蚂蚁科技

数据同步
使用指南

文档版本：20230728

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

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.变更记录

文档版本	变更内容
V20210630	新增 查询数据同步历史记录 章节，提供同步操作记录查看指导。

2.数据同步简介

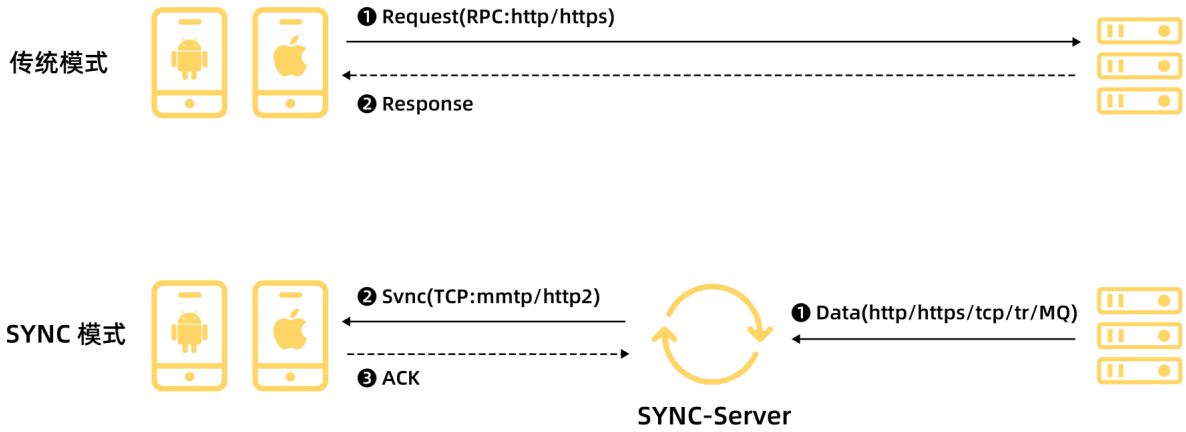
数据同步是 mPaaS 平台的一个核心基础服务组件。数据同步源自蚂蚁集团内面向移动应用、从服务端到客户端进行海量数据推送的全链路解决方案 — SYNC。该组件提供了一个安全的基于传输控制协议（Transmission Control Protocol，简称 TCP）和安全套接层（Secure Sockets Layer，简称 SSL）的数据通道，能够及时、准确、有序地将服务器端的业务数据主动地同步（SYNC）到客户端 App。

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带来不小的损耗。

为改善或解决上述问题，数据同步应运而生。

SYNC 数据同步



功能特性

数据同步具备以下特性：

- 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，数据同步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

  数据同步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数据同步服务器的顺序一致，并且所有消息以增量方式同步至客户端。

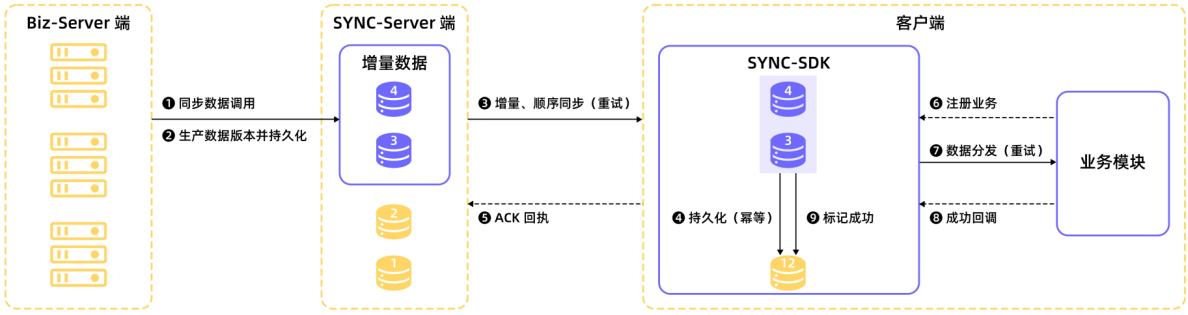
- 高实时性

  当客户端连接的网络状况良好时，数据同步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）。

基本原理

类似于 MySQL 的 binlog 机制，数据同步服务器和客户端 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，业务调用数据同步接口，数据同步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，然后当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）。数据同步服务端会按照 oplog id 从小到大的顺序把每一条 oplog 都推送至客户端。数据同步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）。

