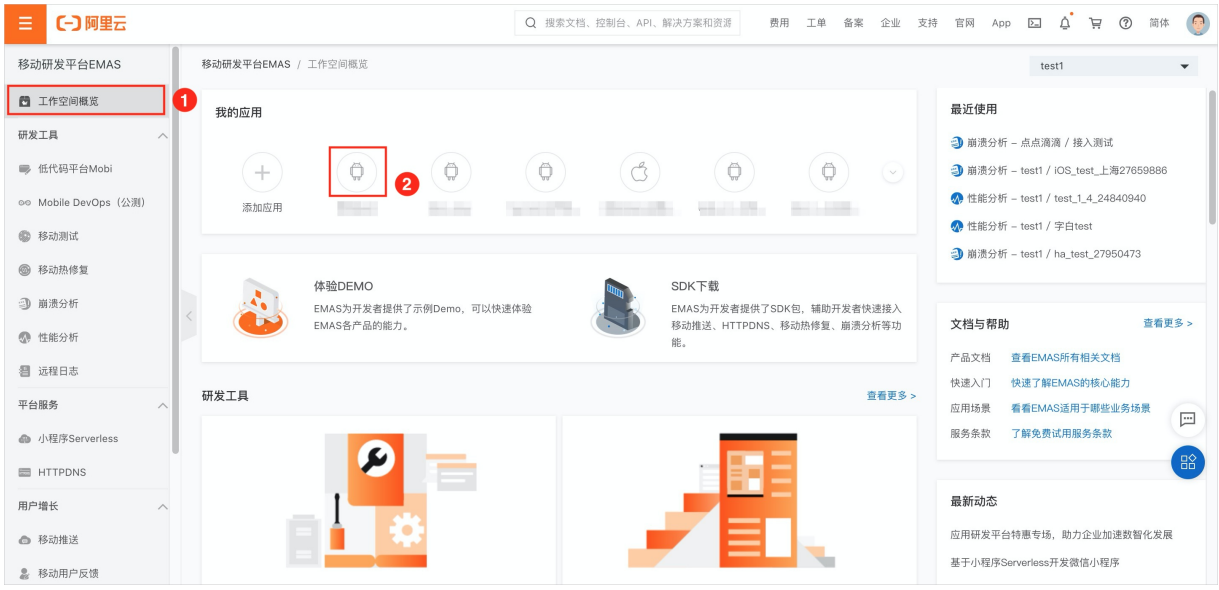
接入概述

1. 准备：获取AppKey/AppSecret；下载Android配置文件，获取远程日志公钥。
2. 添加依赖：采用Maven集成方式。

3. 接入服务：添加自定义Application，以及初始化代码；配置ABI；设置日志拉入级别。
4. 打印日志：引入头文件；在代码中打印日志信息。
5. 上报日志：引入头文件；上报缓存的日志信息到远程日志服务。
6. 混淆配置：如App对代码进行乱序混淆，则修改混淆配置文件。
7. 编译。

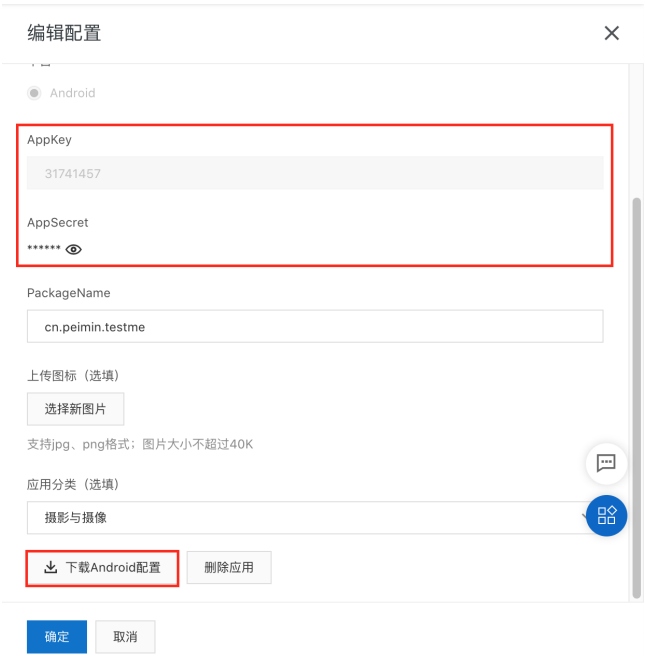
准备

1：在控制台 > **工作空间概览**页面 > **我的应用**区域，单击Android应用图标，打开指定Android应用**编辑配置**右侧栏。



2：在**编辑配置**右侧栏，查看App的AppKey/AppSecret。

参数	说明
AppKey	用于唯一标识App。由系统生成，8位数字。
AppSecret	用于认证App。由系统生成，32位字符串。



3：在**编辑配置**右侧栏，单击**下载Android配置**按钮，下载App配置文件：`aliyun-emas-services.json`

4：打开配置文件，查询 `appmonitor.tlog.rsaSecret` 字段内容，即为远程日志公钥。

注意

- 为避免在日志中泄漏参数 `appkey` / `appsecret` / `rsaSecret` 或App运行过程中产生的数据，建议线上版本关闭SDK调试日志。
- 由于所有用户使用统一的SDK接入，在接入过程中需要在代码中设置 `appkey` / `appsecret` / `rsaSecret` 参数，而此类参数与计量计费密切相关，为防止恶意反编译获取参数造成信息泄漏，建议您开启混淆，并进行App加固后再发布上线。

添加依赖

1：在项目 `build.gradle` 中的 `repositories` 节点内添加阿里云Maven仓库地址。

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

2：在项目 `build.gradle` 中的 `dependencies` 节点内添加依赖：

```
compile('com.aliyun.ams:alicloud-android-ha-adapter:1.1.5.2-open@aar') {
    transitive=true
}
compile('com.aliyun.ams:alicloud-android-tlog:1.1.4.4-open@aar') {
    transitive=true
}
```

3：在 `build.gradle` 项目文件的 `defaultConfig` 节点内根据实际需要配置ABI。

```
ndk {
        abiFilters 'armeabi'    //配置项。可选取值：armeabi-v7a/arm64-v8a/x86/x86_64
}
```

