蚂蚁金服金融科技产品手册

移动同步

产品版本： V20200101

文档版本： V20200101

蚂蚁金服金融科技文档

商标声明

及其他蚂蚁金服金融科技服务相关的商标均为蚂蚁金服金融科技所有。
本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

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。蚂蚁金服金融科技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，并在蚂蚁金服金融科技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蚂蚁金服金融科技授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

目录

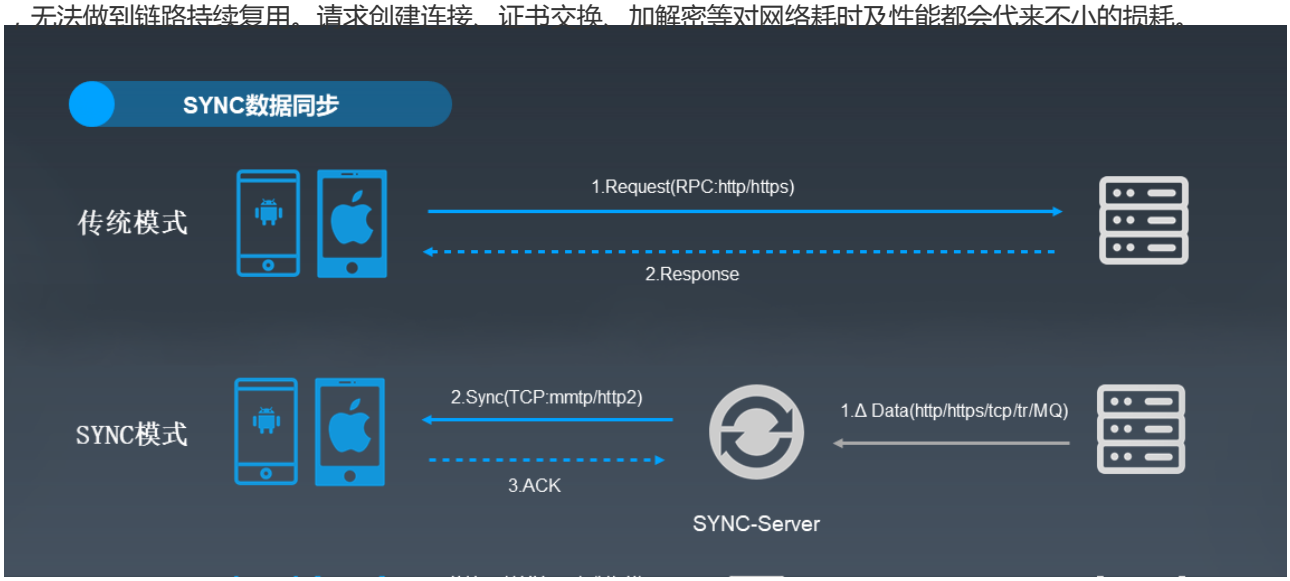
1 移动同步简介

移动同步服务（Mobile Sync Service，简称MSS）是 mPaaS 平台的一个核心基础服务组件。MSS 源自蚂蚁集团内面向移动应用、从服务端到客户端进行海量数据推送的全链路解决方案——SYNC。该组件提供了一个安全的基于传输控制协议（Transmission Control Protocol，简称 TCP）和安全套接层（Secure Sockets Layer，简称 SSL）的数据通道，能够及时、准确、有序地将服务器端的业务数据主动地同步（SYNC）到客户端 APP。

MSS 的主要特性及其能够解决的问题

传统的远程过程调用（Remote Procedure Call，简称 RPC）已立足互联网行业几十年，也能满足绝大部分业务场景和功能需求。但在现阶段，随着移动互联网的全面普及和发展，无论是 APP 的规模还是用户对于 APP 的要求都已进入了一个新的阶段。传统的 RPC 请求因其自身的特性，存在许多的不足：

- 客户端在特定的场景下需要调用 RPC 请求来获取最新的数据，而服务端（云端）实际没有或仅有少量数据发生变化。
- 在客户端启动时，不同的业务模块、业务功能因设计上的独立，需要分别进行 RPC 请求来完成各自业务的数据拉取。
- 客户端无法及时感知服务端发生的数据变化，只能通过定时轮询 RPC 接口的方式来刷新数据。
- 传统 RPC 大多基于 http(s) 的短连接进行数据交互，连接上即使使用 keepalive 等特性也无法长期保持连接，无法做到链路持续复用。请求创建连接、证书交换、加解密等对网络耗时及性能都会代来不小的损耗。



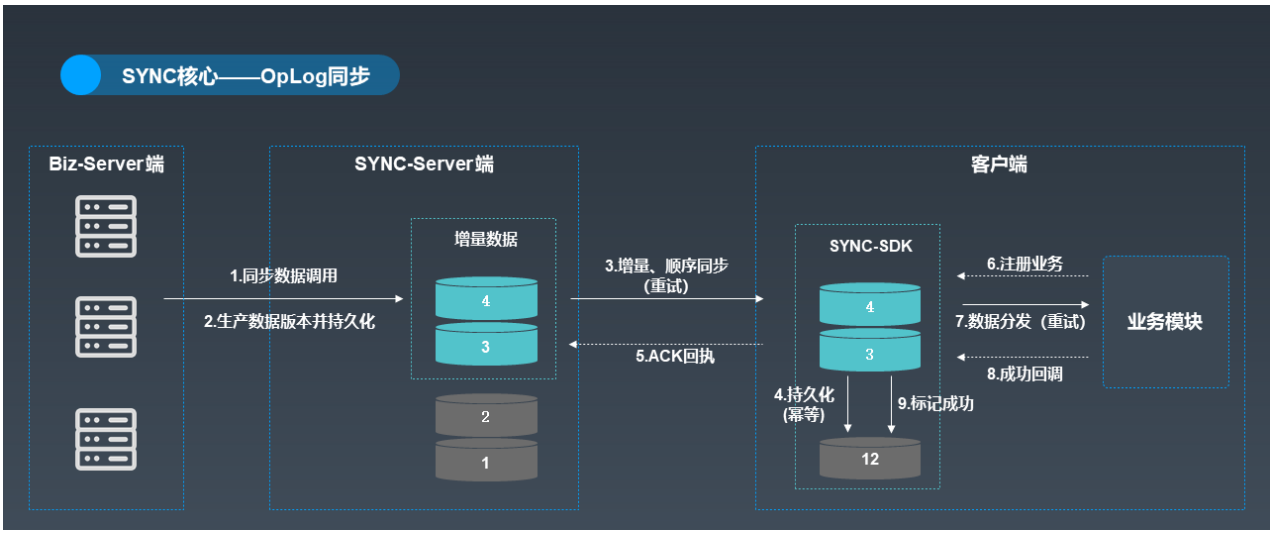
为了改善或解决这些问题，MSS 应运而生，其核心特性有：

- 可靠同步
  针对业务要求的 QoS（Quality of Service）等级为必达的业务场景而言，MSS 保证只要用户在该数据有效期内活跃并且匹配业务推送要求的条件(如客户端版本号、操作系统类型等维度)，就一定让客户端同步到业务推送的数据。
- 增量有序
  MSS 保证同一个通道内到达客户端的消息顺序一定是与业务服务器调用 MSS 服务器的顺序一致，并且所有消息以增量方式同步至客户端。
- 高实时性
  当客户端连接的网络状况良好时，MSS 可以保证非常高的推送实时性，消息推送耗时几乎是纯网络传输的耗时（1s 之内送达）。当客户端连接的网络受到主干网波动、路由器故障、基站信号弱、客户端存储满等不可抗拒

因素干扰时，MSS推送则会待TCP长连接重新建上以后再进行数据同步。

基本原理

类似于 MySql 的 binlog 机制，MSS 服务器和客户端 SDK 之间传递的基本数据单元为oplog，当业务需要同步一个变更数据到指定的用户或设备时，业务调用 MSS 接口，MSS 服务端会将业务需要同步的数据变更包装为一个 oplog 持久化到数据库，然后在客户端在线的时候把oplog推送给客户端。每个 oplog 拥有一个唯一的 oplog id，oplog id 在确定的用户、确定的业务范围内保证唯一并且单调递增（按调用顺序）。MSS 服务端会按照 oplog id 从小到大的顺序把每一条 oplog 都推送至客户端。MSS 服务端和客户端都会记录客户端已经收到的最大 oplog id，称为同步点（亦可以被理解成数据版本号）。



价值优势

- 合并推送
  客户端初始化成功时，服务端可一次性推送多个业务数据，减少不同业务的请求。
- 增量推送
  只在有增量数据时才推送业务数据，可有效减少冗余数据的传输，降低网络成本。
- 减少请求
  在没有增量数据时，不会消耗请求成本，减少业务的冗余请求。
- 提高时效
  当服务端发生数据变化时，可在最短时间内将变化数据直接推送至客户端，无需等待客户端请求。
- 提升体验
  数据无感知推送，在渲染客户端界面之前，数据已到位，降低了用户等待时间。

使用场景

MSS 可应用于客户端内需要实时推送数据的业务场景，如转账结果推送、支付结果推送、消息中心等。您可以通过以下场景对 MSS 的能力有更深入了解。

- 在即时通讯 APP 中，MSS 提供增量、可靠的消息触达能力，将聊天消息按发送方的发送顺序，有序推送至指定用户。
- 在需要动态更新配置的 APP 中，MSS 可以动态地将配置信息进行全设备推送。将 APP 功能开关、

动态参数、动态配置等信息实时推送至指定客户端，或者批量动态地改变 APP 在运行期间的业务参数、业务配置。

- 在支付类 APP 中，MSS能够为交易数据的在线推送提供安全数据通道，保证在线 APP 可实时接收推送数据。同时 MSS 还能够提供数据持久化能力，使 APP 在下一次上线时收到不在线期间的推送数据。

2 基础术语

BizType
为业务类型，是业务场景的唯标识。数据推送后，客户端 MSS SDK 需要通过 Biztype 将数据分发给对应的业务模块。

持久化
持久化是将程序数据在持久状态和瞬时状态间转换的机制。在 MSS 中，该机制产生了两种行为：持久化数据和非持久化数据。

- 持久化数据：当户或设备不在线时，数据将持久化数据库，待户或设备上线后，MSS SDK 将触发同步。
- 非持久化数据：当用户或设备在线时，立即推送数据；不在线则直接抛弃数据，即便用户再次上线也无法再接收到该条数据。

单设备推送
指基于用户维度推送时，消息推送至用户最新上线的设备，且只推送一次。在设备上卸载客户端，重新安装并上线或者用户在其他设备上线时，系统将不重复推送该条数据。

多设备同步
指基于用户维度推送时，支持单个用户的多个设备之间的数据同步，即同一个用户在切换设备的情况下仍然会收到在上一个设备上已经收到过的数据。在设备上卸载客户端，重新安装并上线后，数据依然会再次推送。

后台
指客户端 APP 当前处于压后台状态（用户手机在 home 界面、在操作其他 App 或处于黑屏状态等）。

幂等
根据 SyncOrder 中的 thirdMsgId 字段进去重复（bizType、linkToken、thirdMsgId 组合唯一即可），只允许成功一次，新的数据会被抛弃不予入库，接口返回成功，结果码为 DUPLICAT ED_BIZ _ID。

MSS 数据
指需要通过 MSS 服务端推送的数据。

MSS 推送
指将份数据从服务端主动推送到客户端，若调用业务的客户端在线，则立即触发推送，否则，待客户端上线之后再进行推送。

前台
指客户端 App 当前处于打开的状态。

SYNC
指 MSS 数据同步服务，是指数据从 MSS 服务端同步至客户端 App。

推送类型
分为指定推送和全局推送两种类型。

- 指定推送：指定某个 userId 或 utdId，推送条数据。
- 全局推送：对所有已上线的用户或设备推送数据，全局推送业务为多设备同步。

业务维度
同步的业务维度分为户维度和设备维度，用户维度指根据 userId 来推送数据，设备维度指根据 utdId 来推送数据。

阈值
为服务端允许积压的数据上限。推送数据时，若用户或设备长时间不在线，而 MSS 服务端又一直产生新的数据，则可能导致服务端数据积压，此时，将只保留阈值内的最新数据，超出阈值部分的老数据将被废弃。

在线
指客户端 App 有网络，可保持稳定的 TCP 长连接。
大部分 Android 手机支持 App 在后台时保持在线，苹果手机支持 App 在后台时维持 3 分钟在线（iOS 操作系统性限制）。

3 接入 MSS 方式介绍

为使用 MSS 的功能，您需要将MSS加入您的开发工程，并将开发工程接入mPaaS。整个的接入过程可以分为以下三个环节：

在接入客户端环节，您需要完成基础配置、添加 MSS SDK，并更新您的业务代码。
根据开发环境和开发目的，接入客户端环节存在以下四种情况，请您根据实际情况进行选择性参考。

- 基于mPaaS框架接入 Android 工程到 mPaaS
- 基于原生框架接入 Android 工程到 mPaaS
- 基于mPaaS框架接入 iOS 工程到 mPaaS
- 基于原生框架接入 iOS 工程到 mPaaS

说明：“基于 mPaaS 框架”即使用 mPaaS 插件创建全新工程；“基于原生框架”即使用mPaaS插件对现有工程进行改造。
在接入服务端环节，您需要根据您的实际生产环境，将业务系统接入 MSS 服务器端。
最后，您就可以在控制台，使用 MSS 的功能进行数据同步。

4 接入客户端到 mPaaS

4.1 基于 mPaaS 框架接入 Android 工程到 mPaaS

关于此任务

本文将指导您基于 mPaaS 框架接入 Android 工程到 mPaaS。为完成此任务，您需要执行以完成基础配置和添加 SDK，最终使用 SDK 进行 Android 应用开发。

前置条件

您已经在 **移动开发平台** > **后台服务管理** > **数据同步** > **配置管理** 创建了同步配置，并且获得了同步标识。

操作步骤

1. 完成基础配置。基础配置包括以下三个方面：
    - 配置开发环境。更多信息，请参考 准备配置 。
    - 在控制台创建应用 。
    - 在客户端基于mPaaS框架创建新工程。更多信息，请参考 客户端创建新工程 。
2. 添加 SDK。
   分别在创建的 Portal 和 Bundle 工程中添加 **SYNC 同步服务** 组件依赖。更多信息，请参考 增删组件依赖 。
3. 使用 SDK。
   根据不同的 SDK 版本，您将需要采取不同的方法来使用 SDK。请根据您使用的 SDK 版本查看对应的 SDK 使用文档。
    - 使用SDK（版本 ≥ 10.1.32）
    - 使用SDK（版本 < 10.1.32）

说明：您可以通过查看 Android Studio 中 **mPaaS** > **Baseline Update** 下的 **Current mPaaS SDK Version** 信息来确认您使用的 SDK 版本。

后续操作

使用 SDK（版本 ≥ 10.1.32）

在 10.1.32 及以上版本基线中，mPaaS 中间层的 MPSync 类封装了移动同步组件所有 API，您可以通过下表对这些 API 进行快速了解。更多关于 API 接口的详细信息，请参考 版本 ≥ 10.1.32 。

接口	接口说明
initialize(Context context)	用于初始化接口和移动同步服务。
appToForeground()	用于让客户端 SDK 感知到当前 App 已经启动，使其建立与服务器的网络连接。每次 App 回前台时调用。
appToBackground()	用于让客户端 SDK 感知到当前 App 已经回到后台，使其断开与服务器的网络连接。每次 App 压后台时调用。
updateUserInfo(String sessionId)	用于登录信息 userId/sessionId 有变化时调用。
clearUserInfo()	用于用户登出。
registerBiz(String bizType, ISyncCallback syncCallback)	用于注册一个接收业务数据的 callback。在获取到同步推送的数据后，客户端 SDK 会回调 syncCallback 实现类。
unregisterBiz(String bizType)	用于反注册指定同步配置。在获取到同步推送的数据后，客户端 SDK 则不会回调 syncCallback 实现类。
reportMsgReceived(SyncMessage syncMessag)	用于在 syncCallback 实现类中收到数据后，调用该接口通知移动同步服务端接收同步数据成功。在没有收到 reportMsgReceived 前，移动同步服务会重试投递，重试 6 次之后数据会被永久删除。

isConnected()	用于检查当前移动同步服务是否正常。

代码示例

该示例通过在 10.1.32 基线版本 SDK 进行开发。在示例应用中设置了一个按钮，通过点按按钮动作获取设备 ID，再根据设备 ID，在控制台采用指定设备推送的方式向设备推送同步数据。在示例中，同步标识为bizType。

说明：该示例仅用于演示调用移动同步 API 的方法，并不能作为移动同步的最佳实践。您可以在 获取代码示例 页面获取移动同步组件的最佳实践代码。

```
public void button1Clicked(View view)
{
//使用 getUtdid 方法获取设备ID。
String utdid =UTDevice.getUtdid(MainActivity.this);
//在 Logcat 打印移动同步数据。
Log.e("==========",utdid);
//初始化接口和移动同步服务。
MPSync.initialize(MainActivity.this);
//注册接收业务数据的 callback。在获取到同步推送的数据后，回调 syncCallback。
MPSync.registerBiz("bizType",new SyncCallBackImpl());
//建立与服务器的网络连接。
MPSync.appToForeground();

}

public class SyncCallBackImpl implements ISyncCallback
{
@Override
public void onReceiveMessage(SyncMessage syncMessage) {
//在 Logcat 打印移动同步数据。
Log.e("==========",syncMessage.msgData);
//通知移动同步服务端接收同步数据成功。
MPSync.reportMsgReceived(syncMessage);
}
}
```

使用SDK（版本 < 10.1.32）

在 10.1.32 以下版本基线中，mPaaS 中间层的 类 LongLinkSyncService 提供了移动同步组件所有 API，您可以通过下表对这些 API 进行快速了解。更多关于 API 接口的详细信息，请参考 版本 < 10.1.32 。

接口	接口说明
initialize(Context context)	用于初始化接口和移动同步服务。
appToForeground()	用于让客户端 SDK 感知到当前 App 已经启动，使其建立与服务器的网络连接。每次 App 回前台时调用。
appToBackground()	用于让客户端 SDK 感知到当前 App 已经回到后台，使其断开与服务器的网络连接。每次 App 压后台时调用。
updateUserInfo(String sessionId)	用于登录信息 userId/sessionId 有变化时调用。
registerBiz(String bizType, ISyncCallback syncCallback)	用于注册一个接收业务数据的 callback。在获取到同步推送的数据后，客户端 SDK 会回调 syncCallback 实现类。
unregisterBiz(String	用于反注册指定同步配置。在获取到同步推送的数据后，客户端 SDK 则不会回调 syncCallback

bizType)	实现类。
reportMsgReceived(SyncMessage syncMessag)	用于在 syncCallback 实现类中收到数据后，调用该接口通知移动同步服务端接收同步数据成功。在没有收到 reportMsgReceived 前，移动同步服务会重试投递，重试 6 次之后数据会被永久删除。
isConnected()	用于检查当前移动同步服务是否正常。

代码示例

该示例通过在 10.1.32 基线版本 SDK 进行开发。在示例应用中设置了一个按钮，通过点按按钮动作获取设备 ID，再根据设备 ID，在控制台采用指定设备推送的方式向设备推送同步数据。在示例中，同步标识为bizType。