SYNC 核心 —— OpLog 同步



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

数据同步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数据同步的能力有更深入了解。

- 在即时通讯 App 中，数据同步提供增量、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，数据同步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，数据同步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。同时数据同步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。

3.基本概念

本术语表按拼音首字母对术语进行排序。

B

BizType

为业务类型，是业务场景的唯一标识。数据推送后，客户端数据同步 SDK 需要通过 Biztype 将数据分发给对应的业务模块。

C

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。在数据同步服务中，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：当用户或设备不在线时，数据将持久化至数据库，待用户或设备上线后，数据同步 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。

D

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。

H

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）。

M

幂等

根据 SyncOrder 中的 thirdMsgId 字段进行去重（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

指需要通过数据同步服务端推送的数据。

MSS 推送

指将一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。

Q

前台

指客户端 App 当前处于打开的状态。

S

SYNC

指数据同步服务，是指将数据从数据同步服务端同步至客户端 App。

T

推送类型

分为指定推送和全局推送两种类型。

- 指定推送：指定某一个 userId 或 utdId，推送一条数据。
- 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。

Y

业务维度

同步的业务维度分为用户维度和设备维度，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。

Z

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 在后台时维持三分钟在线（iOS 操作系统性限制）。

4.接入客户端

4.1. 接入 Android

> 说明
>
> 自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68 升级指南 或 mPaaS 10.1.60 升级指南 进行基线版本升级。

本文介绍如何快速将数据同步组件接入到 Android 客户端。目前，数据同步组件支持 **原生 AAR 接入** 和 **组件化接入** 两种接入方式。

接入过程分为两步：

1. 添加 SDK
2. 使用 SDK

前置条件

您已接入工程到 mPaaS。

- 若采用原生 AAR 方式接入，需要先 将 mPaaS 添加到您的项目中。
- 若采用组件化方式接入，需要先完成 组件化接入流程。

添加 SDK

原生 AAR 方式

参考 AAR 组件管理，通过 **组件管理（AAR）** 在工程中安装 **同步服务（SYNC）**组件。

组件化方式

在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **同步服务（SYNC）** 组件。

更多信息，请参考 管理组件依赖 > 增删组件依赖。

使用 SDK

在 10.1.32 及以上版本基线中，mPaaS 中间层的 `MPSync` 类封装了数据同步组件所有 API，通过调用 `MPSync` 对象即可实现数据同步的所有功能。

您可以通过下表快速了解数据同步的相关 API。更多关于 API 的详细信息，参见 Android 接口说明。

接口	接口说明
setup(Application application)	用于初始化数据同步依赖的基础服务。必须在 `initialize` 方法调用前调用。仅限 10.1.60 及以上版本基线。

initialize(Context context)	用于初始化接口和数据同步服务。
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。至少调用一次。
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，调用该接口通知数据同步服务端接收同步数据成功。在没有收到 `reportMsgReceived` 前，数据同步会重试投递，重试 6 次之后数据会被永久删除。
isConnected()	用于检查当前数据同步服务是否正常。