> **注意**
>
> 远程日志服务目前仅支持armeabi-v7a/arm64-v8a/X86/X86_64架构。

接入服务

1. 定义 `Application` 类，并编写 `onCreate` 方法启动服务：

```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        initHa();
    }
    private void initHa() {
         AliHaConfig config = new AliHaConfig();
         config.appKey = "xxxxxxxx"; //配置项：appkey
         config.appVersion = "x.xx"; //配置项：应用的版本号
         config.appSecret = "xxxxxxxxxxxx"; //配置项：appsecret
         config.channel = "mqc_test"; //配置项：应用的渠道号标记，自定义
         config.userNick = null; //配置项：用户的昵称
         config.application = this; //配置项：应用指针
         config.context = getApplicationContext(); //配置项：应用上下文
         config.isAliyunos = false; //配置项：是否为yunos
         config.rsaPublicKey = "xxxxxxx"; //配置项：tlog公钥
         AliHaAdapter.getInstance().addPlugin(Plugin.tlog);
         AliHaAdapter.getInstance().openDebug(true);
         AliHaAdapter.getInstance().start(config);
         TLogService.updateLogLevel(TLogLevel.XXXXXX); //配置项：控制台可拉取的日志级别
    }
}
```

配置说明如下：

参数	说明
config.appKey	用于指定App的AppKey。 【数据类型】字符串 【如何获取】请参见：接入服务中的步骤1。 【是否必选】是 【是否可为空】否 【默认值】无

参数	说明
config.appVersion	用于设置App的版本号。 【数据类型】字符串 【格式要求】自定义 【取值范围】任意长度。 说明 该参数值将在控制台显示为下拉列表选项，建议短小凝练。 【是否必选】是 【是否可为空】否 【默认值】无 【大小写敏感】是。例如，vx.x和Vx.x不是一个版本。 【字符类型】英文大小写、数字。 说明 该参数不支持中文字符、特殊字符。
config.appSecret	用于指定App的AppSecret。 【数据类型】字符串 【如何获取】请参见：接入服务中的步骤1。 【是否必选】是 【是否可为空】否 【默认值】无
config.channel	用于设置渠道标识，上报至服务端，进行渠道区分。 【数据类型】字符串 【取值范围】任意长度 【是否必选】否 【是否可为空】是 【默认值】无 【字符类型】英文大小写、数字。 说明 该参数不支持中文字符、特殊字符。

参数	说明
config.userNick	用于设置用户昵称，上报至服务端，进行用户区分。后续可能依据该参数，进行数据检索。 【数据类型】字符串 【取值范围】任意长度 【是否必选】否 【是否可为空】是 【默认值】无 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。 【命名规范】自定义
config.application	用于指定本应用。 **注意** 不能指向其他应用。 【数据类型】对象 【是否必选】是 【是否可为空】否 【默认值】无
config.context	用于指定App的上下文对象，设置 `getApplicationContext();` 即可。 【数据类型】对象 【是否必选】是 【是否可为空】否 【默认值】无

参数	说明
config.isAliyunos	用于判断App所在平台是否为YunOS。 【数据类型】布尔型 【取值范围】false/true 【是否必选】否 【是否可为空】是 【默认值】false
config.rsaPublicKey	用于指定远程日志公钥。 【数据类型】字符串 【如何获取】请参见：接入服务中的步骤1。 【是否必选】是 【是否可为空】否 【默认值】无
TLogLevel.XXXXXX	用于全局设置控制台可拉取的日志的级别。 【数据类型】枚举型 【取值范围】 • VERBOSE：可拉取所有级别的日志。 • DEBUG：可拉取DEBUG/INFO/WARN/ERROR级别的日志。 • INFO：可拉取INFO/WARN/ERROR级别的日志。 • WARN：可拉取WARN/ERROR级别的日志。 • ERROR：可拉取ERROR级别的日志。 【是否必选】是 【默认取值】ERROR 【配置说明】 • `TLogService.updateLogLevel()` 函数可选调用，如未调用，则全局默认可拉取的日志级别为ERROR。 • 日志级别说明参见：术语解释。

2：在 `AndroidManifest.xml` 中添加代码段注册 `Application` 。

```
<application
    android:name=".MyApplication"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >
    ...
</application>
```

打印日志

1：如业务流程触发日志输出，需引入头文件：

```
import com.alibaba.ha.adapter.service.tlog.TLogService;
```

2：在适当位置添加代码，打印日志信息。示例代码：

```
TLogService.logv("MODEL","TAG","MESSAGE");
TLogService.logd("MODEL","TAG","MESSAGE");
TLogService.logi("MODEL","TAG","MESSAGE");
TLogService.logw("MODEL","TAG","MESSAGE");
TLogService.loge("MODEL","TAG","MESSAGE");
```

函数： `TLogService.<LogLevel>(<MODEL>,<TAG,<MESSAGE>);`

说明： 用于输出指定级别的日志信息。

参数	说明
LogLevel	指定拉取的日志级别。日志级别说明参见：术语解释。 【数据类型】枚举型 【取值范围】 • logv：输出VERBOSE（调试详情）级别的日志。 • logd：输出DEBUG（调试信息）级别的日志。 • logi：输出INFO（一般信息）级别的日志。 • logw：输出WARN（警告信息）级别的日志。 • loge：输出ERROR（错误信息）级别的日志。 【配置说明】输出的日志是否可以被控制台拉取，取决于 `TLogService.updateLogLevel()` 函数的参数设置。例如，当 `TLogService.updateLogLevel(TLogLevel.WARN)` 时，则控制台拉取不到通过logv/logd/logi输出的日志。

参数	说明
MODEL	用于设置输出日志内容的功能模块，便于后续根据来源筛选日志。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】是 【是否大小写敏感】否 【示例】“推送功能模块Push”
TAG	用于设置日志的关键字，便于后续根据标签筛选日志。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】是 【是否大小写敏感】否 【示例】“推送功能模块收到了推送Push.receive，推送功能模块点击了推送Push.click”
MESSAGE	用于输出日志信息。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】是

上报日志

1：如业务流程触发日志输出，需引入头文件：

```
import com.alibaba.ha.adapter.service.tlog.TLogService;
```