说明：该示例仅用于演示调用移动同步 API 的方法，并不能作为移动同步的最佳实践。您可以在 获取代码示例 页面获取移动同步组件的最佳实践代码。

```
public void button1Clicked(View view)
{
//使用 getUtdid 方法获取设备ID。
String utdid = UTDevice.getUtdid(MainActivity.this);
//在 Logcat 打印移动同步数据。
Log.e("==========",utdid);

//初始化接口和移动同步服务。
LongLinkSyncService.getInstance().initialize(MainActivity.this);
//注册接收业务数据的 callback。在获取到同步推送的数据后，回调 syncCallback。
LongLinkSyncService.getInstance().registerBiz("bizType",new SyncCallBackImpl());
//建立与服务器的网络连接。
LongLinkSyncService.getInstance().appToForeground();

//MPSync.initialize(MainActivity.this);
//MPSync.registerBiz("bizType1",new SyncCallBackImpl());
//MPSync.appToForeground();

}

public class SyncCallBackImpl implements ISyncCallback
{
@Override
public void onReceiveMessage(SyncMessage syncMessage) {
//在 Logcat 打印移动同步数据。
Log.e("==========",syncMessage.msgData);
//通知移动同步服务端接收同步数据成功。
LongLinkSyncService.getInstance().reportMsgReceived(syncMessage);
}
}
```

4.2 基于原生框架接入 Android 工程到 mPaaS

4.2.1 基础配置

为接入 Android 工程到 mPaaS，您需要执行以下步骤以完成基础配置。

1. 在控制台创建应用并下载配置文件 。
2. 通用基础配置 。

4.2.2 添加SDK

要将 MSS 接入 Android 客户端，需要添加 MSS SDK 并对工程进行配置。本文将引导您在基于 mPaaS 框架的接入方式下，为 Android 工程添加 SDK。

前置条件

- 您已参考 原生工程通用步骤说明 完成基础配置。
- 您已经在 **移动开发平台** > **后台服务管理** > **数据同步** > **配置管理** 创建了同步配置，并且获得了同步标识。

操作步骤

1. 修改工程的 build.gradle 配置文件，在 dependencies 下新增配置:

```
dependencies {
compile 'com.alipay.android.phone.sync:syncservice-build:2.0.0.170630165953@aar'
compile"com.alibaba:fastjson:1.2.23"
compile 'com.alipay.android.phone.thirdparty:securityguard:6.1.92.161020170915' compile
'com.alipay.android.phone.thirdparty:wire:1.5.3.160824110117' compile
'com.alipay.android.phone.thirdparty:utdid4all:1.1.5.3'
compile 'com.alipay.android.phone.mobilesdk:logging:2.0.0'
compile 'com.alipay.android.phone.mobilesdk:storage-build:1.7.0.161115203116@aar'
}
```

2. 编译项目。

后续操作

根据不同的 SDK版本，您将需要采取不同的后续操作以使用SDK。请根据您使用的 SDK 版本查看对应的 SDK 使用文档。

- 使用 SDK（版本 ≥ 10.1.32）
- 使用 SDK（版本 < 10.1.32）

4.2.3 使用SDK（版本 ≥ 10.1.32）

前置条件

您已经确认到您使用的 SDK 版本 ≥ 10.1.32。如果您使用的 SDK 版本 < 10.1.32，请参考 使用 SDK（版本 < 10.1.32）。

修改项目配置

在 Portal 工程的 ..\app\src\main\AndroidManifest.xml 文件中，确认已经包含了以下配置信息。

```
<!-- 移动同步服务端地址和端口 -->
```

```
<meta-data
android:name="syncport"
android:value="443"/>
<meta-data
android:name="syncserver"
android:value="cn-hangzhou-mss-link.cloud.alipay.com"/>
<!-- work space id -->
<meta-data
android:name="workspaceId"
android:value="default"/>
<!--无限保镖 appkey, used for encrypt-->
<meta-data
android:name="appkey"
android:value="5E5FD99271812_ANDROID"/>
```

在控制台创建应用时，生成了一个配置文件，该配置文件的文件名以“Ant-mpaas-”开头，接着为在服务端创建应用时自动生成的 App ID 和 workspace ID，最后以平台（Android）结尾，类似于 Ant-mpaas-5E5FD99271812-demo-Android.config。如果在创建 Portal工程的时候上传了该配置文件，那么 AndroidManifest.xml 应该已经包含以上配置信息。如果 AndroidManifest.xml 没有包含以上配置信息，则需要参考配置文件添加。

在 Portal 工程的 ..\app\src\main\assets\mpaas.properties确认已经包含了以下配置信息。

```
AppId=5E5FD99271812
Platform=ANDROID
WorkspaceId=default
```

以上各取值与上一步 AndroidManifest.xml 中信息的获取方式一样，也是从服务端生成的配置文件中获取。如果在创建 Portal工程的时候上传了该配置文件，那么 mpaas.properties 应该已经包含以上配置信息；如果 mpaas.properties 没有包含上述信息，则需要参考配置文件添加。

编写代码

mPaaS 中间层的 MPSync 类封装了移动同步组件所有 API，请参考 版本 ≥ 10.1.32 查看API接口信息

示例

以下为注册同步配置、获取数据的示例。在接入代码之前需要配置好具体的同步配置，获取到对应的同步标识。以下示例中的同步标识为 UCHAT。

```
String bizType ="UCHAT";
//实现一个用于接收数据的 callback
//请 import com.alipay.mobile.rome.longlinkservice.ISyncCallback private ISyncCallback syncCallback = new
ISyncCallback() {
@Override
public void onReceiveMessage(final SyncMessage syncMessage) {//在独立的线程中完成，否则可能堵塞后面的操作
Runnable runnable = new Runnable(){
@Override
public void run() {
//做下基本的数据准确性校验，最终的数据是以 JSONArray 放在 msgData 字段中
Log.d("sync_Activity","onReceiveMessage1:"+syncMessage.msgData+"/"+syncMessage.userId+"/"+syncMessage.biz);
if(!TextUtils.isEmpty(syncMessage.msgData) ){ try {
```

```
JSONArray jsonArray = new JSONArray(syncMessage.msgData); int count = jsonArray.length(); final StringBuffer sb
= new StringBuffer(); for(int index = 0; index < count;index++){
JSONObject jsonObject = (JSONObject) jsonArray.get(index);
//最终需要的数据就在下面字段 finalData 中。
String finalData = jsonObject.optString("pl");
//对于isB标志,需要base64解码
if ("1".equals(jsonObject.optString("isB"))) {
finalData = new String(Base64.decode(finalData, Base64.DEFAULT),"UTF-8");
}
//解析 finalData， 获取数据
sb.append(index+":"+finalData+"|");
}
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(SYNCServiceActivity.this,sb,Toast.LENGTH_LONG).show();
}
});
//通知数据已经接收，如果不通知，我们将会重试，达到 6 次之后，数据将被删除。
MPSync.reportMsgReceived(syncMessage);
} catch (JSONException e) {
Log.d("sync_Activity","onReceiveMessage
JSONException"+syncMessage.msgData+"/"+syncMessage.userId+"/"+syncMessage.biz); } catch (Exception e) {
Log.d("sync_Activity","onReceiveMessage :"+ e);
}
}
}
};
Thread thread = new Thread(runnable); thread.start();
}
};
//注册服务对应的 callback，我们的服务在接收到服务端推送的数据之后就通过 callback 把数据传过来
MPSync.registerBiz(bizType,syncCallback);
```

4.2.4 使用SDK（版本 < 10.1.32）

前置条件

您已经确认到您使用的 SDK 版本 < 10.1.32。如果您使用的 SDK 版本 ≥ 10.1.32，请参考 使用 SDK（版本 ≥ 10.1.32） 。

修改项目配置

在 Portal 工程的 ..\app\src\main\AndroidManifest.xml 文件中，确认已经包含了以下配置信息。

```
<!-- 移动同步服务端地址和端口 -->
<meta-data
android:name="syncport"
android:value="443"/>
<meta-data
android:name="syncserver"
android:value="cn-hangzhou-mss-link.cloud.alipay.com"/>
<!-- work space id -->
```

```
<meta-data
android:name="workspaceId"
android:value="default"/>
<!--无限保镖 appkey, used for encrypt-->
<meta-data
android:name="appkey"
android:value="5E5FD99271812_ANDROID"/>
```

在控制台创建应用时，生成了一个配置文件，该配置文件的文件名以“Ant-mpaas-”开头，接着为在服务端创建应用时自动生成的 App ID 和 workspace ID，最后以平台（Android）结尾，类似于 Ant-mpaas-5E5FD99271812-demo-Android.config。如果在创建 Portal工程的时候上传了该配置文件，那么 AndroidManifest.xml 应该已经包含以上配置信息。如果 AndroidManifest.xml 没有包含以上配置信息，则需要参考配置文件添加。

在 Portal 工程的 ..\app\src\main\assets\mpaas.properties确认已经包含了以下配置信息。

```
AppId=5E5FD99271812
Platform=ANDROID
WorkspaceId=default
```

以上各取值与上一步 AndroidManifest.xml 中信息的获取方式一样，也是从服务端生成的配置文件中获取。如果在创建 Portal工程的时候上传了该配置文件，那么 mpaas.properties 应该已经包含以上配置信息；如果 mpaas.properties 没有包含上述信息，则需要参考配置文件添加。

编写代码

LongLinkSyncService 提供了移动同步组件所有 API，通过 LongLinkSyncService.getInstance() 获取到 LongLinkSyncService 的实例。请参考 版本 < 10.1.32 查看API接口信息。

示例

以下为注册同步配置、获取数据的示例。在接入代码之前需要配置好具体的同步配置，获取到对应的同步标识。以下示例中的同步标识为 UCHAT。

```
String bizType ="UCHAT";
//实现一个用于接收数据的 callback
//请 import com.alipay.mobile.rome.longlinkservice.ISyncCallback private ISyncCallback syncCallback = new
ISyncCallback() {
@Override
public void onReceiveMessage(final SyncMessage syncMessage) {//在独立的线程中完成，否则可能堵塞后面的操作
Runnable runnable = new Runnable(){
@Override
public void run() {
//做下基本的数据准确性校验，最终的数据是以 JSONArray 放在 msgData 字段中
Log.d("sync_Activity","onReceiveMessage1:"+syncMessage.msgData+"/"+syncMessage.userId+"/"+syncMessage.bi
z);
if(!TextUtils.isEmpty(syncMessage.msgData) ){ try {
JSONArray jsonArray = new JSONArray(syncMessage.msgData); int count = jsonArray.length(); final StringBuffer sb
= new StringBuffer(); for(int index = 0; index < count;index++){
JSONObject jsonObject = (JSONObject) jsonArray.get(index);
//最终需要的数据就在下面字段 finalData 中。
String finalData = jsonObject.optString("pl");
//对于isB标志,需要base64解码
```

```
if ("1".equals(jsonObject.optString("isB"))) {
finalData = new String(Base64.decode(finalData, Base64.DEFAULT),"UTF-8");
}
//解析 finalData， 获取数据
sb.append(index+":"+finalData+"|");
}
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(SYNCServiceActivity.this,sb,Toast.LENGTH_LONG).show();
}
});
//通知数据已经接收，如果不通知，我们将会重试，达到 6 次之后，数据将被删除。
LongLinkSyncService.getInstance().reportMsgReceived(syncMessage);
} catch (JSONException e) {
Log.d("sync_Activity","onReceiveMessage
JSONException"+syncMessage.msgData+"/"+syncMessage.userId+"/"+syncMessage.biz); } catch (Exception e) {
Log.d("sync_Activity","onReceiveMessage :"+ e);
}
}
}
};
Thread thread = new Thread(runnable); thread.start();}
};
//注册服务对应的 callback，我们的服务在接收到服务端推送的数据之后就通过 callback 把数据传过来
LongLinkSyncService.getInstance().registerBiz(bizType,syncCallback);
```

4.3 基于 mPaaS 框架接入 iOS 工程到 mPaaS

4.3.1 基础配置

为接入 iOS 工程到 mPaaS，您需要执行以下步骤以完成基础配置。

1. 安装开发者工具 。
2. 在控制台创建应用 。
3. 创建基于 mPaaS 框架的新工程 。

4.3.2 添加 SDK

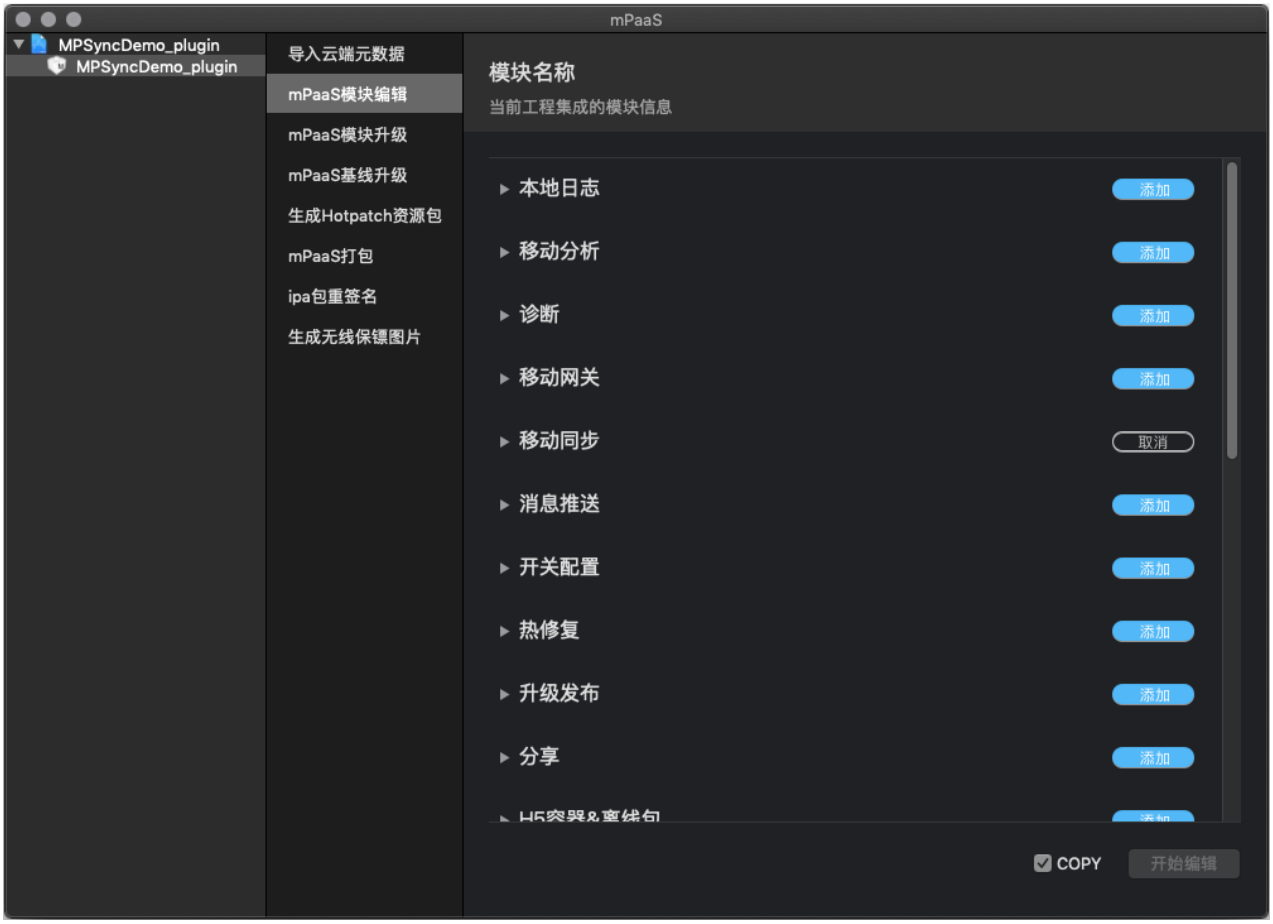
要将 MSS 接入 iOS 客户端，需要添加 MSS SDK 并对工程进行配置。本文将引导您在基于 mPaaS 框架的接入方式下，为 iOS 工程添加 SDK。

前置条件

您已完成基础配置，基于mPaaS框架创建了新的工程。

添加 SDK

如下图所示，使用 mPaaS 插件添加 Sync SDK。更多信息，请参考 mPaaS 插件 > 编辑模块 。



后续步骤

根据不同的 SDK版本，您将需要采取不同的后续操作以使用SDK。请根据您使用的 SDK 版本查看对应的 SDK 使用文档。

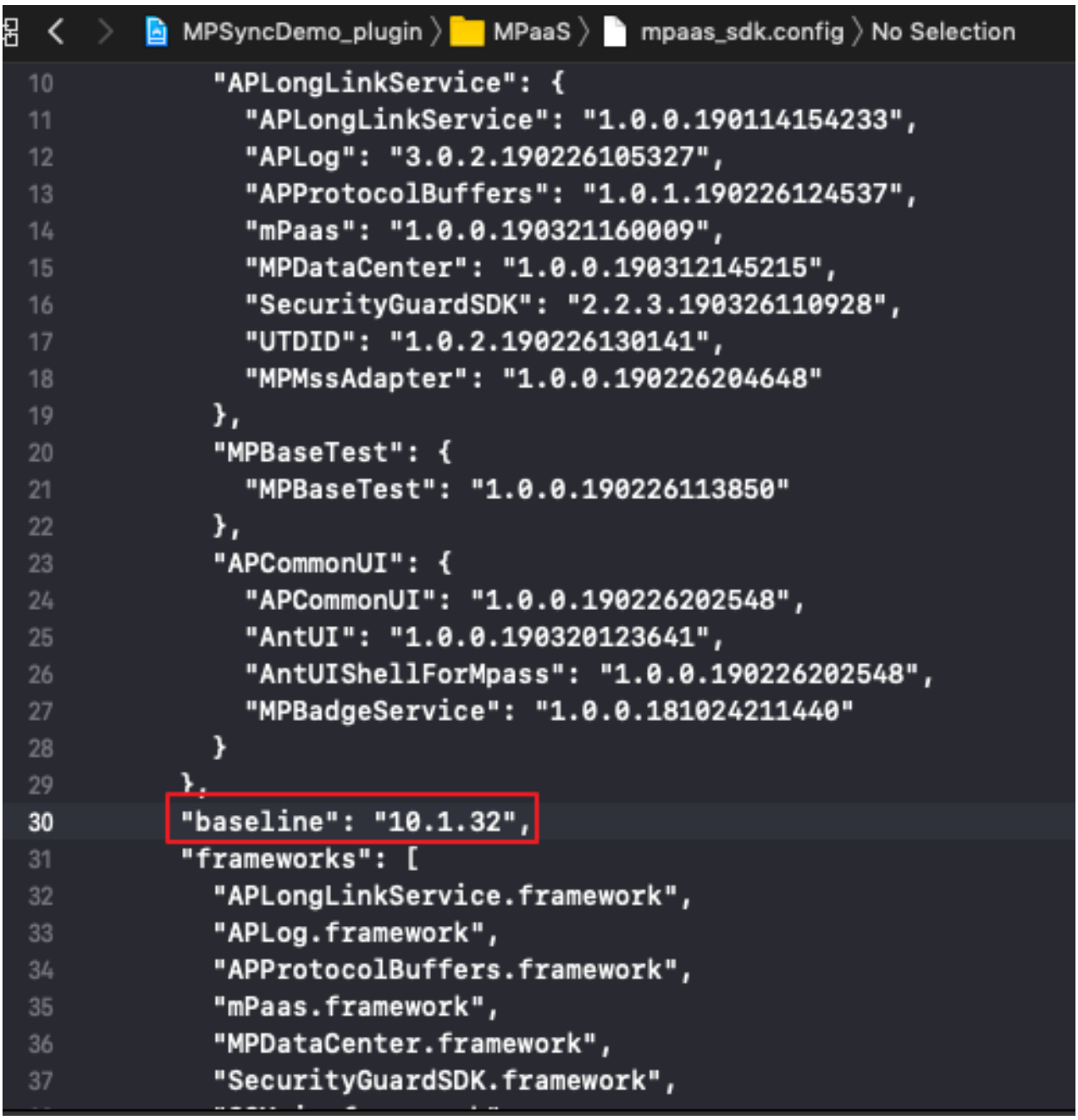
- 使用SDK（版本 ≥ 10.1.32）
- 使用SDK（版本 < 10.1.32）