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数据同步 API 的方法，并不能作为数据同步的最佳实践。您可以在 获取代码示例 页面中下载数据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 打印数据同步数据。
    Log.e("==========",utdid);
    //初始化接口和数据同步服务。
    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 打印数据同步数据。
        Log.e("==========",syncMessage.msgData);
        //通知数据同步服务端接收同步数据成功。
        MPSync.reportMsgReceived(syncMessage);
    }
}
```

后续操作

接入服务端

4.2. 接入 iOS

4.2.1. 添加 SDK

本文介绍如何快速将移动同步服务组件接入到 iOS 客户端。

移动同步服务支持基于 mPaaS 框架接入、基于已有工程且使用 mPaaS 插件接入以及基于已有工程且使用 CocoaPods 接入三种接入方式。您可以参考 接入方式介绍，根据实际业务情况选择合适的接入方式。

前置条件

您已接入工程到 mPaaS。更多信息，请参见以下内容：

- 基于 mPaaS 框架接入
- 基于已有工程且使用 mPaaS 插件接入
- 基于已有工程且使用 CocoaPods 接入

添加 SDK

根据您采用的接入方式，请选择相应的添加方式。

使用 mPaaS Xcode Extension 插件

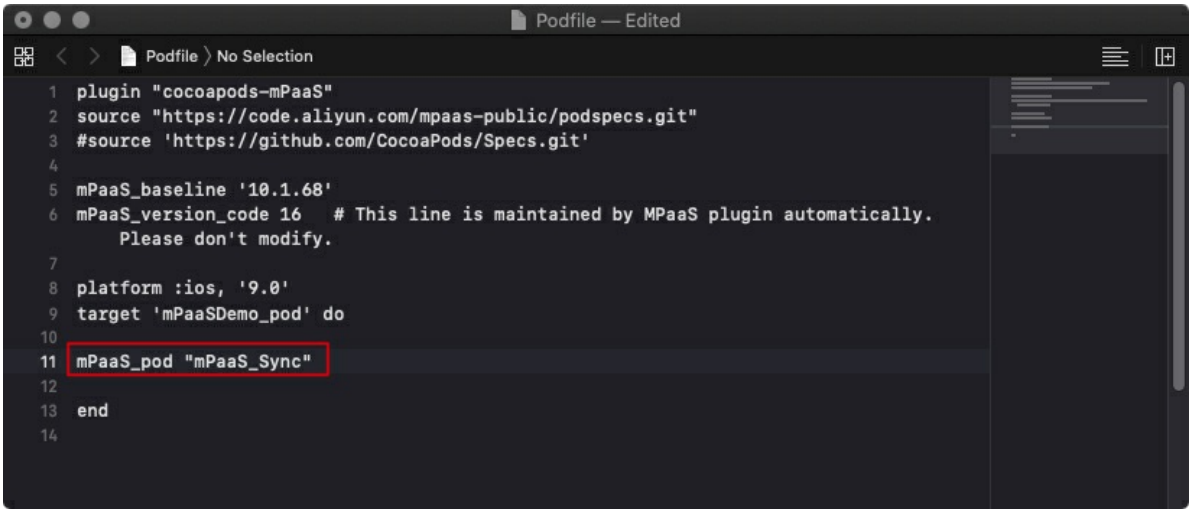
此方式适用于 基于 mPaaS 框架接入 或 基于已有工程且使用 mPaaS 插件接入 的接入方式。

1. 点击 Xcode 菜单项 Editor > mPaaS > 编辑工程，打开编辑工程页面。
2. 选择 移动同步，保存后点击 开始编辑，即可完成添加。

使用 cocoapods-mPaaS 插件

此方式适用于 基于已有工程且使用 CocoaPods 接入 的接入方式。

1. 在 Podfile 文件中，使用 `mPaaS_pod "mPaaS_Sync"` 添加移动同步组件依赖。



2. 执行 `pod install` 即可完成接入。

4.2.2. 使用 SDK

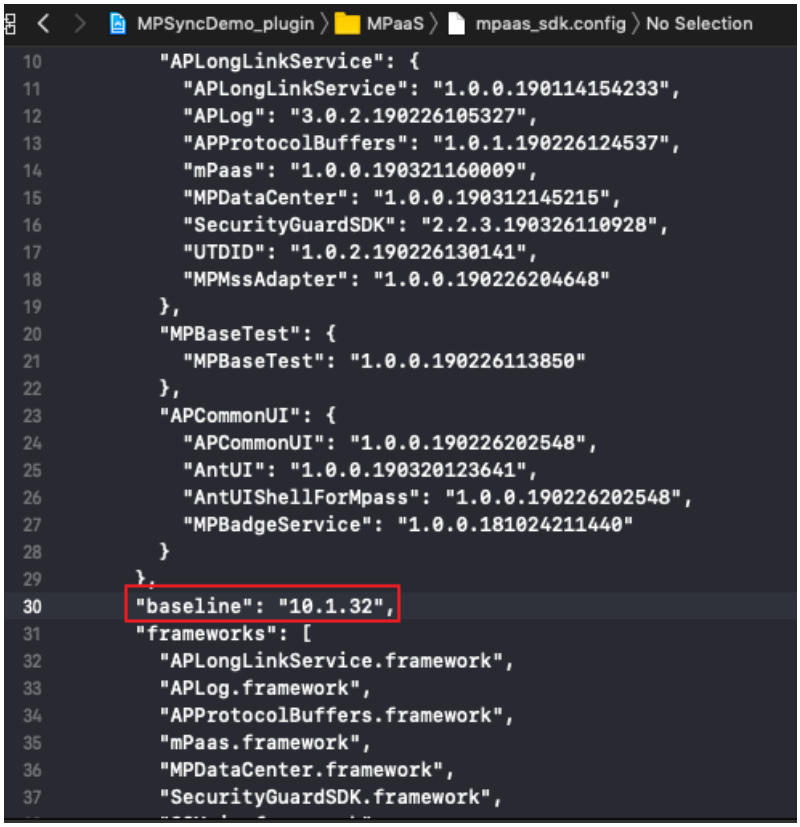
> **说明**
>
> 自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68 升级指南 或 mPaaS 10.1.60 升级指南 进行基线版本升级。