2：在适当位置添加代码，主动上报当日日志信息。示例代码：

```
TLogService.positiveUploadTlog("COMMENT");
```

> **说明**
>
> - 函数 `TLogService.positiveUploadTlog(<COMMENT>);` ，用于主动上报当日缓存的日志信息。
> - 主动上报时系统会删除历史缓存日志，仅上报最新缓存的日志信息，避免相同日志重复上报。

参数	说明
COMMENT	上报日志时设置的备注信息。可用于记录待定位的问题。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】否

混淆配置

如App对代码进行乱序混淆，则在混淆配置文件中添加代码段：

```
-keep class com.taobao.tao.log.**{*;}
-keep public class * extends com.taobao.android.tlog.protocol.model.request.base.FileInfo{*
;}
-keepattributes Exceptions,InnerClasses,Signature,Deprecated,SourceFile,LineNumberTable,*An
notation*,EnclosingMethod
```

编译

如同时使用其他阿里云产品，可能会因为依赖中存在UTDID冲突，造成编译失败。解决办法参见：SDK UTDID冲突解决方案。

样例代码

远程日志服务Android SDK接入工程样例参见：Demo工程。

2.3. Android SDK接入（本地集成）

本文介绍如何通过本地集成方式添加依赖接入远程日志服务。

> 说明
>
> - 接入远程日志服务的Android SDK可采用Maven集成和本地集成2种方式添加依赖。推荐使用Maven集成方式添加依赖，可大幅简化接入操作。
> - 如需使用Maven集成方式添加依赖，操作方法参见：Android SDK接入（Maven集成）。

前提条件

• 已创建工作空间/应用。具体内容参见：**移动研发平台 EMAS** > 快速入门。

使用限制

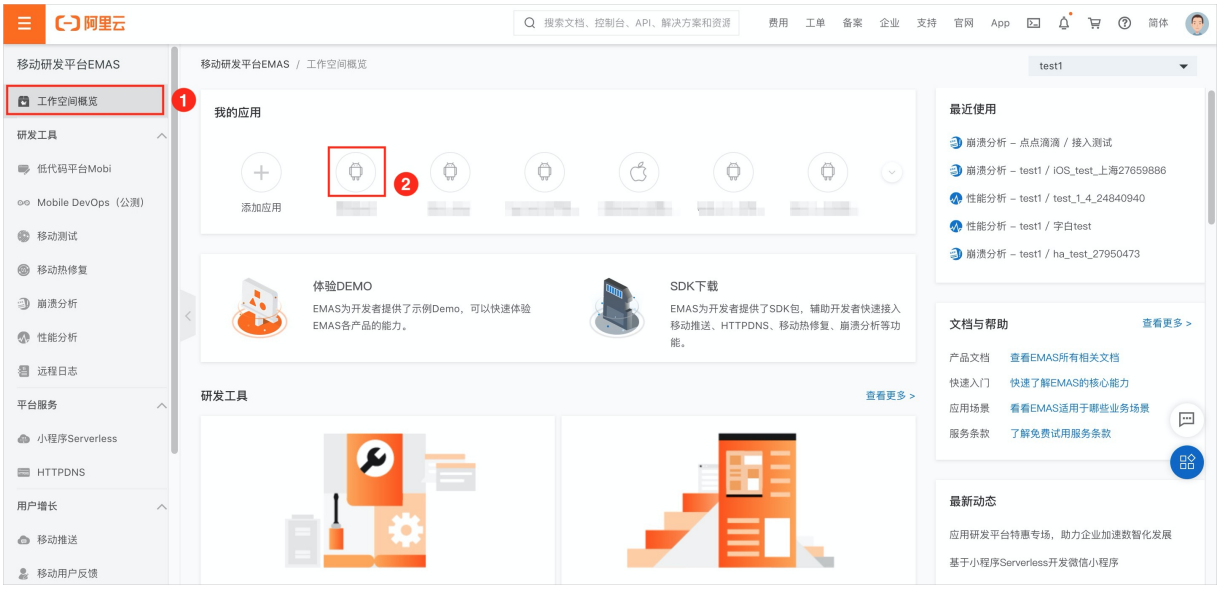
- 仅支持Android 4.2及以上版本。
- 仅支持armeabi-v7a/arm64-v8a/X86/X86_64架构。
- 日志在手机端最多存储7天。

接入概述

1. 准备：获取AppKey/AppSecret；下载Android配置文件，获取远程日志公钥；下载SDK包。
2. 添加依赖：采用本地集成方式。
3. 接入服务：添加自定义Application，以及初始化代码；配置ABI；设置日志拉入级别。
4. 打印日志：引入头文件；在代码中打印日志信息。
5. 上报日志：引入头文件；上报缓存的日志信息到远程日志服务。
6. 混淆配置：如App对代码进行乱序混淆，则修改混淆配置文件。
7. 编译。