4.3.3 使用 SDK（版本 ≥ 10.1.32）

前置条件

您已确认所使用的 SDK 版本 ≥ 10.1.32。如果您使用的 SDK 版本 < 10.1.32，请参考 使用 SDK（版本 < 10.1.32）。

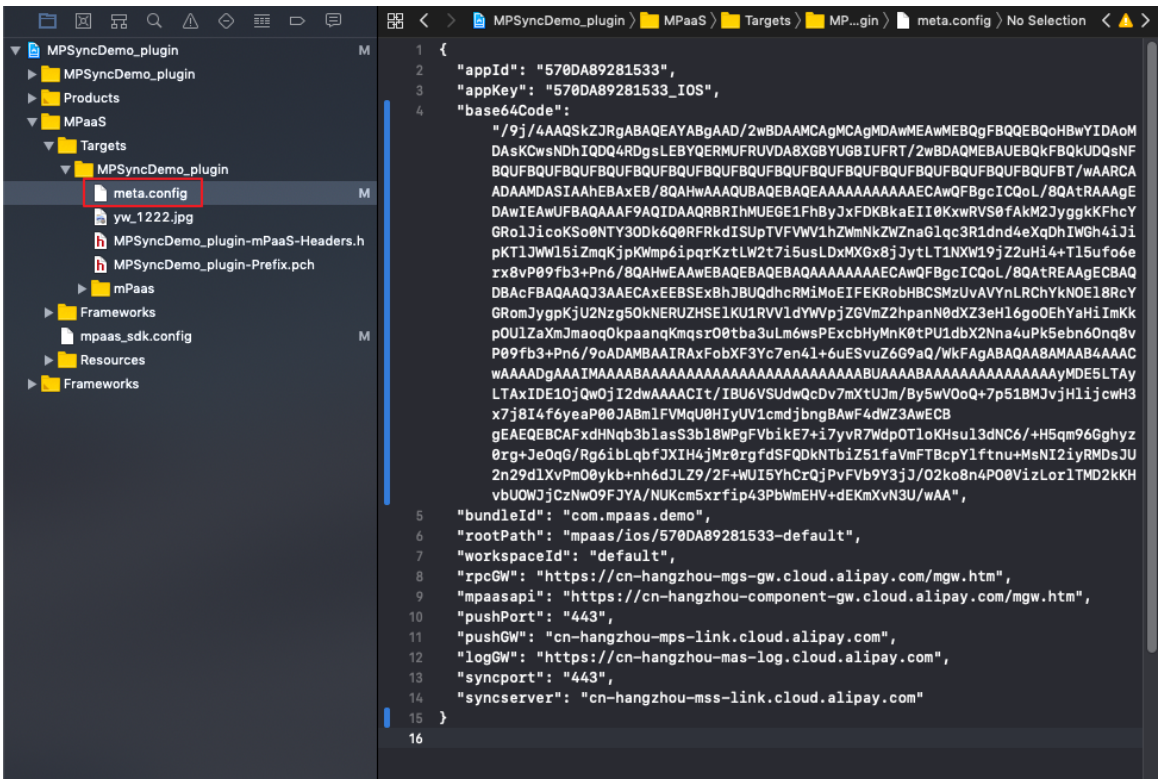
说明：您可以在 mpaas_sdk.config 文件中查看到当前使用的 SDK 版本。



配置工程

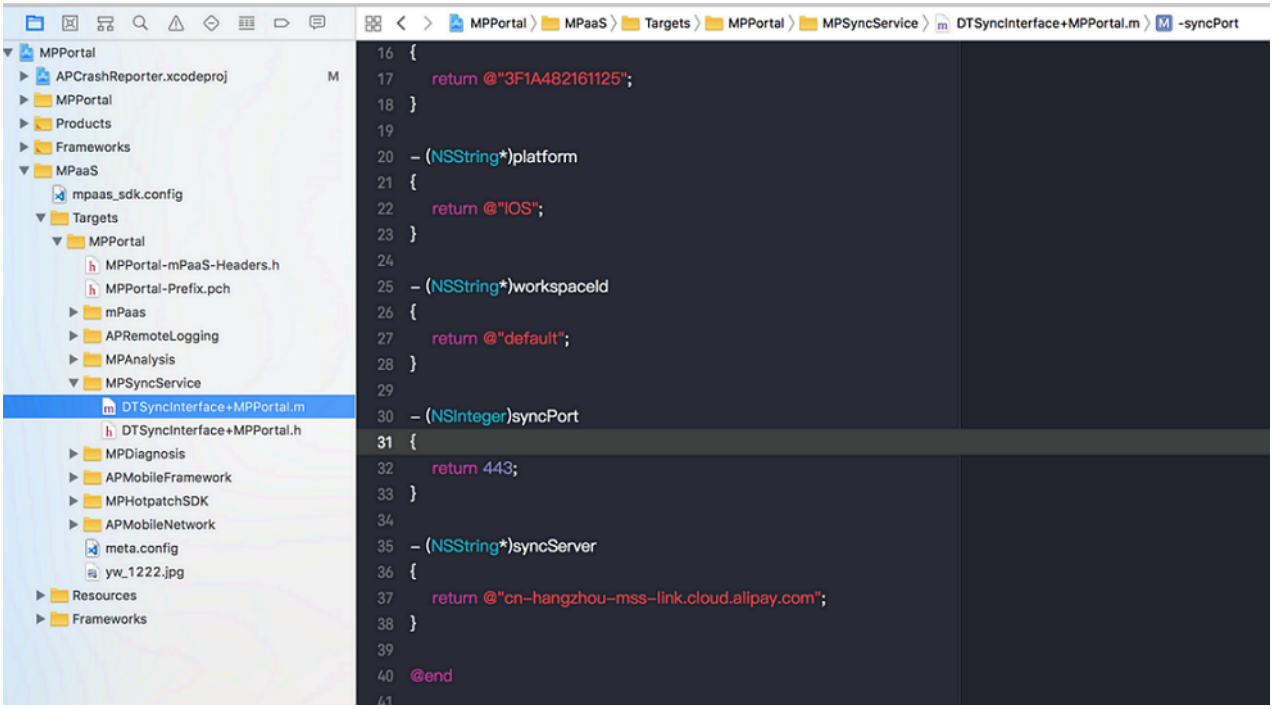
确保工程中添加了 meta.config 文件，该文件包含 Sync 服务地址、端口等配置信息：

- 若您使用最新版插件添加了 Sync SDK，那么该文件会自动生成。
- 否则，您可以到 **控制台** > **代码管理** > **代码配置** 下载 .config 配置文件，重命名为 meta.config 后，添加到工程中。



旧版本升级注意事项

10.1.32 版本之后不再需要添加 DTSyncInterface 类的 Category 文件，中间层会实现包装从 meta.config 中读取，升级版本后请检查工程中是否存在旧版本配置，如果有请移除。下面为新版本应移除的 DTSyncInterface 类的 Category 文件。



示例

以下为注册业务、获取数据代码的示例。在接入代码之前需要申请好具体的同步业务的业务标识，获取到对应的名字。这个名字就是作为使用者的您和服务提供者联系的纽带。以下示例中的同步配置标识为 SYNC-TRADE-DATA。

为了实现监听同步业务的逻辑，需要创建一个类、最好是常驻内存的服务来一直监听 Sync 消息。例如下面的示例，创建了一个 MySyncService 类来监听同步业务的逻辑。

```
#import <MPMssAdapter/MPSyncInterface.h>
#define SYNC_BIZ_NAME"SYNC-TRADE-DATA";

@implementation MySyncService
+ (instancetype)sharedInstance
{
static MySyncService *bizService;

static dispatch_once_t llSOnceToken;

dispatch_once(&llSOnceToken, ^{

bizService = [[self alloc] init];
});
return bizService;
}

-(instancetype)init
{
self = [super init];
if (self) {
[MPSyncInterface initSync];
BOOL registerSingleDeviceSync = [MPSyncInterface registerSyncBizWithName:SYNC_BIZ_NAME syncObserver:self
selector:@selector(revSyncBizNotification:)];
[MPSyncInterface bindUserWithSessionId:@"SESSION_DEMO"]; // 此处的 User 对应控制台下发命令时，所需要填写的
userId, 需要和 MPaaSInterface 的 userId 函数中配置的值相对应；sessionId 是客户端携带的授权 token，userId 和
sessionId 都是用户登录系统返回的数据，当 userId 和 sessionId 变化时，需要重新调用此函数才能保证长连接的建立正确。
}
return self;
}

-(void)revSyncBizNotification:(NSNotification*)notify
{
NSDictionary *userInfo = notify.userInfo;
dispatch_async(dispatch_get_main_queue(), ^{
//业务数据处理
[MySyncService handleSyncData:userInfo];
//回调 SyncSDK，表示业务数据已经处理
[MPSyncInterface responseMessageNotify:userInfo];
});
}

+(void)handleSyncData:(NSDictionary *)userInfo
{
NSString * stringOp = userInfo[@"op"];
NSArray *op = [NSJSONSerialization JSONObjectWithData:[stringOp dataUsingEncoding:NSUTF8StringEncoding]
options:NSJSONReadingMutableContainers error:nil];
if([op isKindOfClass:[NSArray class]]){
```

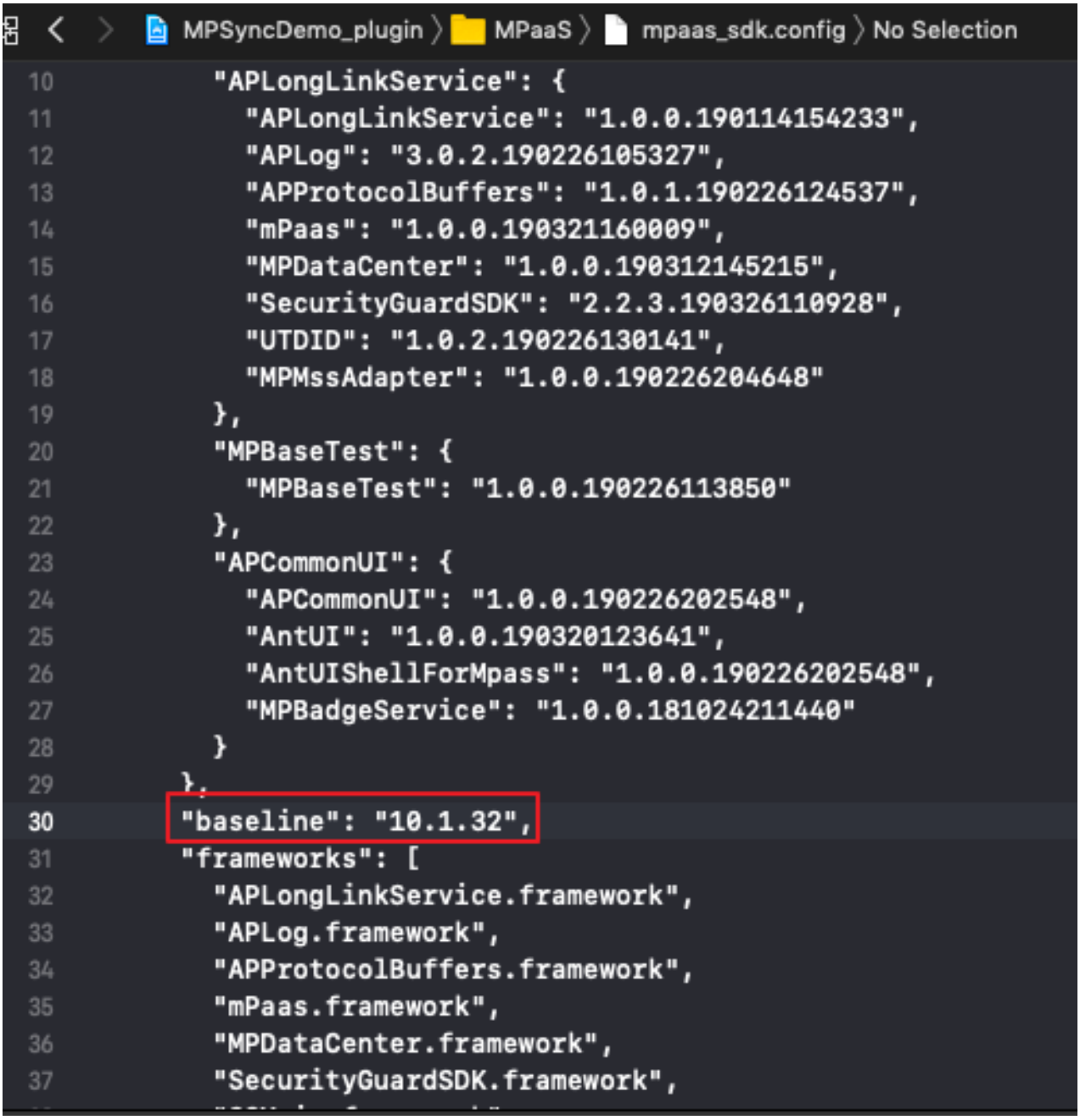
```
[op enumerateObjectsUsingBlock:^(NSDictionary * item, NSUInteger idx, BOOL *stop) {
if([item isKindOfClass:[NSDictionary class]]){
NSString * plString = item[@"pl"];//业务数据 payload
if(item[@"isB"]){
NSData *dataPl = [[NSData alloc] initWithBase64EncodedString:plString options:kNilOptions];
NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8StringEncoding];
NSLog(@"biz payload data:%@,string:%@",dataPl,pl);
}else{
NSLog(@"biz payload:%@",plString);
}
}
}];
}
}

-(void)dealloc
{
BOOL unRegisterSingleDeviceSync = [MPSyncInterface unRegisterSyncBizWithName:SYNC_BIZ_NAME
syncObserver:[MySyncService sharedInstance]];
[MPSyncInterface removeSyncNotificationObserver:self];
}
@end
```

4.3.4 使用 SDK（版本 < 10.1.32）

前置条件

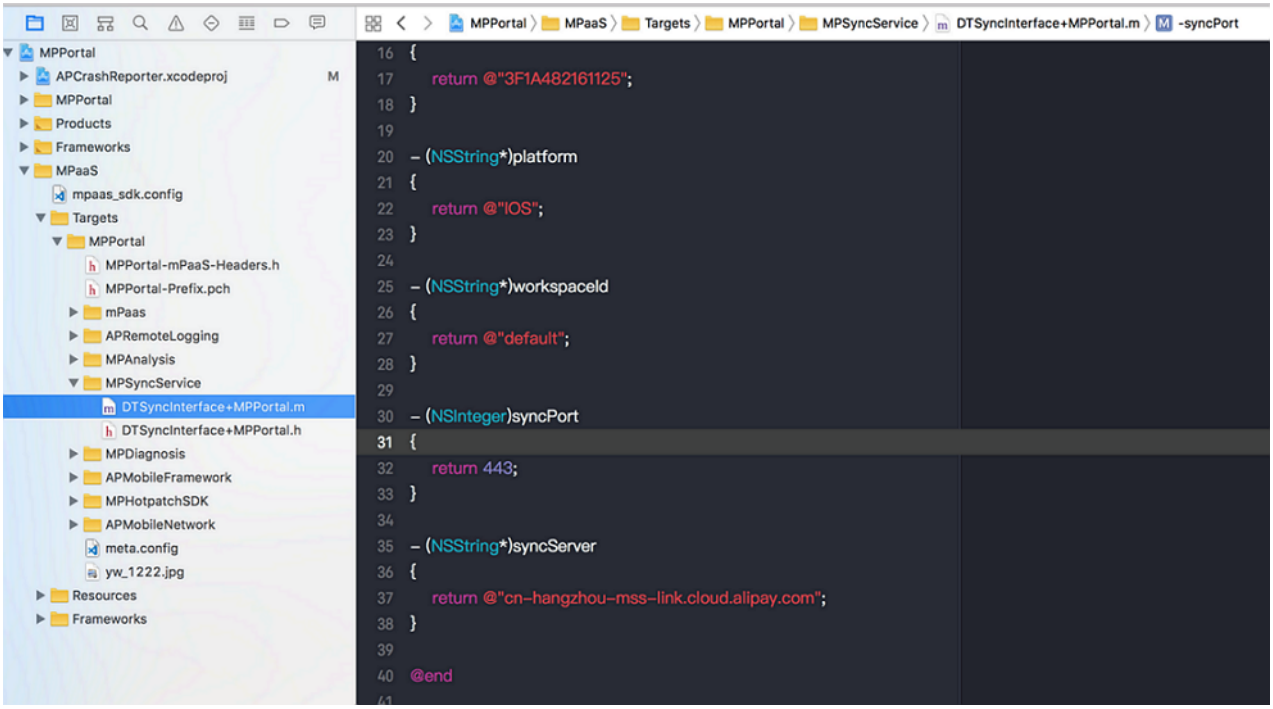
您已确认所使用的 SDK 版本 < 10.1.32。如果您使用的 SDK 版本 ≥ 10.1.32，请参考 使用 SDK（版本 ≥ 10.1.32）。

说明：您可以在 mpaas_sdk.config 文件中查看到当前使用的 SDK 版本。



配置工程

为保证客户端能和服务端成功建立长连接，需要在客户端工程中配置 Sync 地址、端口号等。请参考以下操作进行配置。



示例

以下为注册业务、获取数据代码的示例。在接入代码之前需要申请好具体的同步业务的业务标识，获取到对应的名字。这个名字就是作为使用者的您和服务提供者联系的纽带。以下示例中的同步配置标识为 SYNC-TRADE-DATA。