添加移动同步 SDK 后，在使用 SDK 前需要进行工程配置。

前置条件

确认您所使用的 SDK 版本 ≥ 10.1.32。

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

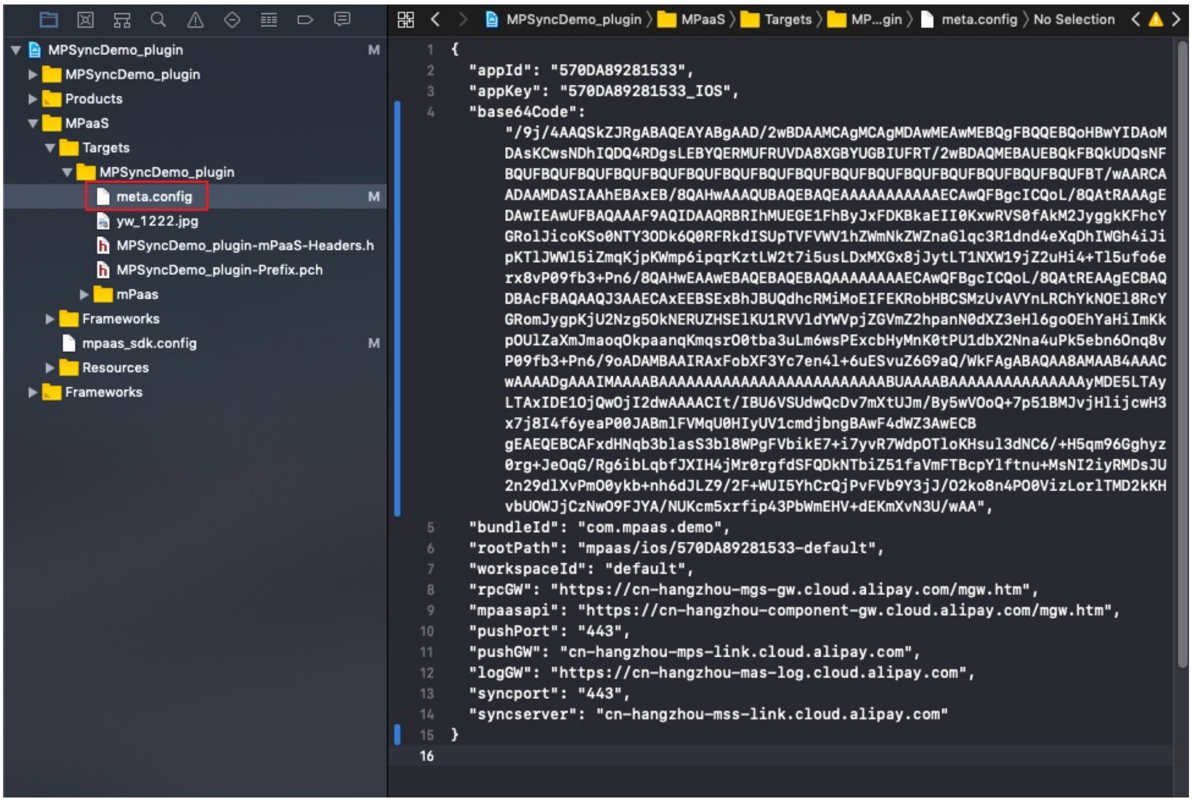


配置工程

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：

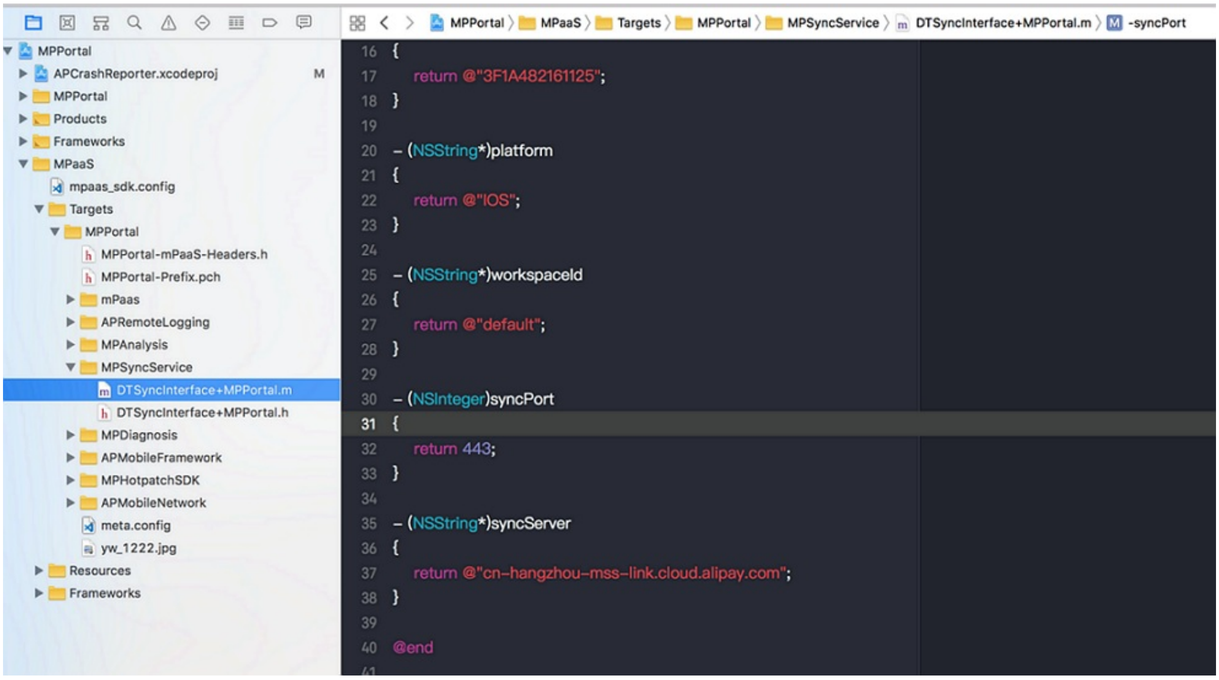
- 若您使用最新版插件添加了 Sync SDK，那么该文件会自动生成。
- 如果您的工程中没有 `meta.config` 文件，

  您可以前往 **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。



代码示例

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

在监听同步业务之前，需要先定义好具体的同步业务的业务标识名称（对应数据同步控制台上的同步配置标识），该参数就是作为使用者的您和服务提供者联系的纽带。在以下示例中，业务标识名称为 `SYNC-TRADE-DATA`。