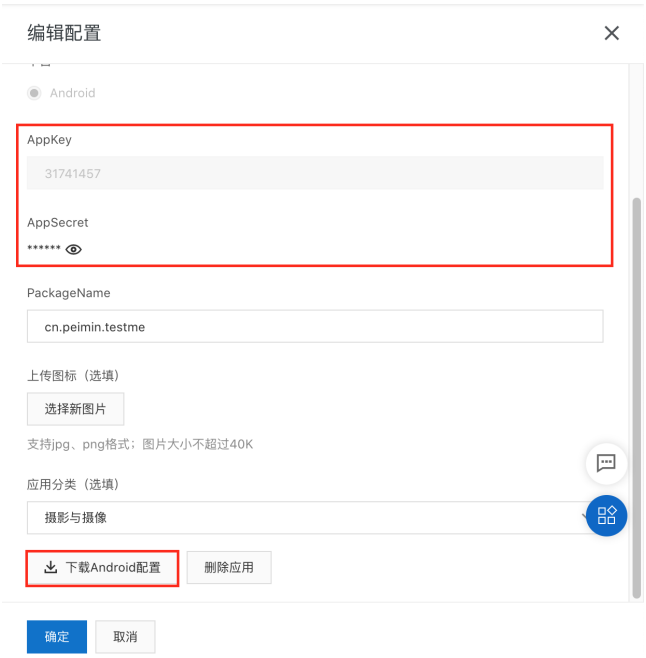
准备

1：在控制台 > **工作空间概览**页面 > **我的应用**区域，单击Android应用图标，打开指定Android应用**编辑配置**右侧栏。



2：在**编辑配置**右侧栏，查看App的AppKey/AppSecret。

参数	说明
AppKey	用于唯一标识App。由系统生成，8位数字。
AppSecret	用于认证App。由系统生成，32位字符串。



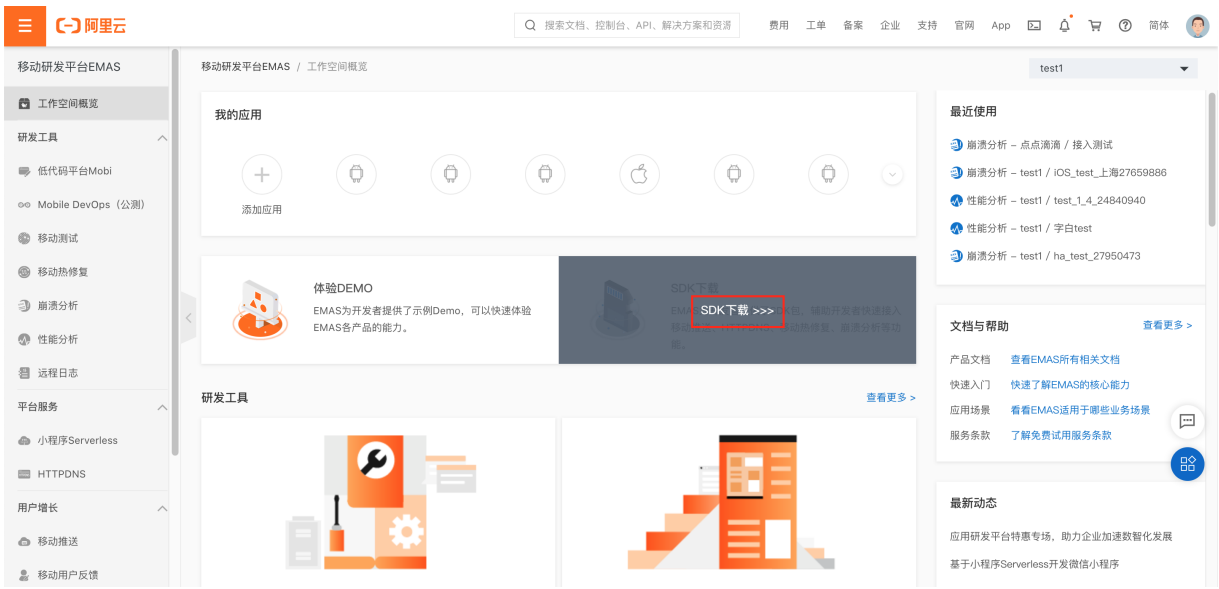
3：在**编辑配置**右侧栏，单击**下载Android配置**按钮，下载App配置文件：`aliyun-emas-services.json`

4：打开配置文件，查询 `appmonitor.tlog.rsaSecret` 字段内容，即为远程日志公钥。

注意

- 为避免在日志中泄漏参数 `appkey` / `appsecret` / `rsaSecret` 或App运行过程中产生的数据，建议线上版本关闭SDK调试日志。
- 由于所有用户使用统一的SDK接入，在接入过程中需要在代码中设置 `appkey` / `appsecret` / `rsaSecret` 参数，而此类参数与计量计费密切相关，为防止恶意反编译获取参数造成信息泄漏，建议您开启混淆，并进行App加固后再发布上线。

5：返回**工作空间概览**页面，单击**SDK下载**链接，打开**SDK下载**右侧栏。



6：在**SDK下载**右侧栏，选中**远程日志**复选框，单击**下载Android版本**按钮，下载远程日志服务的SDK包。

> **说明**
>
> 在远程日志行的Android版列，单击版本号链接，可查看版本变更记录。

3：检查SDK包，确保内容完整。

SDK包文件列表如下：

- alicloud-android-ha-adapter-1.1.5.2-open.aar
- alicloud-android-ha-core-1.1.0.6.1-open.aar
- alicloud-android-ha-protocol-1.1.2.0-open.aar
- alicloud-android-ha-tbrest-1.1.5.0-open.aar
- alicloud-android-ha-tlog-message-rpc-1.1.3.1-open.aar
- alicloud-android-ha-tlog-native-1.1.1.0-open.aar
- alicloud-android-ha-tlog-protocol-1.1.0.9-open.aar

- alicloud-android-ha-tlog-uploader-oss-1.1.0.8-open.aar
- alicloud-android-tlog-1.1.4.4-open.aar
- alicloud-android-utdid-2.5.3.jar
- alicloud-android-setting-service-1.0.0.aar
- alicloud-android-logger-1.0.2.aar
- fastjson-1.1.54.android.jar
- okhttp-3.4.1.jar
- okio-1.9.0.jar
- oss-android-sdk-2.9.3.aar

> **说明**
>
> 如存在名称相同，版本号不同的包文件，添加最高版本的包文件即可。

添加依赖

1：将SDK包内所有文件拷贝至项目的libs目录下。

2：在 `build.gradle` 项目文件中，添加本地SDK文件目录地址。

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

3：在 `build.gradle` 项目文件的 `dependencies{}` 代码段，添加SDK依赖。

```
//1、本地jar库引入
compile fileTree(include:['*.jar'],dir:'libs')

//2、公共库
compile(name:'alicloud-android-ha-adapter-1.1.5.2-open',ext:'aar')
compile(name:'alicloud-android-ha-core-1.1.0.6.1-open',ext:'aar')
compile(name:'alicloud-android-ha-protocol-1.1.0.9-open',ext:'aar')
compile(name:'alicloud-android-ha-tbrest-1.1.5.0-open',ext:'aar')
compile(name:'alicloud-android-utdid-2.5.3',ext:'jar')
compile(name:'fastjson-1.1.54.android',ext:'jar')


//3、移动日志，不接入可注释掉
compile(name:'alicloud-android-tlog-1.1.4.4-open',ext:'aar')
compile(name:'alicloud-android-ha-tlog-message-rpc-1.1.3.1-open',ext:'aar')
compile(name:'alicloud-android-ha-tlog-uploader-oss-1.1.0.8-open',ext:'aar')
compile(name:'alicloud-android-ha-tlog-protocol-1.1.0.9-open',ext:'aar')
compile(name:'alicloud-android-ha-tlog-native-1.1.1.0-open',ext:'aar')
compile(name:'alicloud-android-setting-service-1.0.0',ext:'aar')
compile(name:'alicloud-android-logger-1.0.2',ext:'aar')
compile(name:'okhttp-3.4.1',ext:'jar')
compile(name:'okio-1.9.0',ext:'jar')
compile(name:'oss-android-sdk-2.9.3',ext:'aar')
```