为了实现监听同步业务的逻辑，需要创建一个类、最好是常驻内存的服务来一直监听 Sync 消息。例如下面的示例，创建了一个 MySyncService 类来监听同步业务的逻辑。

```
#import <APLongLinkService/APLongLinkService.h>
#define SYNC_BIZ_NAME"SYNC-TRADE-DATA";

@implementation MySyncService
+ (instancetype)sharedInstance
{
static MySyncService *bizService;

static dispatch_once_t llSOnceToken;

dispatch_once(&llSOnceToken, ^{

bizService = [[self alloc] init];
});
return bizService;
}

-(instancetype)init
{
self = [super init];
if (self) {
[self syncInit];
[self regSyncBiz];
```

```
[self bindUser];
}
return self;
}

- (void)syncInit
{
[DTLongLinkBusiness syncInit];
}

-(void)regSyncBiz{
[[NSNotificationCenter defaultCenter]addObserver:self selector:@selector(revSyncBizNotification:)
name:SYNC_BIZ_NAME object:nil];
[DTLongLinkBusiness hasRegisterNotificationWithBiz:SYNC_BIZ_NAME];
}

- (void)bindUser
{
[DTLongLinkBusiness sendBindUser:uid sessionId:sid]; // 此处的 uid 对应控制台下发命令时，所需要填写的 userId 和 sessionId 是客户端携带的授权 token，userId 和 sessionId 都是用户登录系统返回的数据，当 userId 和 sessionId 变化时，需要重新调用此函数才能保证长连接的建立正确。
}

-(void)revSyncBizNotification:(NSNotification*)notify
{
NSDictionary *userInfo = notify.userInfo;
dispatch_async(dispatch_get_main_queue(), ^{

//业务数据处理
[MySyncService handleSyncData:userInfo];
//回调 SyncSDK，表示业务数据已经处理
[DTLongLinkBusiness responseMessageNotify:userInfo];
});

}

+(void)handleSyncData:(NSDictionary *)userInfo
{
NSString * stringOp = userInfo[@"op"];
NSArray *op = [NSJSONSerialization JSONObjectWithData:[stringOp dataUsingEncoding:NSUTF8StringEncoding]
options:NSJSONReadingMutableContainers error:nil];
if([op isKindOfClass:[NSArray class]]){
[op enumerateObjectsUsingBlock:^(NSDictionary * item, NSUInteger idx, BOOL *stop) {
if([item isKindOfClass:[NSDictionary class]]){
NSString * plString = item[@"pl"];//业务数据 playload
if(item[@"isB"]){
NSData *dataPl = [[NSData alloc] initWithBase64EncodedString:plString options:kNilOptions];
NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8StringEncoding];
NSLog(@"biz playload data:%@,string:%@",dataPl,pl);
}else{
NSLog(@"biz playload:%@",plString);
}
}
}];
}
}
```

```
-(void)dealloc
{
[DTLongLinkBusiness unRegisterNotificationWithBiz:SYNC_BIZ_NAME];
[[NSNotificationCenter defaultCenter]removeObserver:self];
}
@end
```

4.4 基于原生框架接入 iOS 工程到 mPaaS

4.4.1 基础配置

为接入 iOS 工程到 mPaaS，您需要执行以下步骤以完成基础配置。

1. 安装开发者工具 。
2. 在控制台创建应用 。
3. 基于原生 iOS 框架的工程，使用 mPaaS 插件接入 mPaaS 。

4.4.2 添加 SDK

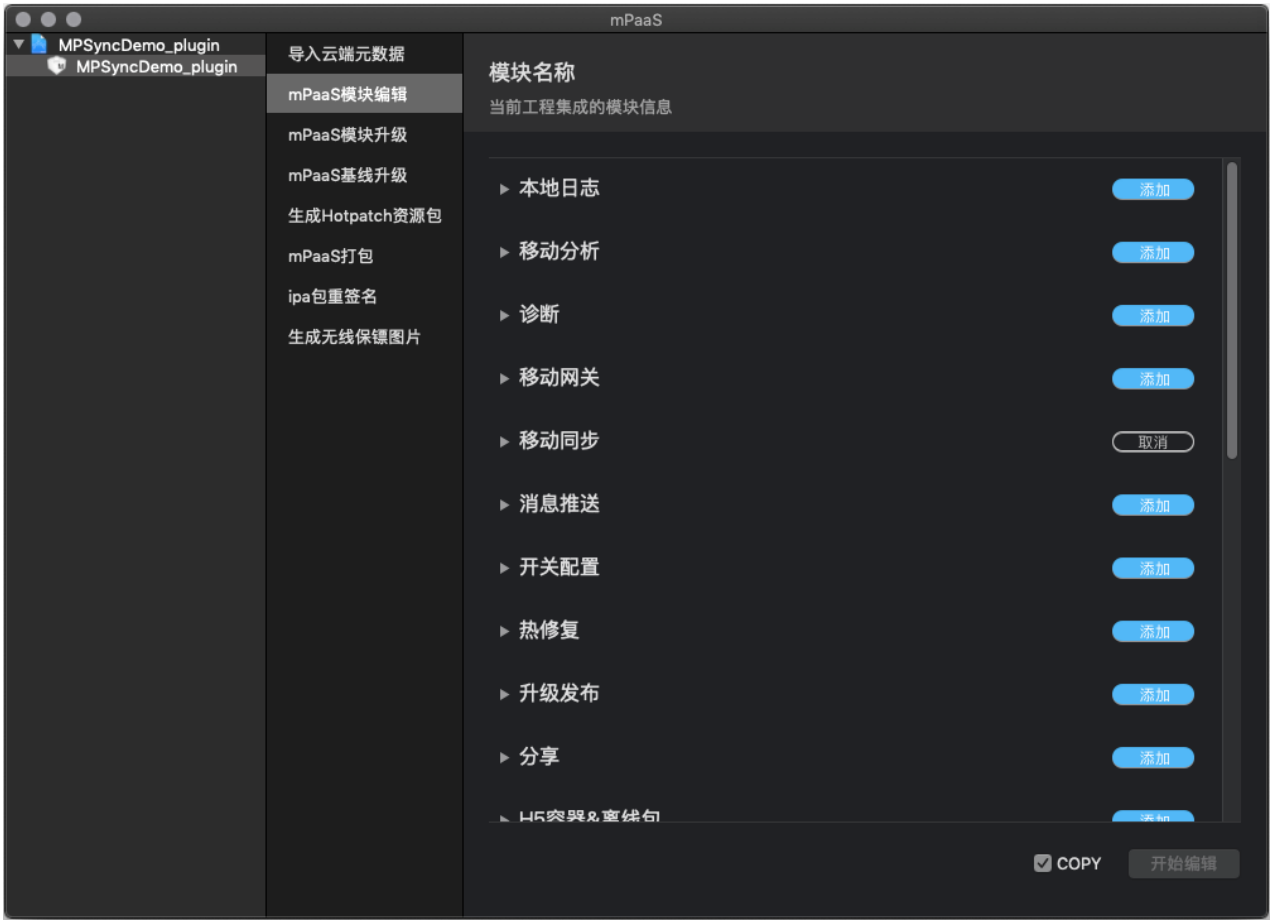
要将 MSS 接入 iOS 客户端，需要添加 MSS SDK 并在使用时对工程进行配置。本文将引导您在基于 mPaaS 框架的接入方式下，为 iOS 工程添加 SDK。

前置条件

您已完成基础配置，将原生 iOS 框架的工程，使用 mPaaS 插件接入 mPaaS。

添加 SDK

如下图所示，使用 mPaaS 插件添加 Sync SDK。更多信息，请参考 mPaaS 插件 > 编辑模块 。

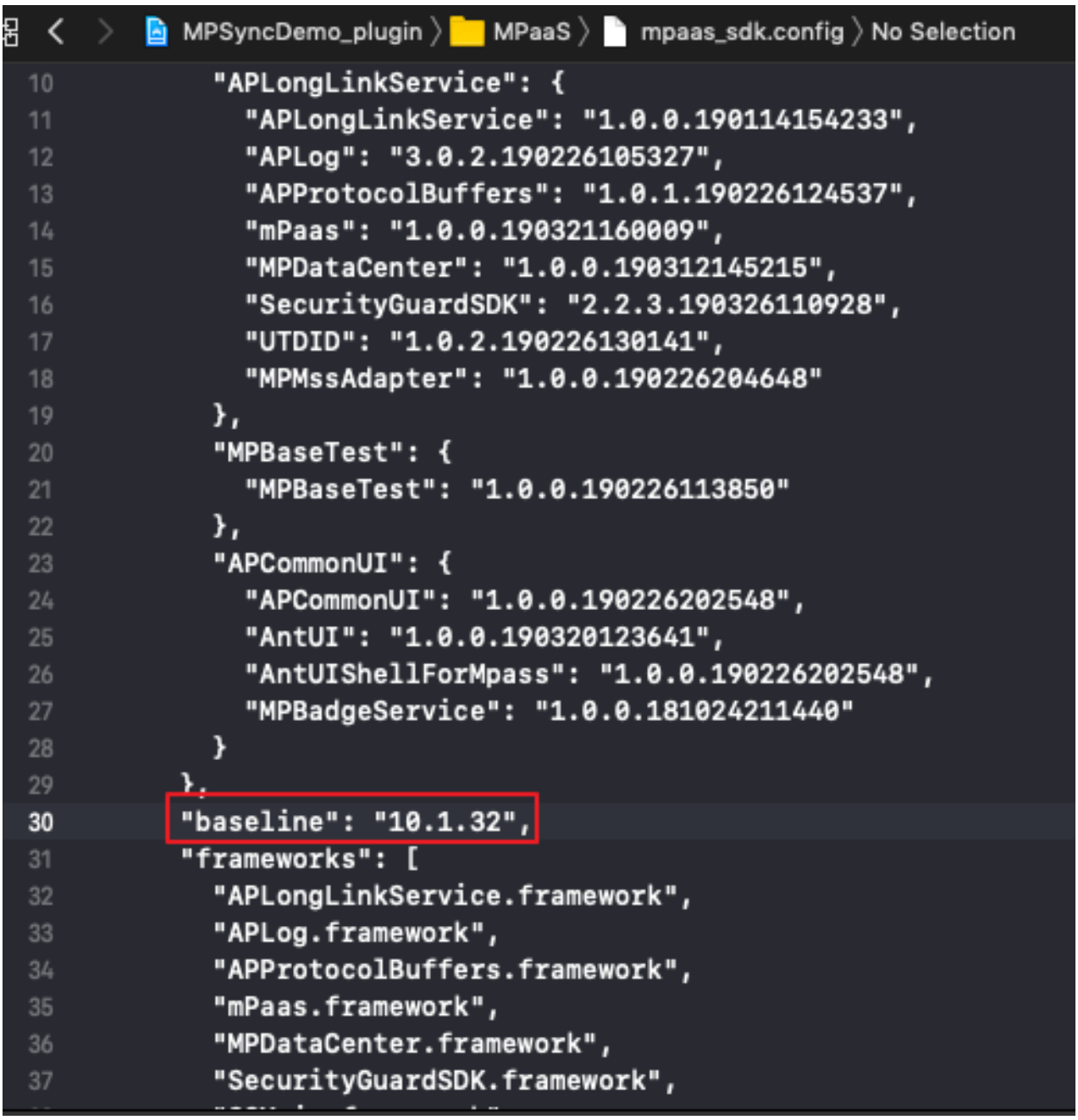


后续步骤

根据不同的 SDK版本，您将需要采取不同的后续操作以使用SDK。请根据您使用的 SDK 版本查看对应的 SDK 使用文档。

- 使用SDK（版本 ≥ 10.1.32）
- 使用SDK（版本 < 10.1.32）

说明
您可以在 mpaas_sdk.config 文件中查看到当前使用的 SDK 版本。

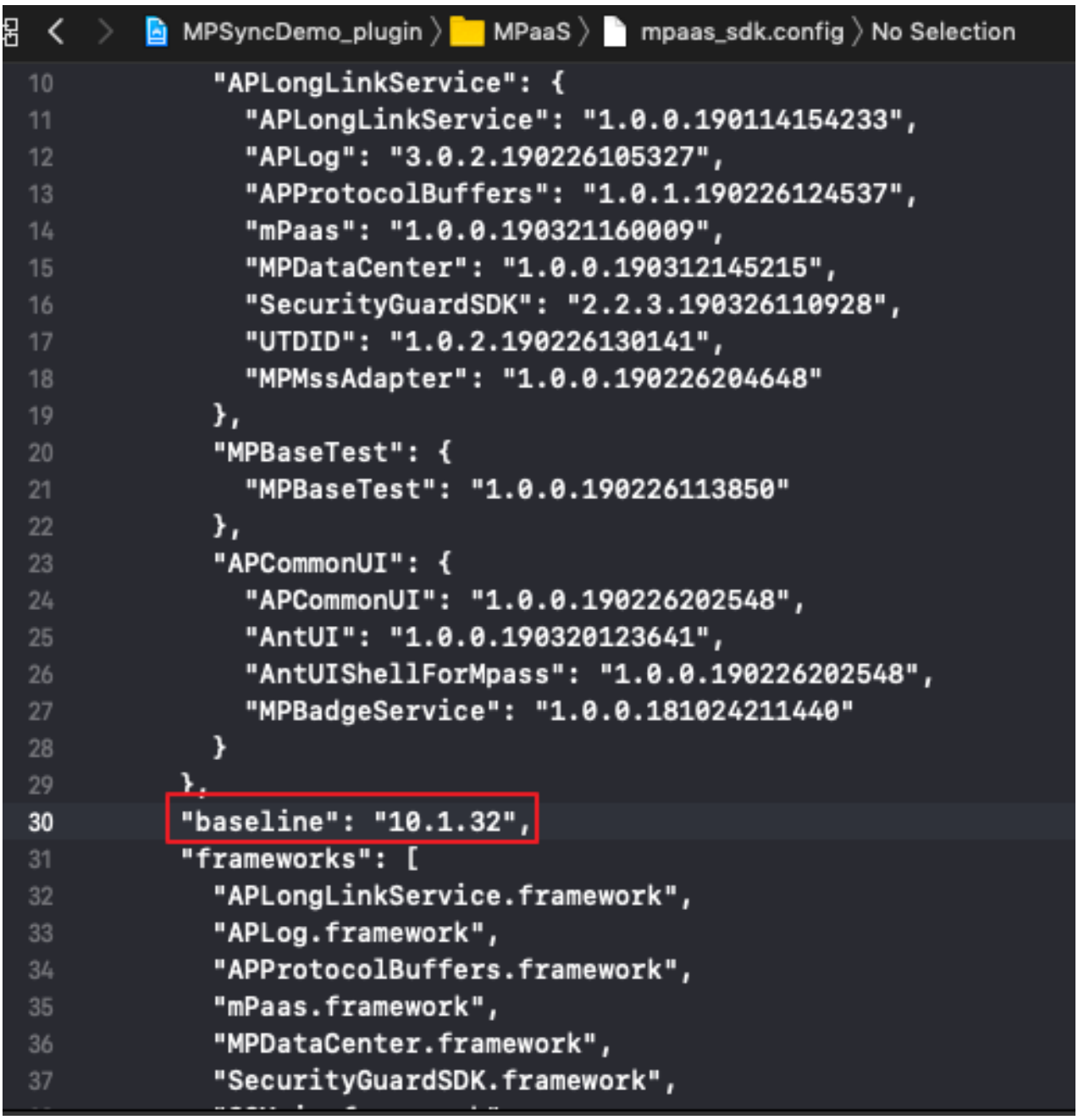


4.4.3 使用SDK（版本 ≥ 10.1.32）

前置条件

您已经确认到您使用的SDK版本 ≥ 10.1.32。如果您使用的SDK版本 < 10.1.32，请参考 使用SDK（版本 < 10.1.32）。

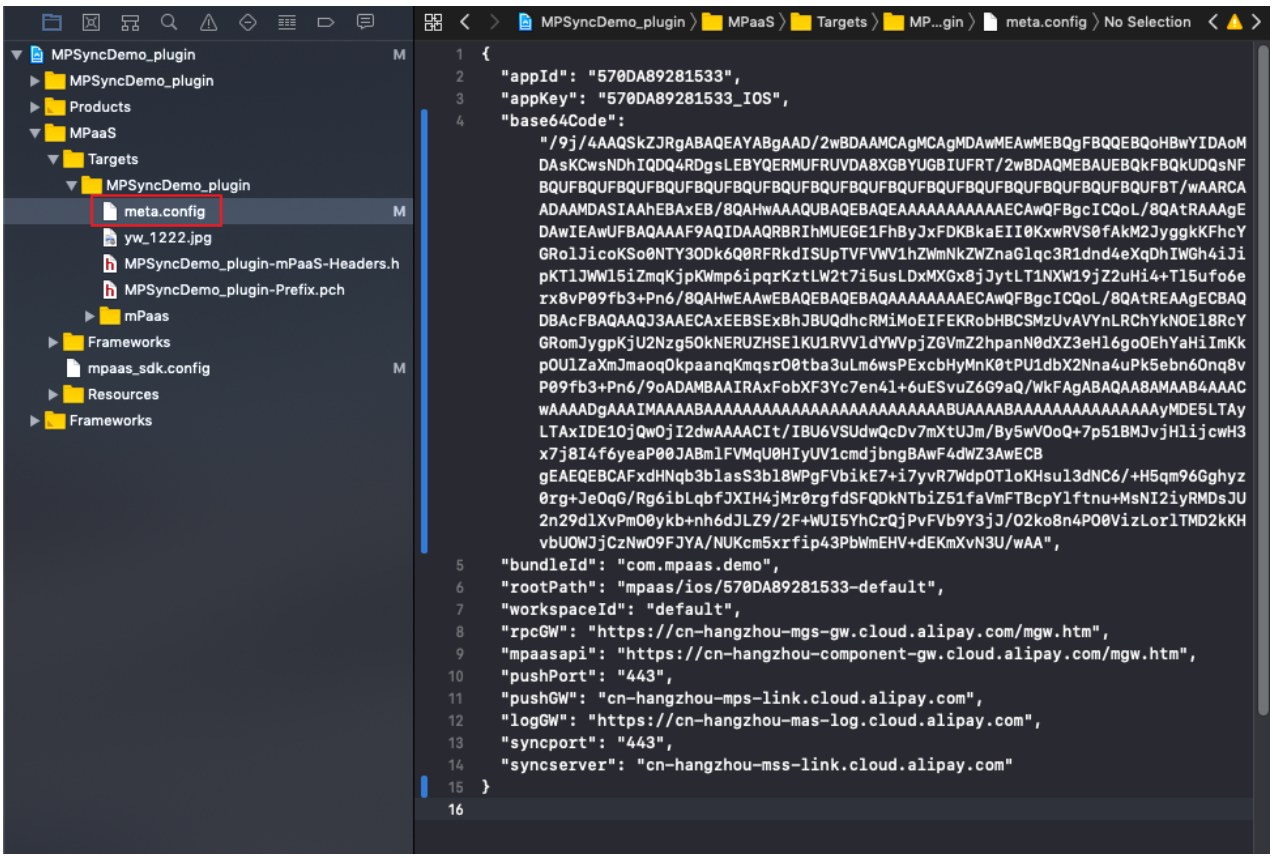
说明
您可以在 mpaas_sdk.config 文件中查看到当前使用的 SDK 版本。