接入服务

1、定义 `Application` 类，并编写 `onCreate` 方法启动服务：

```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        initHa();
    }
    private void initHa() {
         AliHaConfig config = new AliHaConfig();
         config.appKey = "xxxxxxxx"; //配置项：appkey
         config.appVersion = "x.xx"; //配置项：应用的版本号
         config.appSecret = "xxxxxxxxxxxx"; //配置项：appsecret
         config.channel = "mqc_test"; //配置项：应用的渠道号标记，自定义
         config.userNick = null; //配置项：用户的昵称
         config.application = this; //配置项：应用指针
         config.context = getApplicationContext(); //配置项：应用上下文
         config.isAliyunos = false; //配置项：是否为yunos
         config.rsaPublicKey = "xxxxxxx"; //配置项：tlog公钥
         AliHaAdapter.getInstance().addPlugin(Plugin.tlog);
         AliHaAdapter.getInstance().openDebug(true);
         AliHaAdapter.getInstance().start(config);
         TLogService.updateLogLevel(TLogLevel.XXXXXX); //配置项：控制台可拉取的日志级别
    }
}
```

配置说明如下：

参数	说明
config.appKey	用于指定App的AppKey。 【数据类型】字符串 【如何获取】参见：步骤1 【是否必选】是 【是否可为空】否 【默认值】无
config.appVersion	用于设置App的版本号。 【数据类型】字符串 【格式要求】自定义 【取值范围】任意长度。 说明 该参数值将在控制台显示为下拉列表选项，建议短小凝练。 【是否必选】是 【是否可为空】否 【默认值】无 【大小写敏感】是。例如，vx.x和Vx.x不是一个版本。 【字符类型】英文大小写、数字。 说明 该参数不支持中文字符、特殊字符。
config.appSecret	用于指定App的AppSecret。 【数据类型】字符串 【如何获取】参见：步骤1 【是否必选】是 【是否可为空】否 【默认值】无

参数	说明
config.channel	用于设置渠道标识，上报至服务端，进行渠道区分。 【数据类型】字符串 【取值范围】任意长度 【是否必选】否 【是否可为空】是 【默认值】无 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。
config.userNick	用于设置用户昵称，上报至服务端，进行用户区分。后续可能依据该参数，进行数据检索。 【数据类型】字符串 【取值范围】任意长度 【是否必选】否 【是否可为空】是 【默认值】无 【字符类型】英文大小写、数字。 **说明** 该参数不支持中文字符、特殊字符。 【命名规范】自定义
config.application	用于指定本应用。 **注意** 不能指向其他应用。 【数据类型】对象 【是否必选】是 【是否可为空】否 【默认值】无

参数	说明
config.context	用于指定App的上下文对象，设置 `getApplicationContext();` 即可。 【数据类型】对象 【是否必选】是 【是否可为空】否 【默认值】无
config.isAliyunos	用于判断App所在平台是否为YunOS。 【数据类型】布尔型 【取值范围】false/true 【是否必选】否 【是否可为空】是 【默认值】false
config.rsaPublicKey	用于指定远程日志公钥。 【数据类型】字符串 【如何获取】参见：步骤1 【是否必选】是 【是否可为空】否 【默认值】无

参数	说明
TLogLevel.XXXXXX	用于全局设置控制台可拉取的日志的级别。 【数据类型】枚举型 【取值范围】 • VERBOSE：可拉取所有级别的日志。 • DEBUG：可拉取DEBUG/INFO/WARN/ERROR级别的日志。 • INFO：可拉取INFO/WARN/ERROR级别的日志。 • WARN：可拉取WARN/ERROR级别的日志。 • ERROR：可拉取ERROR级别的日志。 【是否必选】是 【默认取值】ERROR 【配置说明】 • `TLogService.updateLogLevel()` 函数可选调用，如未调用，则全局默认可拉取的日志级别为ERROR。 • 日志级别说明参见：术语解释。

2：在 `AndroidManifest.xml` 中添加代码段注册 `Application` 。

```
<application
    android:name=".MyApplication"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >
    ...
</application>
```

打印日志

1：如业务流程触发日志输出，需引入头文件：

```
import com.alibaba.ha.adapter.service.tlog.TLogService;
```

2：在适当位置添加代码，输出日志信息。示例代码：

```
TLogService.logv("MODEL","TAG","MESSAGE");
TLogService.logd("MODEL","TAG","MESSAGE");
TLogService.logi("MODEL","TAG","MESSAGE");
TLogService.logw("MODEL","TAG","MESSAGE");
TLogService.loge("MODEL","TAG","MESSAGE");
```

函数： `TLogService.<LogLevel>(<MODEL>,<TAG,<MESSAGE>);`

说明： 用于输出指定级别的日志信息。

参数	说明
LogLevel	指定拉取的日志级别。日志级别说明参见：术语解释。 【数据类型】枚举型 【取值范围】 • logv：输出VERBOSE（调试详情）级别的日志。 • logd：输出DEBUG（调试信息）级别的日志。 • logi：输出INFO（一般信息）级别的日志。 • logw：输出WARN（警告信息）级别的日志。 • loge：输出ERROR（错误信息）级别的日志。 【配置说明】输出的日志是否可以被控制台拉取，取决于`TLogService.updateLogLevel()`函数的参数设置。例如，当`TLogService.updateLogLevel(TLogLevel.WARN)`时，则控制台拉取不到通过logv/logd/logi输出的日志。
MODEL	用于设置输出日志内容的功能模块，便于后续根据来源筛选日志。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】是 【是否大小写敏感】否 【示例】“推送功能模块Push”
TAG	用于设置日志的关键字，便于后续根据标签筛选日志。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】是 【是否大小写敏感】否 【示例】“推送功能模块收到了推送Push.receive，推送功能模块点击了推送Push.click”
MESSAGE	用于输出日志信息。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】是