配置工程

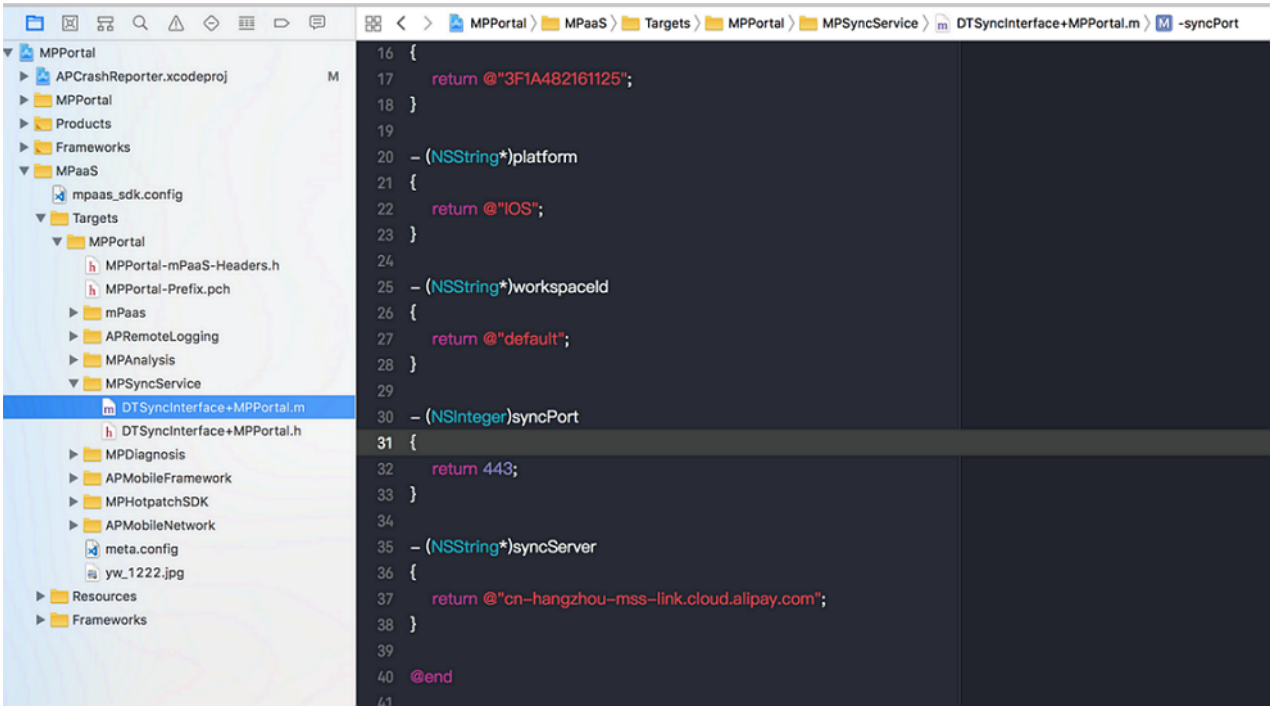
确保工程中添加了 meta.config 文件，该文件包含 Sync 服务地址、端口等配置信息：

- 若您使用最新版插件添加了 Sync SDK，那么该文件会自动生成。
- 如果您的工程中没有 meta.config 文件，那么您需要到到 **控制台** > **代码管理** > **代码配置** 下载 .config 配置文件，重命名为 meta.config 后，添加到工程中。



旧版本升级注意事项

10.1.32 版本之后不再需要添加 DTSyncInterface 类的 Category 文件，中间层会实现包装从 meta.config 中读取，升级版本后请检查工程中是否存在旧版本配置，如果有请移除。下面为新版本应移除的 DTSyncInterface 类的 Category 文件。



示例

以下为注册业务、获取数据代码的示例。
在接入代码之前需要申请好具体的同步业务的业务标识，获取到对应的名字。这个名字就是作为使用者的您和服务提供者联系的纽带。以下示例中的同步配置标识为 SYNC-TRADE-DATA。

为了实现监听同步业务的逻辑，需要创建一个类、最好是常驻内存的服务来一直监听 Sync 消息。例如下面的示例，创建了一个 MySyncService 类来监听同步业务的逻辑。

```
#import <MPMssAdapter/MPSyncInterface.h>
#define SYNC_BIZ_NAME"SYNC-TRADE-DATA";

@implementation MySyncService
+ (instancetype)sharedInstance
{
static MySyncService *bizService;

static dispatch_once_t llSOnceToken;

dispatch_once(&llSOnceToken, ^{

bizService = [[self alloc] init];
});
return bizService;
}

-(instancetype)init
{
self = [super init];
if (self) {
[MPSyncInterface initSync];
BOOL registerSingleDeviceSync = [MPSyncInterface registerSyncBizWithName:SYNC_BIZ_NAME syncObserver:self
selector:@selector(revSyncBizNotification:)];
[MPSyncInterface bindUserWithSessionId:@"SESSION_DEMO"]; // 此处的 User 对应控制台下发命令时，所需要填写的
userId, 需要和 MPaaSInterface 的 userId 函数中配置的值相对应；sessionId 是客户端携带的授权 token，userId 和
sessionId 都是用户登录系统返回的数据，当 userId 和 sessionId 变化时，需要重新调用此函数才能保证长连接的建立正确。
}
return self;
}

-(void)revSyncBizNotification:(NSNotification*)notify
{
NSDictionary *userInfo = notify.userInfo;
dispatch_async(dispatch_get_main_queue(), ^{
//业务数据处理
[MySyncService handleSyncData:userInfo];
//回调 SyncSDK，表示业务数据已经处理
[MPSyncInterface responseMessageNotify:userInfo];
});
}

+(void)handleSyncData:(NSDictionary *)userInfo
{
NSString * stringOp = userInfo[@"op"];
```

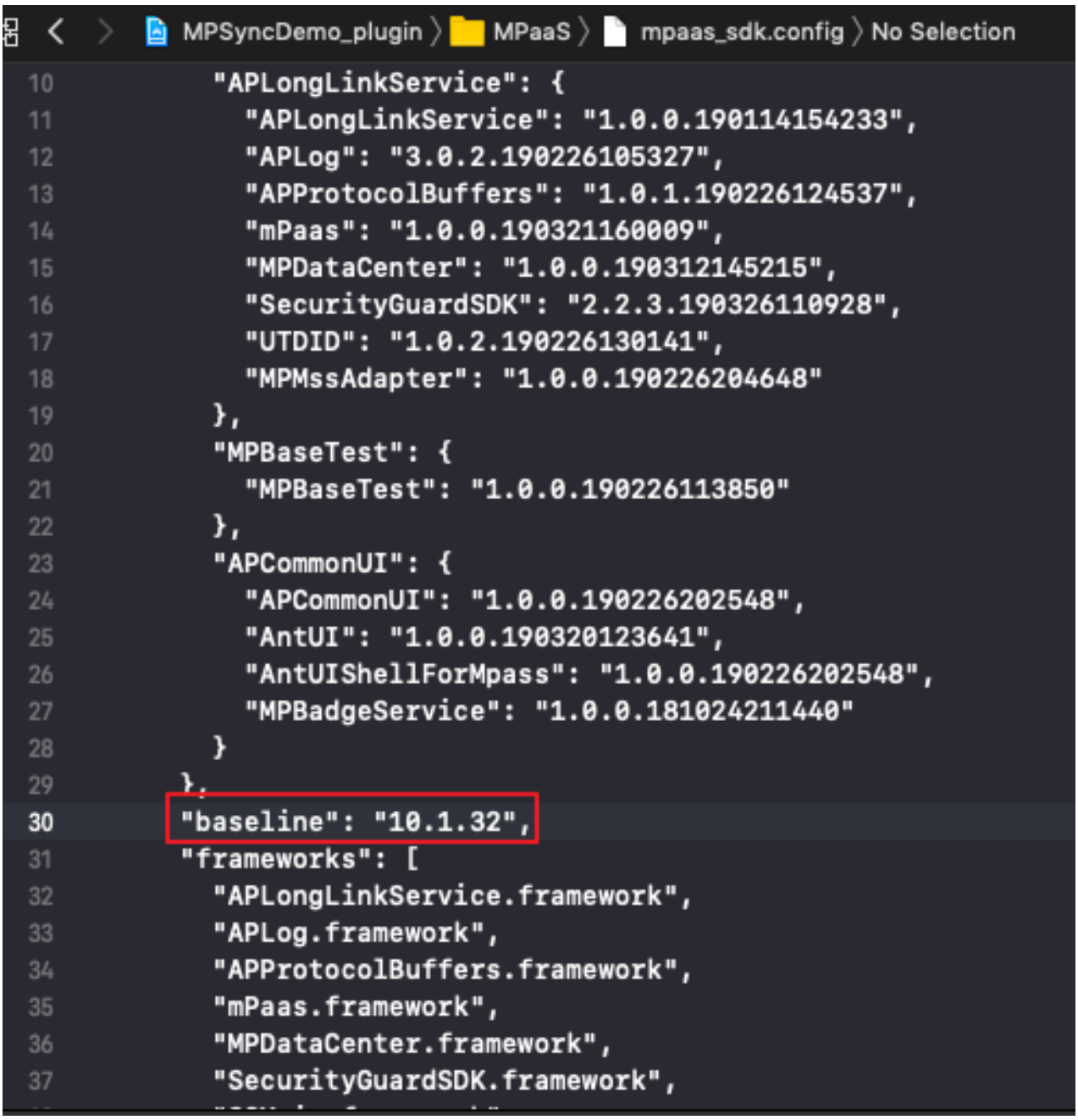
```
NSArray *op = [NSJSONSerialization JSONObjectWithData:[stringOp dataUsingEncoding:NSUTF8StringEncoding]
options:NSJSONReadingMutableContainers error:nil];
if([op isKindOfClass:[NSArray class]]){
[op enumerateObjectsUsingBlock:^(NSDictionary * item, NSUInteger idx, BOOL *stop) {
if([item isKindOfClass:[NSDictionary class]]){
NSString * plString = item[@"pl"];//业务数据 payload
if(item[@"isB"]){
NSData *dataPl = [[NSData alloc] initWithBase64EncodedString:plString options:kNilOptions];
NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8StringEncoding];
NSLog(@"biz payload data:%@,string:%@",dataPl,pl);
}else{
NSLog(@"biz payload:%@",plString);
}
}
}];
}
}

-(void)dealloc
{
BOOL unRegisterSingleDeviceSync = [MPSyncInterface unRegisterSyncBizWithName:SYNC_BIZ_NAME
syncObserver:[MySyncService sharedInstance]];
[MPSyncInterface removeSyncNotificationObserver:self];
}
@end
```

4.4.4 使用SDK（版本 < 10.1.32）

前置条件

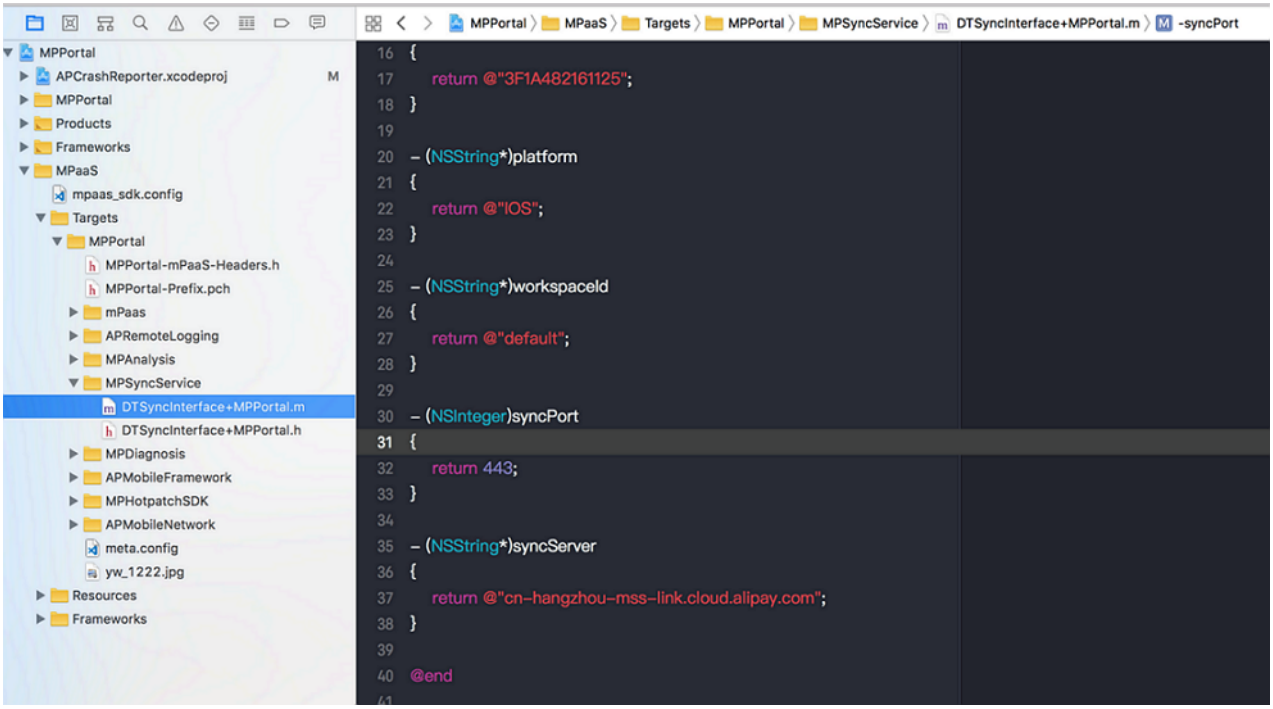
您已经确认到您使用的SDK版本 < 10.1.32。如果您使用的SDK版本 ≥ 10.1.32，请参考 使用SDK（版本 ≥ 10.1.32）。

说明
您可以在 mpaas_sdk.config 文件中查看到当前使用的 SDK 版本。



配置工程

为保证客户端能和服务端成功建立长连接，需要在客户端工程中配置 Sync 地址、端口号等。请参考以下操作进行配置。



示例：注册业务，获取数据代码
在接入代码之前需要申请好具体的同步业务的业务标识，获取到对应的名字。这个名字就是作为使用者的您和服务提供者联系的纽带。以下示例中的同步配置标识为 SYNC-TRADE-DATA。

为了实现监听同步业务的逻辑，需要创建一个类、最好是常驻内存的服务来一直监听 Sync 消息。例如下面的示例，创建了一个 MySyncService 类来监听同步业务的逻辑。

```
#import <APLongLinkService/APLongLinkService.h>
#define SYNC_BIZ_NAME"SYNC-TRADE-DATA";

@implementation MySyncService
+ (instancetype)sharedInstance
{
static MySyncService *bizService;

static dispatch_once_t llSOnceToken;

dispatch_once(&llSOnceToken, ^{

bizService = [[self alloc] init];
});
return bizService;
}

-(instancetype)init
{
self = [super init];
if (self) {
[self syncInit];
[self regSyncBiz];
[self bindUser];
}
```

```
return self;
}

- (void)syncInit
{
[DTLongLinkBusiness syncInit];
}

-(void)regSyncBiz{
[[NSNotificationCenter defaultCenter]addObserver:self selector:@selector(revSyncBizNotification:)
name:SYNC_BIZ_NAME object:nil];
[DTLongLinkBusiness hasRegisterNotificationWithBiz:SYNC_BIZ_NAME];
}

- (void)bindUser
{
[DTLongLinkBusiness sendBindUser:uid sessionId:sid]; // 此处的 uid 对应控制台下发命令时，所需要填写的 userId 和
sessionId 是客户端携带的授权 token，userId 和 sessionId 都是用户登录系统返回的数据，当 userId 和 sessionId 变化时
，需要重新调用此函数才能保证长连接的建立正确。
}

-(void)revSyncBizNotification:(NSNotification*)notify
{
NSDictionary *userInfo = notify.userInfo;
dispatch_async(dispatch_get_main_queue(), ^{

//业务数据处理
[MySyncService handleSyncData:userInfo];
//回调 SyncSDK，表示业务数据已经处理
[DTLongLinkBusiness responseMessageNotify:userInfo];
});

}

+(void)handleSyncData:(NSDictionary *)userInfo
{
NSString * stringOp = userInfo[@"op"];
NSArray *op = [NSJSONSerialization JSONObjectWithData:[stringOp dataUsingEncoding:NSUTF8StringEncoding]
options:NSJSONReadingMutableContainers error:nil];
if([op isKindOfClass:[NSArray class]]){
[op enumerateObjectsUsingBlock:^(NSDictionary * item, NSUInteger idx, BOOL *stop) {
if([item isKindOfClass:[NSDictionary class]]){
NSString * plString = item[@"pl"];//业务数据 playload
if(item[@"isB"]){
NSData *dataPl = [[NSData alloc] initWithBase64EncodedString:plString options:kNilOptions];
NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8StringEncoding];
NSLog(@"biz playload data:%@,string:%@",dataPl,pl);
}else{
NSLog(@"biz playload:%@",plString);
}
}
}];
}
}

-(void)dealloc
```

```
{
[DTLongLinkBusiness unRegisterNotificationWithBiz:SYNC_BIZ_NAME];
[[NSNotificationCenter defaultCenter]removeObserver:self];
}
@end
```

5 接入服务端

5.1 接入专有云到 MSS 服务端

5.1.1 接入简介

专有云租户业务系统接入 MSS（Mobile Sync Service）的服务器端的开发流程包含 **配置服务** 和 **编写调用代码并处理结果** 两部分。对同步数据有较高用户安全性要求的业务场景，在 完成接入后，还需进行 用户一致性验证。在编写调用代码并处理结果时，根据采用同步接口的不同，又分为使用单数据同步接口和使用全局数据同步接口两种方式进行介绍。
说明：在专有云环境下使用的是客户机房内部网络，客户业务系统无需通过蚂蚁金融云提供的 OpenAPI 体系，即可直接对接在专有云环境下部署的 MSS 服务器。请根据实际部署环境，修改接口调用域名（IP）地址。

前置条件

1. 您已在专有云环境下部署好 MSS 服务，且各项健康检查、端口检测均已验证通过。
2. 您已创建好一个 APP，了解客户端接入工作的进度，并且能够取到该 APP 的 App ID 和 workspace ID。
3. 您有一个可发起 HTTP 调用的服务端应用。

配置服务

MSS 服务端与客户端之间基于 **BizType（同步标识）** 进行交互，**BizType** 为同一 APP 在某一工作环境下，不同业务数据之间逻辑隔离的核心属性。在调用服务端接口之前，需要先进行推送配置，详细操作请参见 新增配置 。

编写调用代码并处理结果

根据采用同步接口的不同，又分为使用单数据同步接口和使用全局数据同步接口两种方式。请根据需要阅读以下两篇文档以获得更多详细信息。

- 使用单数据同步接口
- 使用全局数据同步接口

5.1.2 单数据同步接口

接口说明

同步单条数据至指定用户或设备。

- 当推送目标用户或设备处于在线状态时，推送数据将直接在线同步至终端。
- 当推送目标用户或设备处于不在线转状态时，推送数据将持久化至数据库，待终端再次上线并与 MSS 服务建立连接后，再进行同步操作（配置数据持久化操作请参见 新增配置 ）。

接口 URL 示例

```
http://11.160.18.15/webapi/sync/single?instanceId=sit_320C94C171133
```