上报日志

1：如业务流程触发日志输出，需引入头文件：

```
import com.alibaba.ha.adapter.service.tlog.TLogService;
```

2：在适当位置添加代码，主动上报当日日志信息。示例代码：

```
TLogService.positiveUploadTlog("COMMENT");
```

> 说明
>
> - 函数 `TLogService.positiveUploadTlog(<COMMENT>);` ，用于主动上报当日缓存的日志信息。
> - 主动上报时系统会删除历史缓存日志，仅上报最新缓存的日志信息，避免相同日志重复上报。

参数	说明
COMMENT	上报日志时设置的备注信息。 【数据类型】字符串 【字符类型】英文大小写、中文、数字、特殊字符 【是否必选】否

混淆配置

如App对代码进行乱序混淆，则在混淆配置文件中添加代码段：

```
-keep class com.taobao.tao.log.**{*;}
-keep public class * extends com.taobao.android.tlog.protocol.model.request.base.FileInfo{*
;}
-keepattributes Exceptions,InnerClasses,Signature,Deprecated,SourceFile,LineNumberTable,*An
notation*,EnclosingMethod
```

编译

如同时使用其他阿里云产品，可能会因为依赖中存在UTDID冲突，造成编译失败。解决办法参见：SDK UTDID冲突解决方案。

样例代码

远程日志服务Android SDK接入工程样例参见：Demo工程。

2.4. Android SDK接入验证

Android SDK接入过程中如出现问题，可按照编译阶段常见问题排查所列要点，进行排查；Andoird SDK接入之后，可按照运行阶段业务逻辑验证进行接入验证。