- IP（域名）地址：请替换成现场部署的 MSS 服务器 IP 配置。
- instanceId：组成结构为 workspaceId_AppId，instanceID 为用于 APP 数据逻辑隔离的核心属性。

代码示例

```
import net.sf.json.JSONObject;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

/**
* 单条推送，使用httpclinet模拟post请求发送到指定服务器（只需要修改apiURL中的服务 器ip地址即可）
*/
public class SingleTest {
private static String apiURL ="http://11.162.169.36/webapi/sync/single?instanceId=sit_320C94C171133";

/**
* post请求
*
* @param url URL
* @param json JSONObject
* @return JSONObject
*/

private static JSONObject doPost(String url, JSONObject json) {
CloseableHttpClient httpclient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost(url);
JSONObject response = null;
try {
StringEntity s = new StringEntity(json.toString());
StringEntity s = new StringEntity(json.toString(),"UTF-8");
s.setContentEncoding("UTF-8");
//发送json数据需要设置contentType
s.setContentType("application/json");
httpPost.setEntity(s);
CloseableHttpResponse res = httpclient.execute(httpPost);
if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
HttpEntity entity = res.getEntity();
```

```
//返回json格式：
String result = EntityUtils.toString(entity);
response = JSONObject.fromObject(result);
}
} catch (Exception e) {
throw new RuntimeException(e);
}
return response;
}

public static void main(String arg[]) {
JSONObject params = new JSONObject();
//bizType
params.put("bizType","UCHAT");
//用户ID or 设备ID
params.put("linkToken","2088102147396225");
//消息体，原样透传至客户端
JSONObject payload = new JSONObject();
payload.put("name","李四");
payload.put("age","30");
payload.put("address","浙江省杭州市");
//String格式
params.put("payload", JSON.toJSONString(payload));

//业务ID，可包含业务规则，长度不超过100，本DEMO为随机一个值
Double i = Math.random() * 1000000;
String[] num = i.toString().split("\\.");
params.put("thirdMsgId","test_third_msg_id_"+ num[0] + System.currentTimeMillis());

//可支持的最小客户端版本
params.put("appMinVersion","0.0.0.0");
//可支持的最大客户端版本
params.put("appMaxVersion","100.100.100.100");

//当前时间
long today = System.currentTimeMillis();
//有效期30天
long validTimeEnd = today + 30 * 24 * 60 * 60 * 1000L;
//有效期
params.put("validTimeEnd", validTimeEnd);

//提交Post请求
JSONObject ret = doPost(apiURL, params);
System.out.println(ret);
}
}
```

请求参数

参数名称	是否必填	数据类型	最大长度	说明
bizType	是	String	30	业务场景标识：在 mPaaS 控制台上进行推送配置时设置的 同步标识，详细操作请参见 管理控制台 。
linkToken	是	String	100	根据配置的 bizType 决定，用户维度业务填写 userId, 设备维度业务填写 utdid。

payload	否	String	4096	推送的数据，为文本字符串或 Josn 字符串。与 binaryPayload 二选一，不可都为空。
thirdMsgId	是	String	100	一次数据同步请求 ID，由业务方自定义，用于数据关联和幂等控制。bizType + linkToken + thirdMsgId 需唯一，重复时，对应的新推送数据不予入库，接口返回调用成功。
binaryPayload	否	String	4096	推送的数据，由一个原始 byte 数组经过 base64 编码之后生成的字符串。与 payload 二选一，不可都为空。
osType	否	String	10	指定消息推送的客户端操作系统类型，（iOS / Android），不限制操作系统类型时，留空。
appMinVersion	否	String	20	支持的最小客户端版本号【包含】，仅推送至大于等于该版本的客户端。例如：8.6.0.9999 ，强烈建议使用标准格式的版本号。
appMaxVersion	否	String	20	支持的最大客户端版本号【包含】，仅推送至小于等于该版本的客户端。如：9.0.0.9999 ，强烈建议使用标准格式的版本号。
validTimeEnd	否	long	13	推送有效期结束时间，过期后，MSS 服务端将不再推送过期数据至客户端。格式： (new Date()).getTime(); 。

返回参数

返回数据为 JOSN 格式，示例代码如下：

```
{
"msg":"SUCCESS",
"success": true
}
```

各属性含义及解释：

名称	类型	示例	说明
success	boolean	true / false	业务调用结果，成功返回 true，失败返回 false。失败的情况下，可通过 msg 查看失败原因，参考下表 结果码 。
msg	String	SUCCESS	结果码 。

结果码:

调用结果	结果码	含义
true	SUCCESS	业务成功 - 在线推送成功
true	DUPLICATED_BIZ_ID	bizId 重复，在线推送成功
true	NOT_ONLINE	用户 / 设备不在线
false	THIRDMSGID_IS_NULL	thirdMsgId 为空
false	BIZ_NOT_ONLINE	同步配置未提交上线
false	ARGS_IS_NULL	请求必选参数为空
false	NOT_SUPPORT_GLOBAL	接口不支持全局业务
false	PAYLOAD_LONG	消息体超长
false	THIRD_MSG_ID_LONG	第三方消息 Id 过长
false	SYSTEM_ERROR	系统异常

5.1.3 全局数据同步接口

接口说明

同步一条数据至全网（所有）用户或设备。推送数据将持久化至数据库（配置数据持久化操作请参见 管理控制台 ）。因成本、收益和风险权衡，推送数据将不会立即同步当前应用在前台的用户，而是在应用下一次到前台时，由 MSS 客户端 SDK 触发同步。

接口 URL 示例

```
http://11.160.18.15/webapi/sync/global?instanceId=sit_320C94C171133
```

- IP（域名）地址：请替换成现场部署的 MSS 服务器 IP 配置。
- instanceID：组成结构为 workspaceId_AppId，instanceID 为 MSS 上用于 APP 数据逻辑隔离的核心属性。

代码示例

```
import com.alibaba.fastjson.JSON;
import net.sf.json.JSONObject;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

/**
* 全局推送，使用httpclinet模拟post请求发送到指定服务器（只需要修改apiURL中的服务 器ip地址即可）
*/
public class GlobalTest {
private static String apiURL ="http://11.160.18.15/webapi/sync/global?instanceId=default_99FB626081956";

/**
* post请求
*
* @param url String
* @param json JSONObject
* @return JSONObject
*/

private static JSONObject doPost(String url, JSONObject json) {
CloseableHttpClient httpclient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost(url);
JSONObject response = null;
try {
StringEntity s = new StringEntity(json.toString());
StringEntity s = new StringEntity(json.toString(),"UTF-8");
s.setContentEncoding("UTF-8");
//发送json数据需要设置contentType
```

```
s.setContentType("application/json");
httpPost.setEntity(s);
//httpPost.set
CloseableHttpResponse res = httpclient.execute(httpPost);
if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
// 返回json格式：
String result = EntityUtils.toString(res.getEntity());
response = JSONObject.fromObject(result);
}
} catch (Exception e) {
throw new RuntimeException(e);
}
return response;
}

public static void main(String arg[]) {
JSONObject params = new JSONObject();
//bizType
params.put("bizType","GLOBAL-SDK-CONFIG");

//消息体，原样透传至客户端
JSONObject payload = new JSONObject();
payload.put("switchName","showImage");
payload.put("switchValue", true);

//String格式
params.put("payload", JSON.toJSONString(payload));

//业务ID，可包含业务规则，长度不超过100，本DEMO为随机一个值
Double i = Math.random() * 1000000;
String[] num = i.toString().split("\\.");
params.put("thirdMsgId","test_third_msg_id_"+ num[0] + System.currentTimeMillis());

//IOS or ANDROID,不限制时勿传
params.put("osType","IOS");

//可支持的最小客户端版本,不限制时勿传
params.put("appMinVersion","0.0.0.0");
//可支持的最大客户端版本,不限制时勿传
params.put("appMaxVersion","100.100.100.100");

//当前时间
long today = System.currentTimeMillis();
//有效期30天
long validTimeEnd = today + 30 * 24 * 60 * 60 * 1000L;
//有效期
params.put("validTimeEnd", validTimeEnd);

//提交Post请求
JSONObject ret = doPost(apiURL, params);
System.out.println(ret);
}
}
```

请求参数

参数名称	是否必填	数据类型	最大长度	说明
bizType	是	String	30	业务场景标识：在 mPaaS 控制台上进行推送配置时设置的 **同步标识**，详细操作请参见 管理控制台 。
payload	否	String	4096	推送的数据，为文本字符串或 Josn 字符串，与 binaryPayload 二选一，不可都为空。
thirdMsgId	是	String	100	一次数据同步请求 ID，由业务方自定义，用于数据关联和幂等控制。bizType + linkToken + thirdMsgId 需唯一，重复时，保留前一次接口调用成功时的数据，抛弃新数据，接口返回调用成功。
binaryPayload	否	String	4096	推送的数据，由原始 byte 数组经过 base64 编码后生成的字符串，与 payload 二选一，不可都为空。
osType	否	String	10	指定消息推送的客户端操作系统类型，（iOS / Android），不限制操作系统类型时，留空。
appMinVersion	否	String	20	支持的最小客户端版本号【包含】，仅推送至大于等于该版本的客户端。例如：8.6.0.9999 ，强烈建议使用标准格式的版本号。
appMaxVersion	否	String	20	支持的最大客户端版本号【包含】，仅推送至小于等于该版本的客户端。如：9.0.0.9999 ，强烈建议使用标准格式的版本号。
validTimeEnd	否	long	13	推送有效期结束时间，过期后，MSS 服务端将不再推送过期数据至客户端。格式： (new Date()).getTime();。

返回参数

返回数据为 JOSN 格式，示例代码如下：

```
{
"msg":"SUCCESS",
"success": true
}
```

各属性含义及解释

名称	类型	示例	说明
success	boolean	true / false	业务调用结果，成功返回 true，失败返回 false。失败的情况下，可通过 msg 查看失败原因，参考下表 结果码 。
msg	String	SUCCESS	结果码。

结果码

结果	结果码	含义
true	SUCCESS	业务成功 - 在线推送成功
true	DUPLICATED_BIZ_ID	bizId 重复，业务成功
false	THIRDMSGID_IS_NULL	thirdMsgId 为空
false	BIZ_NOT_ONLINE	同步配置未提交上线
false	ARGS_IS_NULL	请求必选参数为空
false	NOT_SUPPORT_GLOBAL	接口不支持全局业务
false	PAYLOAD_LONG	消息体超长

false	THIRD_MSG_ID_LONG	第三方消息过长
false	SYSTEM_ERROR	系统异常

5.1.4 用户一致性验证

背景

某些业务场景下，业务对同步的数据有很高的用户安全性要求，此时，需要进行用户一致性验证。

- 目的
  在同步数据时，确保数据推送的目标用户即为客户端有效登录用户。
- 基本原理
  - 客户端连接到 MSS 服务端时，上传用户标识和授权 token（例如 sessionId）数据。
  - 服务端可调用一个由租户实现的一致性验证接口，通过此接口以及获取的用户标识和授权 token，由客户系统来控制是否一致。MSS 服务将记录一致性标识，用于后续同步数据时 session 状态的判断。
    说明：MSS 服务端判断 session 为有效时，将同步数据至相应用户；判断 session 为无效时，将不同步数据至相应用户。
  - 对于高安全要求的同步配置，可开启一致性验证，数据将仅推送通过用户一致性验证的设备上，对于未开启一致性验证的同步配置，在推送数据时，将会忽略一致性验证结果。同步配置操作请参见 管理控制台 。

配置一致性验证接口

在 mPaas 控制台上配置登录 session 校验的 RPC 接口。

操作入口

在选定 APP 的 **后台服务管理** > **移动网关** 页面上，配置 API，详细操作可参见 配置 API 。

接口名称

由于 MSS 服务端会固定调用 API，故创建的 API 的 **operationType** 必须为 **com.antcloud.session.validate**，并且请求参数也需固定如下：

名称	是否必填	数据类型	最大长度	说明
instanceId	是	String	100	业务场景标识，组成结构为 workspaceId_AppId
userId	是	String	100	用户唯一标识
sessionId	是	String	100	客户端携带的授权 token

返回结果

客户校验系统自定义一致性检验逻辑，需要返回的数据为 JOSN 格式。

代码示例

```
{
"response": {
"resultCode":"OK",
"resultMsg":"Operation is done successfuly",
"success":"true",
"result": {
"sid":"kkdddd",
"valid":"true/false",
}
}
}
```

各属性含义及解释

名称	类型	示例	说明
success	boolean	true / false	业务调用结果，成功返回 true，失败返回 false。失败的情况下，可通过 returnCode 查看失败原因，参考下表 结果码 。
returnCode	String	SUCCESS	结果码 。
resultMsg	String	SUCCESS	结果信息。
result	JSONString	{ “sid”: “kkdddd”, “valid”:” true/false”,}	结果对象，参数说明请见下表 结果对象参数 。

结果码

结果	结果码	含义
true	OK	业务成功
false	OPERATION_ERROR	OPERATION 错误， 只处理 com.antcloud.session.validate 接口

result 结果对象参数

名称	类型	示例	说明
sid	String	kkdddd	授权的 token 或 sessionId
valid	boolean	true / false	一致性验证结果

5.2 接入公有云到 MSS 服务端

5.2.1 接入简介

公有云租户业务系统接入移动同步服务 MSS（Mobile Sync Service）的服务器端的开发流程，分为 **接入**、**编写调用代码并处理结果** 和 **用户一致性验证** 三个部分。根据接入方式的不同，接入分为 **HTTP 接入** 和 **JAVA SDK 接入** 两种形式；根据采用接口的不同，编写调用代码并处理结果分为使用单数据同步接口和使用全局数据同步接口两种方式；用户一致性验证一般用于对同步数据有较高用户安全性要求的业务场景。

前置条件

- 您已经开通 mPaaS 产品，并且从租户管理员处得到 AccessKey 和 AccessSecret。
- 您已经创建好一个 App，了解客户端接入工作的进度，并且能够取到该 App 的 App ID 和 workspace ID。
- 您有一个服务器端应用。

接入

- HTTP 接入
- JAVA SDK 接入

编写调用代码并处理结果

- 使用单数据同步接口
- 使用全局数据同步接口

用户一致性验证

5.2.2 HTTP 接入

接入过程

- 组织 HTTP 请求参数。
- 计算签名并添加到 HTTP 请求参数中。
- 按要求发起 POST 请求。
- 获取 response 并处理。

HTTP 请求示例

请求地址

https://prodapigw.cloud.alipay.com/gateway.do

POST 数据示例

```
sign_type=HmacSHA1 // openapi 参数，签名算法
req_time=`current_time` // openapi 参数，请求时间
access_key=`your-access-key` // openapi 参数，ak
product_instance_id=`your-product-instance-id` // openapi 参数，产品实例ID
third_msg_id=`your-third-msg-id` // 接口业务参数，详见接口参数介绍
req_msg_id=`your-req-msg-id` // openapi 参数，请求ID
method=mpaas.msync.sync.syncdata // openapi 参数，请求的方法
version=1.0 // openapi 参数，接口版本
instance_id=`your-instance-id` // 接口业务参数，详见接口参数介绍
biz_type=`your-biz-type` // 接口业务参数，详见接口参数介绍
```

```
link_token=`your-link-token` // 接口业务参数，详见接口参数介绍
payload=`payload` // 接口业务参数，详见接口参数介绍
immediate_sync=true // 接口业务参数，详见接口参数介绍
sign=`your-sign` // openapi 参数，签名
```

返回数据示例

```
{
"result_code":"OK",
"result_msg":"success",
"bizResult": {
"returnCode":"2007",
"opLogId": 170405163602000001,
"success": true,
"returnReason":"NOT_ONLINE"
},
"req_msg_id":"19a493db1e244ae2b8af11f30172217a"
}
```

5.2.3 JAVA SDK 接入

接入过程

- 引入 jar 包
- 编写调用代码并处理结果。

引入 jar 包

引入 jar 包前，确认您完成了 Maven 的配置，参考 配置 Maven 。

完成 mvn 配置后，在主控 pom 中引入如下依赖：

```
<dependency>
<groupId>cn.com.antcloud.api</groupId>
<artifactId>antcloud-prod-api-sdk</artifactId>
<version>1.17.20170329</version>
</dependency>

<dependency>
<groupId>com.alipay.msync</groupId>
<artifactId>msync-common-service-facade</artifactId>
<version>1.0.0.20181017</version>
</dependency>
```

示例代码

```
// 初始化client
AntCloudProdClient client = AntCloudProdClient.newBuilder()
.setEndpoint("https://prodapigw.cloud.alipay.com/gateway.do") // 访问地址，固定不变
.setAccess("your-access-key","your-access-secret").build(); // TODO 替换为您的 access-key 和 access-secret。 可咨询
```

```
您的租户管理员

// 构造request
AntCloudProdClientRequest request = new AntCloudProdClientRequest();
request.setMethod("mpaas.msync.sync.syncdata"); // 设置访问的接口名称，可参考 API 文档
request.setVersion("1.0"); // 设置访问的接口版本
request.setProductInstanceId("73041097455087");

// 构造业务请求参数
SyncOrder syncOrder = new SyncOrder();
syncOrder.setInstanceId("default_99FB626081956"); // TODO 设置您的InstanceId，由 workspaceId_appId 构成，在 MPAAS 管控页面可查询到
syncOrder.setLinkToken("2088202921657332"); // TODO 设置您要推送的目标用户或者目标设备
syncOrder.setPayload(Base64.encodeToString("test content".getBytes(), true)); // 设置要发送的内容，需要对发送的内容做 base64 处理
syncOrder.setThirdMsgId("test_third_msg_id_19"); // 设置推送消息唯一ID，由您的业务系统决定，需要唯一
syncOrder.setBizType("UCHAT"); // TODO 设置推送的同步标识，参考 数据同步服务 管控端配置
syncOrder.setImmediateSync(true); // 设置是否立即发送，如果设置为 false 则只能通过拉取触达
//syncOrder.setAppMaxVersion("100.100.100.100"); // 设置接收消息的最大版本
//syncOrder.setAppMinVersion("0.0.0.0"); // 设置接收消息的最小版本
//syncOrder.setOsType("IOS"); // IOS/ANDROID // 设置接收消息的客户端操作系统平台，目前支持 iOS 和 Android。 注意，对应的值是大写字母的： IOS 和 ANDROID

request.putParametersFromObject(syncOrder);

try {
// 发起请求
AntCloudProdClientResponse response = client.execute(request);

if (response.isSuccess()
&& response.getData(OpenApiResult.class).getBizResult() != null) {
// 取到syncResult
SyncResult syncResult = JSON
.parseObject(((JSONObject) (response.getData(OpenApiResult.class)
.getBizResult())).toJSONString(), SyncResult.class);

// TODO 自行根据结果完成必要的逻辑
LOGGER.info("s:"+ syncResult.isSuccess() +", rc:"+ syncResult.getReturnCode()
+", rs:"+ syncResult.getReturnReason());
} else {
LOGGER.warn(response.getResultMsg());
}
} catch (InterruptedException e) {
LOGGER.error("process error!", e);
}
```

5.2.4 单数据同步接口

单数据同步接口是指同步数据到指定的用户或者设备。

基本参数信息

- method: mpaas.msync.sync.syncdata
- version: 1.0

业务参数信息

名称	类型	是否必须	示例	描述
app_max_version	String	否	10.9.0.20170302001	推送数据过滤客户端版本，小于等于该版本号的客户端发送。
app_min_version	String	否	1.0.0	推送数据过滤客户端版本，大于等于该版本号的客户端发送。
payload	String	是		推送的数据，对一个原始 byte 数组进行 base64 编码后的字符串，原始 byte 数组长度小于 24 * 1024。
third_msg_id	String	是	1760339273	一次数据同步请求 ID，同一个同步标识内唯一。ID 重复的请求将会被忽略。要求小于 100 Byte。
biz_type	String	是	UCHAT	管控平台配置的同步标识，参考 管控平台 。
device_id	String	否		限定设备：针对用户推送时，同时指定接收的设备。
immediate_sync	boolean	否	true/false	是否立即推送，默认是。 true - 入库成功后判断是否在线，在线立即推送，不在线下次拉取； false - 入库成功后，不判断是否在线，等下次拉取。
link_token	String	是		推送目标 ID，如果同步配置是基于设备推送，这里放入设备 id，如果是基于用户推送，这里放入用户 id。
os_type	String	否	IOS/ANDROID	按手机平台过滤进行推送。默认情况下不传递参数，即不过滤、IOS和ANDROID手机平台都推送。
instance_id	String	是		workspaceId_appId 的字符串。
start_date	String	否	2017-04-01 12:00:00	当前时间 大于等于 start_date 才推送。
expire_date	String	否	2017-04-23 12:00:00	当前时间 小于等于 expire_date 才推送。

返回参数

返回数据格式为 JSON 格式，示例如下：

```
{
"response": {
"result_code":"OK",
"result_msg":"Operation is done successfuly",
"req_msg_id":"请求中的req_msg_id",
"bizResult": {
"success":"true",
"returnCode":"success",
"returnReason":"success",
"opLogId":"14798722309"
}
},
"sign":"签名"
}
```

各属性含义解释：

名称	类型	示例	描述
success	boolean	true/false	业务调用是否成功，成功返回 true，失败返回 false。失败的情况下，通过 returnCode 查看失败原因，可参考下表业务结果码。
returnCode	String	ERROR	结果码。
returnReason	String	SYSTEM-ERROR	结果信息。
opLogId	Long 型数字	14798722309	MSS 数据唯一标示 ID。

业务结果码

结果	结果码	含义
SUCCESS	OK	业务成功 - 在线推送成功
SUCCESS	DUPLICATED_BIZ_ID	bizId 重复，业务成功
SUCCESS	NOT_ONLINE	用户/设备不在线
ERROR	THIRDMSGID_IS_NULL	thirdMsgId 为空
ERROR	BIZ_NOT_ONLINE	同步配置未提交上线
ERROR	ARGS_IS_NULL	请求必选参数为空
ERROR	NOT_SUPPORT_GLOBAL	接口不支持全局业务
ERROR	PAYLOAD_LONG	消息体超长
ERROR	THIRD_MSG_ID_LONG	第三方消息 Id 过长
ERROR	SYSTEM_ERROR	系统错误

5.2.5 全局数据同步接口

全局数据同步是指同步数据到所有设备。

基本信息

- method: mpaas.msync.sync.syncglobal
- version: 1.0

业务参数

名称	类型及长度要求	是否必须	示例	描述
app_max_version	String	否	10.9.0.20170302001	推送数据过滤客户端版本，小于等于该版本号的客户端发送。
app_min_version	String	否	1.0.0	推送数据过滤客户端版本，大于等于该版本号的客户端发送。
payload	String	是		推送的数据，对一个原始 byte 数组进行 base64 编码后的字符串，原始 byte 数组长度小于 24 * 1024。
third_msg_id	String	是	1760339273	一次数据同步请求 ID，同一个同步标识内唯一。ID 重复的请求将会被忽略。要求小于 100 Byte。

biz_type	String	是	UCHAT	管控平台配置的同步标识 ，参考 管控平台 。
immediate_sync	boolean	否	true/false	是否立即推送，默认是。 true - 入库成功后判断是否在线，在线立即推送，不在线下次拉取； false - 入库成功后，不判断是否在线，等下次拉取。
os_type	String	否	IOS/ANDROID	推送按手机平台过滤，默认全部。
instance_id	String	是		workspaceId_appId 的字符串。
start_date	String	否	2017-04-01 12:00:00	当前时间 大于等于 start_date 才推送。
expire_date	String	否	2017-04-23 12:00:00	当前时间 小于等于 expire_date 才推送。

返回参数

返回数据格式为 JSON 格式，示例如下：

```
{
"response": {
"result_code":"OK",
"result_msg":"Operation is done successfuly",
"req_msg_id":"请求中的req_msg_id",
"bizResult": {
"success":"true",
"returnCode":"success",
"returnReason":"success",
"opLogId":"14798722309"
}
},
"sign":"签名"
}
```

各属性含义解释：

名称	类型	示例	描述
success	boolean	true/false	业务调用是否成功，成功返回 true，失败返回 false。失败的情况下，通过 returnCode 查看失败原因，可参考下表业务结果码。
returnCode	String	ERROR	结果码。
returnReason	String	SYSTEM-ERROR	结果信息。

业务结果码

结果	结果码	含义
SUCCESS	OK	业务成功
SUCCESS	DUPLICATED_BIZ_ID	bizId 重复，业务成功
SUCCESS	NOT_ONLINE	用户/设备不在线
ERROR	INVALID_SIGNATURE	产品签名验证失败
ERROR	THIRDMSGID_IS_NULL	thirdMsgId 为空

ERROR	BIZ_NOT_ONLINE	同步配置未提交上线
ERROR	ARGS_IS_NULL	请求必选参数为空
ERROR	NOT_SUPPORT_GLOBAL	接口不支持全局业务
ERROR	PAYLOAD_LONG	消息体超长
ERROR	THIRD_MSG_ID_LONG	第三方消息 Id 过长
EROR	GLOBAL_OPLOG_LINKTOKEN_NOT_EMPTY	全局推送时 LinkToken 必须为空
ERROR	SYSTEM_ERROR	系统错误

5.2.6 用户一致性验证

背景

- 在某些业务场景下，业务对同步的数据有很高的安全性要求，为了确保推送的目标用户就是当前登录的用户而没有被伪造。
- 数据同步服务提供了这样的高级功能，供用户开启使用。其基本原理是：
    - 客户端在连接到服务器端时，上报了用户标示和授权 token（比如 sessionId）数据。
    - 服务器端可以调用一个由租户实现的一致性验证接口，通过这个接口，由租户来控制是否一致。数据同步服务会记录下是否一致的标示。
    - 对于高安全要求的同步配置，租户可以开启一致性验证，数据只会推送到通过一致性验证的用户设备上。对于未开启一致性验证的同步配置，则会忽略一致性验证结果。

配置一致性验证接口

操作入口

在选定 APP 的 **后台服务管理**-> **移动网关** 页面，添加API，详细的添加帮助可以参考 移动网关-API管理 。

接口名称

添加 API 的 operationType 必须为 com.antcloud.session.validate。请求参数配置如下：

名称	类型及长度要求	是否必须	示例	描述
instanceId	String	是		workspaceId_appId 的字符串
userId	String	是	20880939	用户 ID
sessionId	String	是		客户端携带的授权token

返回参数

实现一个一致性检验逻辑，需要返回数据格式为 JSON 格式，示例如下：

```
{
"response": {
"resultCode":"OK",
```

```
"resultMsg":"Operation is done successfuly",
"success":"true",
"result": {
"sid":"kkdddd",
"valid":"true/false",
}
}
}
```

各属性含义解释：

名称	类型	示例	描述
success	boolean	true/false	业务调用是否成功，成功返回 true，失败返回 false。失败的情况下，通过 returnCode 查看失败原因，可参考下表业务结果码。
returnCode	String	ERROR	结果码。
resultMsg	String	SYSTEM-ERROR	结果信息。
sid	String	kkdddd	授权 token，或者 sessionId。
valid	boolean	true/false	验证结果。

业务结果码

结果	结果码	含义
true	OK	业务成功
false	OPERATION_ERROR	OPERATION 错误， 只处理 com.antcloud.session.validate 接口

6 管理控制台

6.1 控制台简介

数据同步控制台提供管理推送配置及执行业务数据推送的功能。

添加推送配置，相当于定义了一个具体的数据推送的业务场景。业务数据推送就是实现该配置所对应的业务场景。

通过数据同步控制台，您可以进行以下操作：

- 新增配置
- 发送业务数据
- 查看配置详情
- 修改配置
- 上/下线配置
- 查询配置推送的统计数据
- 服务管理

前置条件

您已开通 mPaaS 产品。

6.2 新增配置

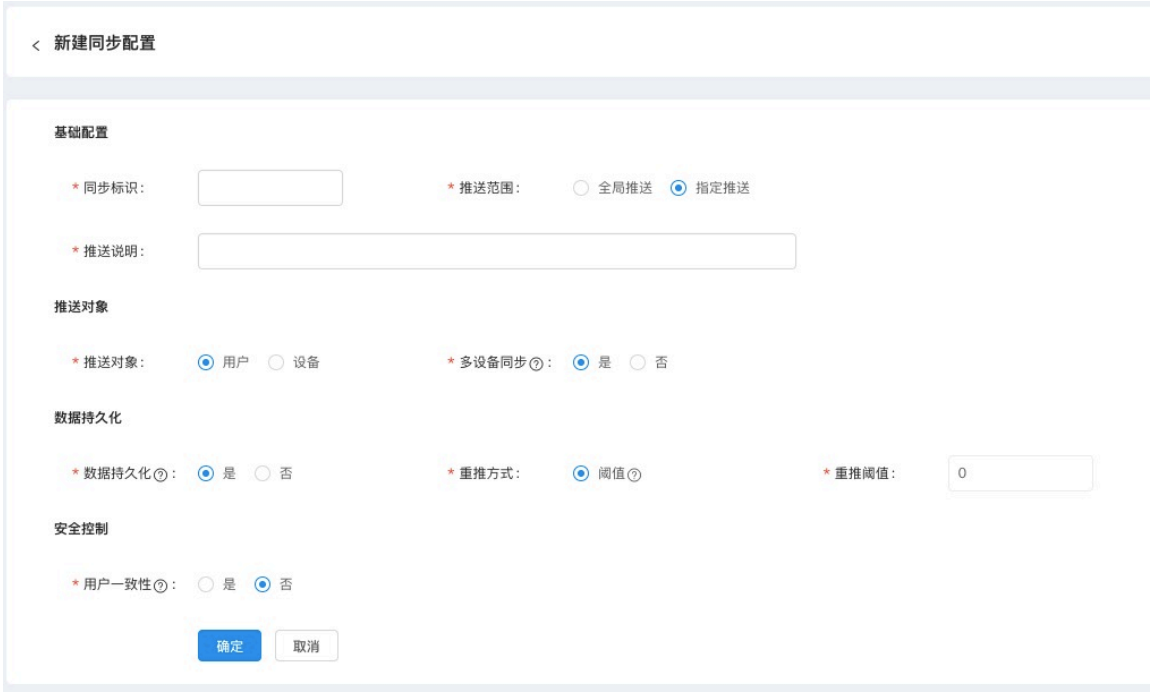
进入 **移动开发平台mPaaS** 控制台，完成以下步骤以新增配置。

操作步骤

1. 点击需要添加配置的mPaaS应用。如果当前没有应用，可以点击页面底部 创建mPaaS应用 按钮，填写表单创建新的mPaaS应用。

2. 在左侧导航栏点击 **后台服务管理** > **数据同步**，进入 **数据同步控制台**。
3. 在 **配置管理** 标签页下，点击 **添加** 按钮，进入 **新建同步配置** 页面。

4. 在该页面上填写各个配置项信息。



各配置项的说明如下表所示。

配置项	说明
同步标识	用来标识一种具体的数据推送业务场景，建议使用大写英文字母加字符“-”的格式，如 DEVICE-LOCK。
推送说明	可以填写一些备注信息，用来说明该配置对应的具体业务场景。
推送范围	用来表示数据推送流程中可接收数据的用户或者设备的范围。全局推送表示所有用户或者设备都可以收到；指定推送表示仅指定的某个用户或者设备可以收到。
推送对象	表示数据推送是针对用户还是设备的。
多设备同步	仅在推送对象为用户时需要选择。如果选择为是，则表示支持单个用户的多个设备之间的数据同步，即同一个用户在切换设备的情况下仍然会收到在上一个设备上已经收到过的数据。
数据持久化	表示推送的数据会被保存到数据库中，默认最长期限为 30 天，这样数据就不会丢失，如果当时推送数据时用户不在线，当该用户下一次在线时就会收到。
重推方式	用来指定当数据积压在服务端时的处理策略，仅在数据持久化配置为是的情况下有效。阈值推送是指仅推送给客户端阈值所指定数量的服务端最新积压数据。
重推阈值	仅在数据持久化配置为“是”且重推方式配置为“阈值”的情况下有效。
用户一致性	仅当推送对象设置为用户时有效，当设置为“是”时，移动同步服务会在推送数据时验证用户一致性，满足一致性要求时推送，否则本次不推送。详见 用户一致性验证（ 专有云 ， 公有云 ）。

6.3 发送业务数据

进入 **移动开发平台mPaaS** 控制台，完成以下步骤以新增配置。

前提条件

控制台中已有一条推送配置并处于上线状态。

操作步骤

1. 点击需发送业务数据的mPaaS应用。
2. 在左侧导航栏点击 **后台服务管理** > **数据同步**，进入 **数据同步控制台**。
3. 在 **配置管理** 标签页下，点击配置列表中某条配置右侧的 **操作** 按钮，进入 **新建同步** 配置页面。

4. 填写对话框中的各个配置项信息，然后点击 **确定** 按钮进行发送。

各配置项的说明如下表所示。

配置项	说明
用户/设备 ID	仅针对指定用户或者指定设备类型的业务需要填写。
数据内容	数据的文本内容，字符串格式。
数据唯一 ID	仅做数据持久化的业务需要填写，用来标识一个唯一的数据内容，当有两次数据唯一 ID 重复的推送时，后一条推送会被忽略掉。
系统	表示需要接收数据的客户端操作系统类型，默认 Android 和 iOS。
版本区间	表示需要接收数据的客户端的应用版本区间，选填。
有效期	表示本次推送的数据的最长有效期，默认 30 天。

6.4 查看配置详情

进入 **移动开发平台 mPaaS** 控制台，完成以下步骤以查看配置详情。

操作步骤

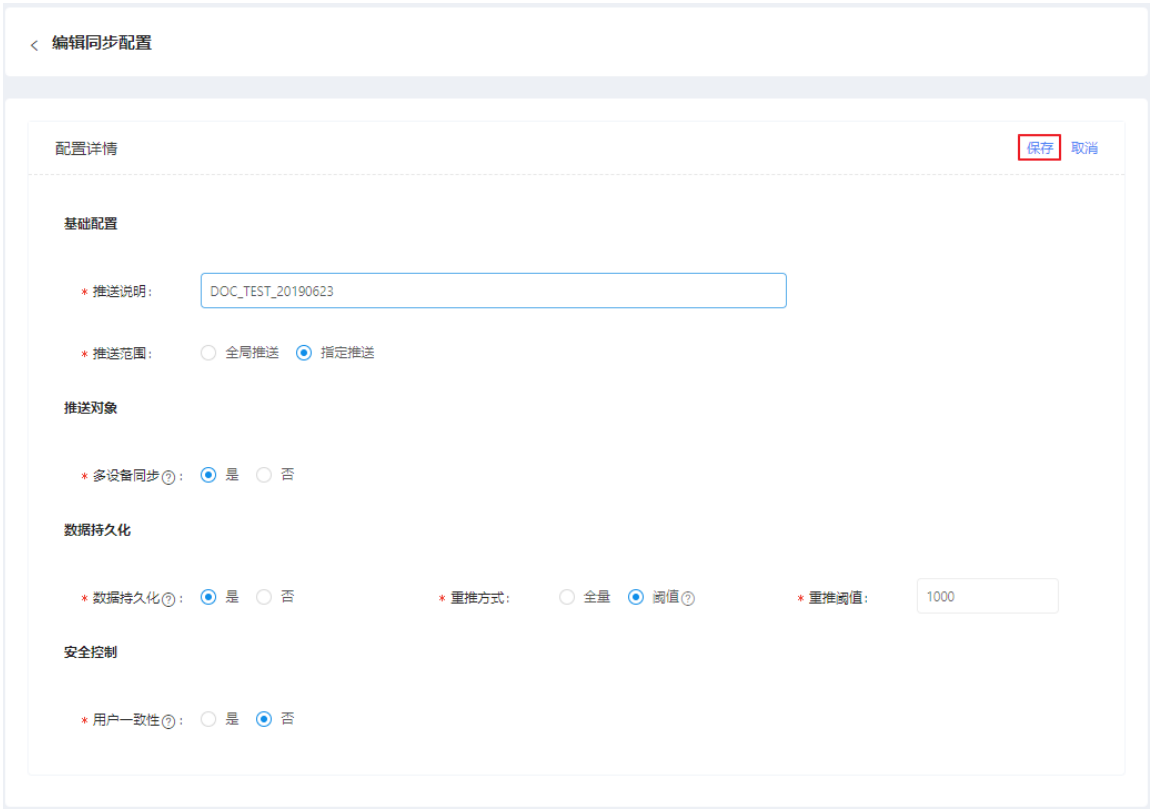
1. 点击相应的的mPaaS应用。
2. 在左侧导航栏点击 **后台服务管理** > **数据同步**，进入 **数据同步控制台**。
3. 进入 **配置管理** 标签页，点击配置列表中某条配置的标识，即可查看该配置的详情。

6.5 修改配置

进入 **移动开发平台 mPaaS** 控制台，完成以下步骤以查看配置详情。

操作步骤

1. 点击相应的的mPaaS应用。
2. 在左侧导航栏点击 **后台服务管理** > **数据同步**，进入 **数据同步控制台**。
3. 进入 **配置管理** 标签页，点击配置列表中某条配置的标识，进入该配置详情页。
4. 点击页面右上方的 **修改** 按钮，修改表单后，点击 **保存** 即可。



注意：**同步标识** 和 **推送对象** 不允许被修改。

6.6 上/下线配置

进入 **移动开发平台 mPaaS** 控制台，完成以下步骤以查看配置详情。

操作步骤

1. 点击相应的的mPaaS应用。
2. 在左侧导航栏点击 **后台服务管理** > **数据同步**，进入 **数据同步控制台**。
3. 点击配置列表中某条配置右侧的 **下线/上线** 按钮，然后在弹出的确认对话框中点击 **确定**，即可将该配置下线或上线。

业务上线后就可以通过接口调用或者控制台操作的方式进行正常的数据推送。
业务下线就表示该业务暂时被禁用，如果需要再次使用，可以进行上线操作。

6.7 查询配置推送的统计数据

MSS 提供了三种配置推送的统计数据： 基础指标 、 BizType汇总 和 用户/设备状态查询 。进入 **移动开发平台 mPaaS** 控制台，完成以下步骤以查看各配置的推送统计数据。