前提条件

1：确认使用armeabi-v7a/arm64-v8a/X86/X86_64架构。

2：确认使用Android 4.2及以上版本。

编译阶段常见问题排查

1：本地集成场景，确认已下载使用最新版本的SDK包，且内容完整无缺失。

2：确认已正确添加依赖，且不存在依赖冲突。

3：如SDK包中存在名称相同，版本号不同的文件，确认已添加最高版本的文件。

4：确认已正确配置ABI。

5：确认初始化代码中的配置项已正确配置。

6：如对代码进行乱序混淆，确认已添加混淆配置。

运行阶段业务逻辑验证

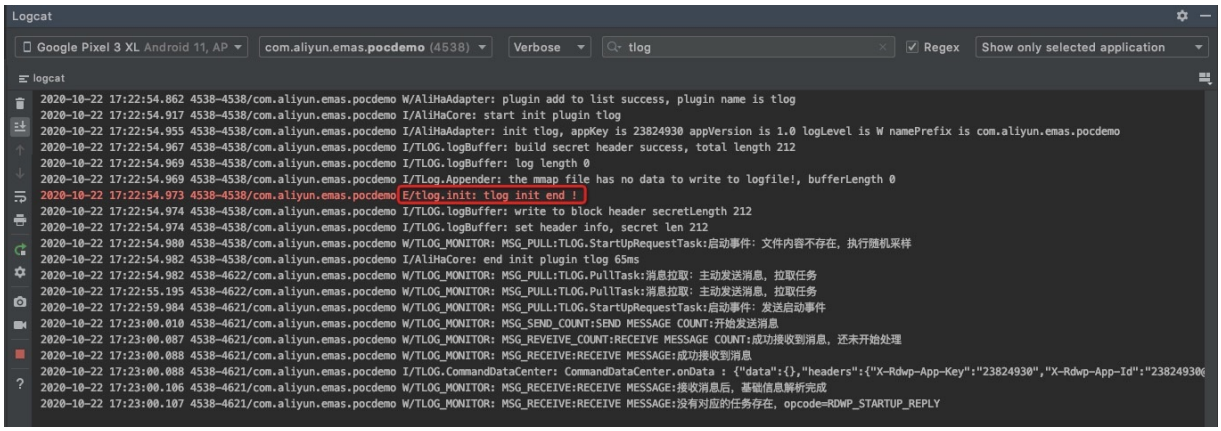
步骤

1：识别设备

【手机端】

1：重新安装并启动App。

2：查看日志，搜索关键字 `tlog` ，如存在 `tlog init end !` ，则初始化成功。



如初始化失败，建议优先检查 `rsaPublicKey` 设置。

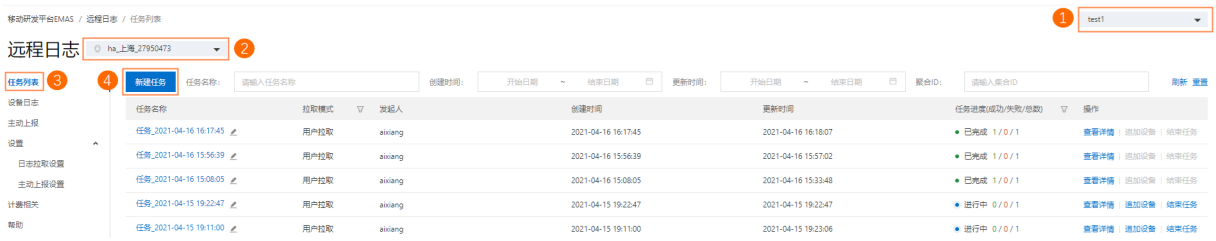
【控制台】确认识别手机端设备。

1：打开远程日志控制台。具体操作参见：打开控制台页面。

2：在下拉列表选择产品和应用。

3：从左侧导航栏，选择**任务列表**页签。

4：在**任务列表**页签，单击**新建任务**按钮，打开**新建任务**页面。



5：在左侧导航栏，分别选择**设备日志**和**主动上报**，输入**设备ID/名称**，单击**刷新**按钮，查看手机端设备是否已被识别。

如未识别，可能的原因是：

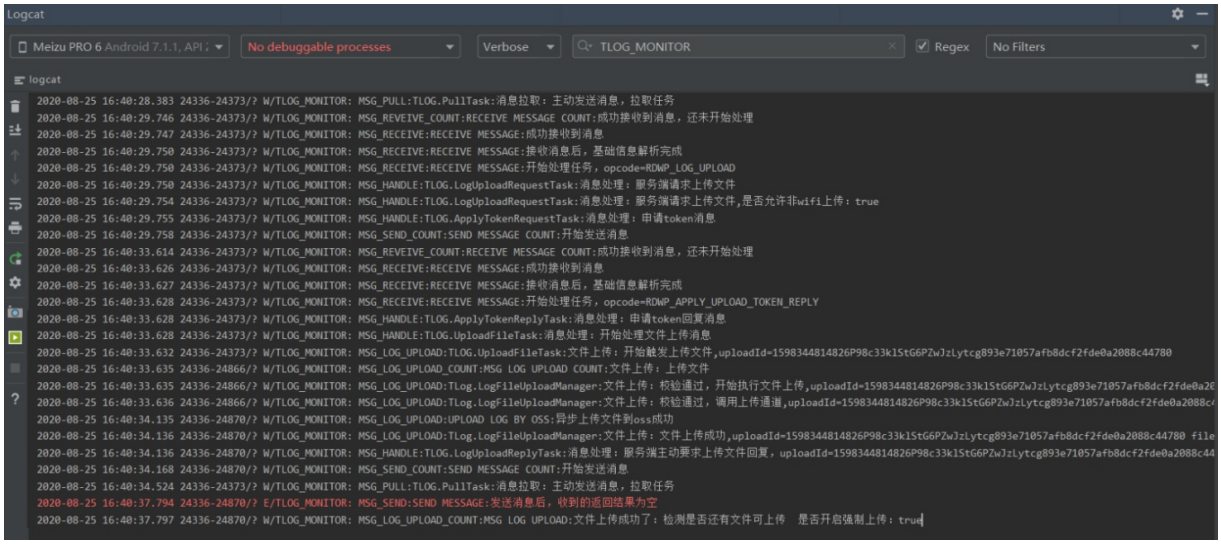
- 网络未连接/延迟：确认手机端已联网，稍作等待，刷新再试。
- SDK接入失败/SDK未获取数据/数据发送失败/后端问题：联系技术支持解决。

步骤2：拉取日志

【手机端】确认日志上传成功。

1：操作App进行前后台切换，触发日志输出。

2：查看客户端日志，搜索关键字 `TLOG_MONITOR` ，查看相关信息，确认日志是否上传成功。



如上传失败，联系技术支持解决。

【控制台】确认正确拉取日志。

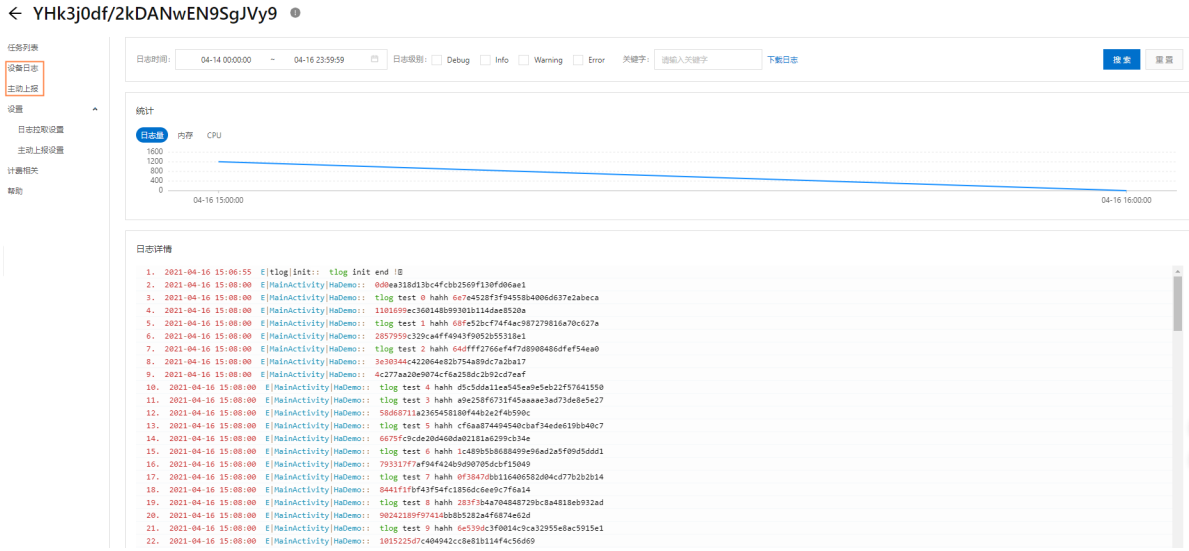
1：创建日志拉取任务，拉取指定手机端设备的日志。具体操作参见：新建任务。

2：查看拉取任务，确认日志拉取任务已完成。具体操作参见：查看/管理任务。

3：在左侧导航栏，分别选择**设备日志**和**主动上报**，确认指定手机端设备的日志已拉取成功。

- **设备日志**的具体操作请参见：查看日志。
- **主动上报**的具体操作请参见：查看主动上报日志。

查看主动上报日志



如预期日志未被成功拉取或成功上报，可能的原因是：SDK接入失败/SDK未获取数据/数据发送失败/后端问题。联系技术支持解决。

3.React-Native

3.1. React Native库接入

接入方式：

```
npm install @emas/emas-react-native-tlog
```

3.2. Native SDK接入

EMAS React-Native页面性能监控底层依赖的仍然是Native（iOS/Android）端的SDK，所以React-Native相关的SDK安装完毕后，还需要在Native（iOS/Android）端接入相应的Native SDK。

准备工作

接入SDK前，请确保在阿里云移动研发平台（EMAS）上创建相关产品，并获得相应的配置文件（包含appKey, appSecret, rsaPublicKey）等关键配置信息。

iOS SDK接入

本文档适用于使用cocoaPods管理依赖的Xcode项目，适用于iOS 8.0及以上版本的App编辑react-native项目iOS目录下的Podfile文件。