操作步骤

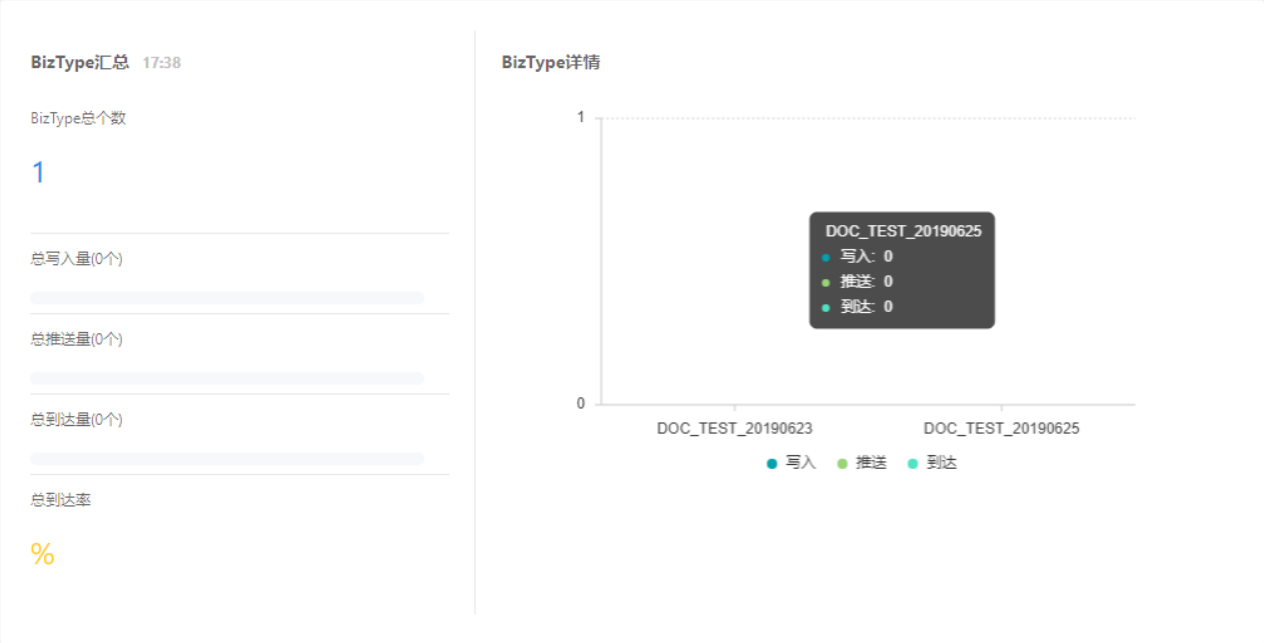
1. 点击需要添加配置的mPaaS应用。
2. 在左侧导航栏点击 **后台服务管理** > **数据同步**，进入 **数据同步控制台**。
3. 进入 **数据统计** 标签页。在此页面，即可查看各配置的推送统计数据

基础指标

MSS在数据统计页面提供了业务类型的统计数据，即BizType汇总。这些统计数据包括BizType总个数、总写入量、总推送量、总到达量和总到达率。
根据以上数据，MSS提供了柱状图表以直观显示写入、推送、和到达等数据。

BizType汇总

MSS在数据统计页面提供了业务类型的统计数据，即BizType汇总。这些统计数据包括BizType总个数、总写入量、总推送量、总到达量和总到达率。
根据以上数据，MSS提供了柱状图表以直观显示写入、推送、和到达等数据。

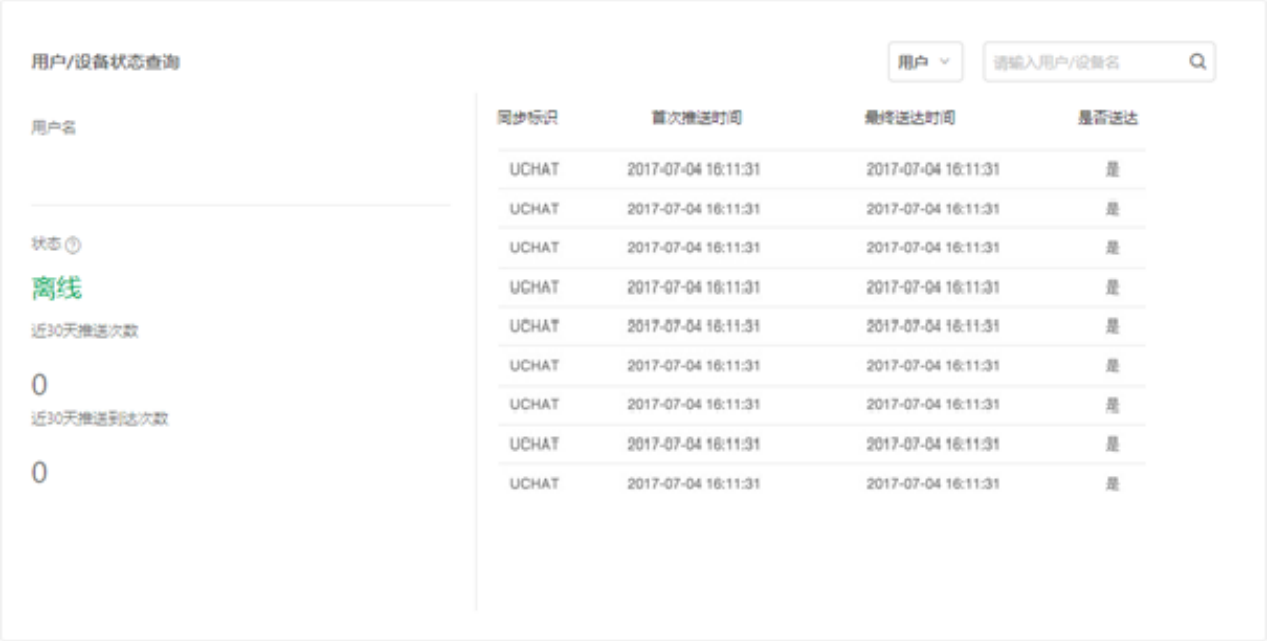


用户/设备状态查询

在数据统计页面，您也可以在 **用户/设备状态查询** 区域的右上方选择 **用户** 或 **设备**，在搜索框中输入 用户名 或 设备名，可以以查看特定用户或设备的状态。
MSS在此页面为用户或设备提供了以下数据：

- 用户名 / 设备名
- 状态
- 近30天推送次数
- 近30天推送到达次数
- 推送列表

>



6.8 服务管理

在服务管理标签页，提供了签名校验的开关。该功能开关全局有效，可以根据需要暂时地开启或者关闭所有签名校验相关功能。

7 API 接口说明

7.1 Android 接口说明

7.1.1 版本 ≥ 10.1.32

```
java.lang.Object
- com.mpaas.mss.adapter.api.MPSync
```

- 概览
- 公共函数 appToBackground()appToForeground()clearUserInfo()initialize(Context context)isConnected()registerBiz(String bizType, ISyncCallback syncCallback)reportMsgReceived(SyncMessage syncMessag)unregisterBiz(String bizType)updateUserInfo(String sessionId)

概览

在 10.1.32 及以后的基线版本中，mPaaS 中间层的 MPSync 类封装了移动同步组件所有 API。通过 MPSync 对象即可实现移动同步的所有功能。

公共函数

公共函数	
void	appToBackground() 用于让客户端 SDK 感知到当前 App 已经回到后台，使其断开与服务器的网络连接。每次 App 压后台时调用。
void	appToForeground() 用于让客户端 SDK 感知到当前 App 已经启动，使其建立与服务器的网络连接。每次 App 回前台时调用。
void	clearUserInfo() 用于用户登出。
void	initialize(Context context) 初始化接口，初始化移动同步服务。
boolean	isConnected() 用于检查当前移动同步服务是否正常。
void	registerBiz(String bizType, ISyncCallback syncCallback) 用于注册一个接收业务数据的 callback。在获取到同步推送的数据后，客户端 SDK 会回调 syncCallback实现类。
void	reportMsgReceived(SyncMessage syncMessag) 用于在 syncCallback 实现类中收到数据后，调用该接口通知移动同步服务端接收同步数据成功。在没有收到 reportMsgReceived 前，移动同步服务会重试投递，重试 6 次之后数据会被永久删除。
void	unregisterBiz(String bizType) 用于反注册指定同步配置。在获取到同步推送的数据后，客户端 SDK 则不会回调 syncCallback 实现类。
boolean	updateUserInfo(String sessionId) 用于登录信息 userId/sessionId 有变化时调用。

appToBackground()

声明

public static void appToBackground()

说明

用于让客户端 SDK 感知到当前 App 已经回到后台，使其断开与服务器的网络连接。每次 App 压后台时调用。

建议在首页的 onStop() 方法内调用。

如果压后台不调用此 API， 将会导致长时间网络连接，带来耗电量、流量增加的问题。

参数

无。

返回值

无。

appToForeground()

声明

public static void appToForeground()

说明

用于让客户端 SDK 感知到当前 App 已经启动，使其建立与服务器的网络连接。每次 App 回前台时调用。

建议在首页的 onResume() 方法内调用。

参数

无。

返回值

无。

clearUserInfo()

声明

public static void clearUserInfo()

说明

用于用户登出。

参数

无。

返回值

无。

initialize(Context context)

声明

public static void initialize(Context ctx)

说明

初始化接口，初始化移动同步服务。如果不调用，将导致当前 App 不能使用本服务。

全局仅需调用一次（App 打开到关闭的生命周期内只需要调用一次）。

参数

ctx	Context：一个不为空的Context。

返回值

无。

isConnected()

声明

public static boolean isConnected()

说明

检查当前移动同步服务是否正常。

参数

无。

返回值

正常返回 true。

不正常返回 false。

registerBiz(String bizType, ISyncCallback syncCallback)

声明

public static void registerBiz(String biz, ISyncCallback callback)

说明

用于注册一个接收业务数据的?callback。在获取到同步推送的数据后，客户端 SDK 会回调?syncCallback实现

类。
每个同步配置都需调用一次该 API。

参数

bizType	String：同步标识
syncCallback	ISyncCallback：回调实现类

返回值

无。

reportMsgReceived(SyncMessage syncMessag)

声明

public static void reportMsgReceived(SyncMessage msg)

说明

用于在 syncCallback 中收到同步推送的数据后，调用该接口通知移动同步服务端接收同步数据成功。
在没有收到 reportMsgReceived 前，移动同步服务端会重试投递，重试 6 次之后数据就被永久删除。

参数

syncMessag	SyncMessage：同步消息

返回值

无。

unregisterBiz(String bizType)

声明

public static void unregisterBiz(String biz)

说明

反注册指定同步配置。
移动同步服务在收到该同步配置的数据后，不会调用 syncCallback。

参数

biz	String：同步标识

返回值

无。

updateUserInfo(String sessionId)

声明

public static boolean updateUserInfo(String sessionId)

说明

方法内部的调用基于 LongLinkSyncService.getInstance().updateUserInfo(String userId, String sessionId)接口，其中 userId 使用的是在 MPLogger 中设置的用户 ID。
用于在登录信息 userId/sessionId 有变化时调用，以更新用户登录信息。

登录时，两个参数都不能为空，如果 userId 未设置，该方法会返回 false，调用失败。
如果 session 过期，或者是客户端在用户登录过一次之后具备了自动免登的功能，那么每次免登成功时也必须要调用本方法。总体调用原则是: userId 与 sessionId 两个参数任意一个发生变化时都必须要调用本方法。

参数

sessionId	String：会话 ID。

返回值

更新用户信息成功则返回 true。
如果登录时 userId 未设置返回 false。

7.1.2 版本 < 10.1.32

LongLinkSyncService 提供了移动同步组件所有 API，通过 LongLinkSyncService.getInstance() 获取到 LongLinkSyncService 的实例。

initialize(Context context)

初始化接口，初始化移动同步服务。如果不调用，将导致当前 App 不能使用本服务。
全局仅需调用一次（App 打开到关闭的生命周期内只需要调用一次）。
必须传入一个不为空的 Context 。

appToForeground()

每次 App 回前台时调用。
建议在首页的 onResume() 方法内调用。
调用这个方法可以让移动同步服务感知到当前 App 已经启动，移动同步服务将会建立与服务器的网络连接。

appToBackground()

每次 App 压后台时调用。
建议在首页的 onStop() 方法内调用。
调用这个方法可以让移动同步服务感知到当前 App 已经回到后台，移动同步服务将断开与服务器的网络连接。
如果压后台不调用此 API， 将会导致长时间网络连接，带来耗电量、流量增加的问题。

updateUserInfo(String userId, String sessionId)

用于登录信息 userId/sessionId 有变化时调用。
登录时，两个参数都不能为空。
如果是登出，两个值都传 null 或者 空字符串 即可，如：updateUserInfo(null, null)。如果 session 过期，或者是客户端在用户登录过一次之后具备了自动免登的功能，那么每次免登成功时也必须要调用本方法。
总体调用原则是: userId 与 sessionId 两个参数任意一个发生变化时都必须要调用本方法。

registerBiz(String bizType, ISyncCallback syncCallback)

该接口注册一个用于接收业务数据的 callback，当有数据之后，移动同步服务回调 syncCallback 。
代码实现请参照后面一节的 Demo 代码 。
每个同步配置都需调用一次这个 API 。

unregisterBiz(String bizType)

反注册指定同步配置。
移动同步服务在收到该同步配置的数据后，不会调用 syncCallback。

reportMsgReceived(SyncMessage syncMessag)

在 syncCallback 中收到数据后，调用该接口通知移动同步服务接收成功。
在没有收到 reportMsgReceived 前，移动同步服务会重试投递，重试 6 次之后数据就被永久删除。

isConnected()

检查当前移动同步服务是否正常。
正常返回 true。
不正常返回 false。

7.2 iOS 接口说明

7.2.1 版本 ≥ 10.1.32

MPMssAdapter.framework 中 MPSyncInterface 这个类提供了移动同步服务所有 API 接口， 里面所有的方法都是类方法，可以直接类名调用方法。

+(void)initSync;

初始化接口，初始化移动同步服务。如果不调用，将导致当前 App 不能使用本服务。
全局仅需调用一次（App 打开到关闭的生命周期内只需要调用一次）。

+(MPSyncNetConnectType)connectStatus;

查当前移动同步服务连接情况。
返回连接状态 MPSyncNetConnectType。

+(BOOL)registerSyncBizWithName:(NSString *)bizName syncObserver:(id)observer selector:(SEL)selector;

注册对业务名称为 bizName 的通知监听，内部调用了 [[NSNotificationCenter defaultCenter] addObserver:observer selector:selector name:bizName object:nil];进行通知监听。
bizName 与服务端控制台配置项对应
如果不调用该接口则不分发该 biz 消息，消息会积压在客户端 SDK 的数据库。
需要监听同步服务端消息的同步标识最好启动时候开始监听。
返回注册结果 YES/NO。

+(BOOL)unRegisterSyncBizWithName:(NSString *)bizName syncObserver:(id)observer;

通知移动同步服务客户端 SDK 已经取消某同步配置的消息监听，不再接收该同步配置的
Sync 消息，内部调用了 [[NSNotificationCenter defaultCenter] removeObserver:observer name:bizName object:nil];进行监听移除。
调用该接口后不会再分发该 biz 的消息，消息会积压在 SyncSDK 的数据库，与 registerSyncBizWithName 接口对应。
返回结果 YES/NO。

+(void)removeSyncNotificationObserver:(id)observer;

取消 Sync 的通知监听，通常在监听类的 dealloc 函数中调用，内部调用了[[NSNotificationCenter defaultCenter] removeObserver:observer]; 进行监听者移除。
无返回值。

+(void)responseMessageNotify:(NSDictionary *)userInfo;

消息处理完成通知回调（callback），参数是通知里面的 userInfo(notify.userInfo)。
回调 SyncSDK，表示业务数据已经处理在 registerSyncBizWithName 接口注册的通知处理函数中，数据处理完后调用。
无返回值。

+(void)bindUserWithSessionId:(NSString *)sessionId;

用于登录信息 userId 或 sessionId 有变化时调用。
登录时调用，采用的 userId 为 MPaaSInterface 的 -(NSString*)userId函数。
如果 sessionId 过期， 或者是客户端在用户登录过一次之后具备了自动免登的功能，那么每次免登成功时也必须要调用本方法。
总体调用原则为：userId 或 sessionId 任意一个发生变化时都必须要调用本方法。
当 userId 发生变化时，先调用 unBindUser 解绑，然后调用bindUserWithSessionId: 重新建连。
sessionId 用于校验 session 合法性，需要服务端配合。如果设为 nil，则默认为@”SESSION_DEMO”。
无返回值。

+(void)unBindUser;

用户登出时候调用，解绑当前连接用户。
无返回值。

+(NSString *)getSyncDeviceId;

获取设备 Id，根据设备维度推 Sync 数据时采用此 Id。
返回设备 Id。
重要：当接口中 SessionId 设置为无效值时，控制台的用户一致性选项必须处于关闭状态，否则校验不过无法成功推送 Sync。请参考 服务管理 开启或关闭签名校验。

7.2.2 版本 < 10.1.32

APLongLinkService.framework 中 DTLongLinkBusiness 这个类提供了移动同步服务所有 API 接口， 里面所有的方法都是类方法，可以直接类名调用方法。

+(void)syncInit;

初始化接口，初始化移动同步服务。如果不调用，将导致当前 App 不能使用本服务。
全局仅需调用一次（App 打开到关闭的生命周期内只需要调用一次）。

+(NetConnectType)connectStatus;

查当前移动同步服务连接情况。
返回连接状态 NetConnectType。

+(BOOL)hasRegisterNotificationWithBiz:(NSString*)biz;

注册完系统监听器后，通知移动同步服务客户端 SDK，已经完成消息的监听，可以接收
Sync 消息。
如果不调用该接口则不分发该 biz 消息，消息会积压在客户端 SDK 的数据库。
需要监听同步服务端消息的同步标识最好启动时候开始监听。
返回注册结果 YES/NO。

+(BOOL)unRegisterNotificationWithBiz:(NSString*)biz;

通知移动同步服务客户端 SDK 已经取消某同步配置的消息监听，不再接收该同步配置的Sync 消息。
调用该接口后不会再分发该 biz 的消息，消息会积压在 SyncSDK 的数据库，与 asRegisterNotificationWithBiz 接口对应。
返回结果 YES/NO。

+(void)responseMessageNotify:(NSDictionary *)userInfo;

消息处理完成通知回调（callback），参数是通知里面的 userInfo(notify.userInfo)。
无返回值。

+(void)sendBindUser:(NSString *)userId sessionId:(NSString *)sessionId;

用于登录信息 userId 或 sessionId 有变化时调用。
登录时调用，两个参数都不能为空。
如果 sessionId 过期， 或者是客户端在用户登录过一次之后具备了自动免登的功能，那么每次免登成功时也必须要调用本方法。
总体调用原则为：userId 或 sessionId 任意一个发生变化时都必须要调用本方法。
无返回值。

+(void)sendUnBindUser;

用户登出时候调用，解绑当前连接用户。
无返回值。


蚂蚁金服
ANT FINANCIAL
金融科技
FINANCIAL TECHNOLOGY