1. **指定官方仓库和阿里云仓库。**

   在 `Podfile` 中指定source。

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

2. **添加SDK。**

   ```
   pod 'AlicloudTLog', '~> 1.0.0'
   ```

   添加完SDK后执行如下命令：

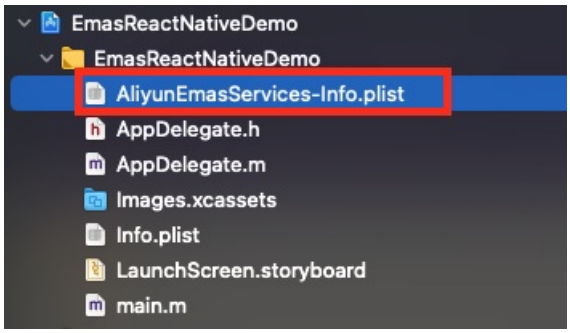
   ```
   pod install
   ```

3. **下载配置文件。**

   在EMAS平台上下载App对应的配置文件- `AliyunEmasServices-Info.plist` 。

   > **说明**
   >
   > 已经完成准备工作，即已经在EMAS平台上创建相关产品。

将配置文件放到iOS的工程目录内， 如图所示：



4. **初始化SDK。**

   打开 `AppDelegate.m` ，对SDK进行初始化：

   i. 引入头文件：

   ```
   #import <AlicloudTLog/AlicloudTlogProvider.h>
   #import <AlicloudHAUtil/AlicloudHAProvider.h>
   ```

   ii. 在didFinishLaunchingWithOptions方法内对SDK进行初始化，示例代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
ictionary *)launchOptions
{
   ...

  //emas初始化
  @try {
    [self emasInit];
  } @catch (NSException *exception) {
    NSLog(@"Emas-->exception is %@", exception);
  } @finally {
  }

  return YES;
}

- (void) emasInit {
  // channel根据实际情况填写
  NSString* channel = @"emas-rn";
  // nick根据app实际情况填写
  NSString* nick = @"emas-rn-demo";
  NSString* appVersion = [self getAppVersion];

  [[AlicloudTlogProvider alloc] autoInitWithAppVersion:appVersion channel:channel n
ick:nick];

  [AlicloudHAProvider start];
}

- (NSString *)getAppVersion
{
    NSDictionary *appinfo = [[NSBundle mainBundle] infoDictionary];
    NSString *version = [appinfo objectForKey:@"CFBundleShortVersionString"];
    if (!version) {
        version = @"1.0.0";
    }
    return version;
}
```

Android SDK接入

1. **添加Maven仓库。**

   在ReactNative的android目录下的build.gradle文件中，添加仓库地址：

```
buildscript {
    ...
    repositories {
        google()
        mavenCentral()
        maven {
            url 'http://maven.aliyun.com/nexus/content/groups/public/'
        }
        maven { url "http://maven.aliyun.com/nexus/content/repositories/releases" }
    }
    dependencies {
        classpath("com.android.tools.build:gradle:4.1.2")
    }
}
```

2. **添加SDK。**

   在android/app目录下的build.gradle文件中添加如下依赖：

```
dependencies {
    ....

    //HA Adapter 高可用
    implementation('com.aliyun.ams:alicloud-android-ha-adapter:1.1.5.1-open')
     //tlog
    implementation('com.aliyun.ams:alicloud-android-tlog:1.1.4.1-open')

}
```

3. **初始化SDK。**

   打开MainApplication（一般的ReactNative都是使用自动创建的MainApplication，如果接入的App有自定义的Application，就在自定义的Application里面添加初始化代码），初始化代码一般放在onCreate方法中，示例代码：

```
import com.alibaba.ha.adapter.AliHaAdapter;
import com.alibaba.ha.adapter.AliHaConfig;
import com.alibaba.ha.adapter.Plugin;

...
@Override
public void onCreate() {
 super.onCreate();
 SoLoader.init(this, /* native exopackage */ false);
 initializeFlipper(this, getReactNativeHost().getReactInstanceManager());

 emasInit();
}

private void emasInit() {

  AliHaConfig config = new AliHaConfig();
  //设为自己的appKey
  config.appKey = "xxx";
  config.appVersion = BuildConfig.VERSION_NAME;
  //设为自己的appSecret
  config.appSecret = "xxxxx";
  //根据实际情况设置
  config.channel = "xxxx";
  //根据实际情况设置
  config.userNick = "xxxxx";
  config.application = this;
  config.context = this.getApplicationContext();
  config.isAliyunos = false;
  /* tlog公钥，在控制台下载aliyun-emas-services.json文件，文件内的appmonitor.tlog.rsaSecret
字段即为公钥信息 */
  config.rsaPublicKey = HA_RSA_PUBLIC_KEY;
  config.initAsync = false;

  AliHaAdapter adapter = AliHaAdapter.getInstance();
  adapter.addPlugin(Plugin.tlog);
  adapter.openDebug(true);
  //是否使用http，根据实际情况设置，默认是false即使用https
  adapter.openHttp(true);

  adapter.start(config);
 }
```

3.3. API说明

- error日志的函数原型为：

```
**
* error日志
* @param module module名称
* @param tag tag名称
* @param content 日志内容
*/
export declare function error(module: string, tag: string, content: string): void;
```

- warn日志的函数原型为：

```
/**
* warn日志
* @param module module名称
* @param tag tag名称
* @param content 日志内容
*/
export declare function warn(module: string, tag: string, content: string): void;
```

- debug日志的函数原型为：

```
/**
 * debug日志
* @param module module名称
* @param tag tag名称
* @param content 日志内容
*/
export declare function debug(module: string, tag: string, content: string): void;
```

- info日志的函数原型为：

```
/**
 * info日志
 * @param module module名称
 * @param tag tag名称
 * @param content 日志内容
 */
export declare function info(module: string, tag: string, content: string): void;
```

示例代码：

```
import * as EmasLog from '@emas/emas-react-native-tlog';
//或者用下面这种形式，也可以只导入某个具体的方法
import {error, warn, debug, info} from  '@emas/emas-react-native-tlog';

EmasLog.error('TLog', 'Emas', 'Emas-->info log');
EmasLog.info('TLog', 'Emas', 'Emas-->info log');
EmasLog.warn('TLog', 'Emas', 'Emas-->warn log');
EmasLog.debug('TLog', 'Emas', 'Emas-->debug log');
```