```
#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_N
AME 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:NSUTF
```

```
8StringEncoding] 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 o
ptions:kNilOptions];
                    NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8Str
ingEncoding];
                    NSLog(@"biz payload data:%@,string:%@",dataPl,pl);
                }else{
                     NSLog(@"biz payload:%@",plString);
                }
            }
        }];
    }
}

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

5.接入服务端

5.1. 接入说明

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

5.2. 单数据同步接口

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

5.3. 全局数据同步接口

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 httpclient 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

结果码说明如下：

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
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

5.4. 用户一致性验证

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，需要确保推送的目标用户就是当前登录的用户而没有被伪造。为实现这一目的，数据同步服务提供了用户一致性验证的功能，用户可根据需要开启使用。

该功能的基本原理是：

- 客户端在连接到服务器端时，上报用户标识（userId）和授权 token（sessionId）。userId 和 sessionId 都是用户登录系统返回的数据，当 userId 和 sessionId 变化时，需要调用相关接口才能保证长连接的建立正确。
- 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，由租户来控制是否一致。数据同步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。数据同步服务端判断 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，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。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

由于数据同步服务端会固定调用 API，因此创建的 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。

- 参数说明

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，参考下表结果码说明。
returnCode	String	SUCCESS	结果码。
resultMsg	String	SUCCESS	结果信息。
result	JSONString	{ “sid” : “kkdddd” , “valid” :” true/ false” ,}	结果对象，参数说明请见下表的结果码说明。
sid	String	kkdddd	指授权的 token 或 sessionId。
valid	boolean	true / false	一致性验证结果。

- 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。

- 代码示例

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

6.使用控制台

6.1. 控制台介绍

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。一条推送配置相当于一个具体的数据推送的业务场景，而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：

- 新增推送配置
- 发送业务数据
- 查看配置详情
- 修改配置
- 下线配置
- 查询配置推送的统计数据
- 服务管理
- 查询数据同步操作记录

6.2. 新增配置

一条同步配置相当于一个具体的数据推送的业务场景，而业务数据推送就是实现该配置所对应的业务场景。因此，在推送业务数据之前，需要先创建同步配置。

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤以新增配置。

操作步骤

1. 在左侧导航栏点击 **后台服务管理 > 数据同步**，进入数据同步页面。
2. 在 **配置管理** 标签页下，点击 **+ 新建同步配置** 按钮，进入 **新建同步配置** 页面。
3. 在该页面上填写各个配置项的信息。

   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。

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。阈值推送是指仅向客户端推送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。

4. 上述配置信息填写完毕后，点击 **确定** 完成同步配置创建。新增的同步配置默认处于 **上线** 状态。同步配置上线后，您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。

6.3. 上线配置

6.4. 发送业务数据

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤新增数据同步任务，以发送业务数据。

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. 在左侧导航栏点击 **后台服务管理 > 数据同步**，进入数据同步页面。
2. 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。
3. 填写对话框中的各个配置项信息，然后点击 **确定** 按钮发送业务数据。

   各配置项的说明如下表所示。

配置项	说明
用户 ID/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。

6.5. 查看配置详情

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

操作步骤

1. 在左侧导航栏点击 **后台服务管理 > 数据同步**，进入数据同步页面。
2. 在 **配置管理** 标签页下，点击配置列表中某条配置的标识，即可查看该配置的详情。

6.6. 修改配置

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

操作步骤

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

> **说明**
>
> **同步标识** 和 **推送对象** 不支持修改。

6.7. 下线配置

数据同步期间，如需暂停数据同步（例如数据出现问题，需要紧急停止同步），可以通过下线同步配置来实现。

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤将数据同步配置下线。

操作步骤

1. 在左侧导航栏点击 **数据同步**，进入数据同步页面。
2. 在 **配置管理** 标签页下的配置列表中，点击目标配置右侧的 **下线** 并确认，即可将该配置下线。

   同步配置下线后，相应的同步业务将暂时被禁用，如果需要再次使用，可以点击 **上线** 将配置上线。

6.8. 查询配置推送的统计数据

MSS 提供用户及设备维度的推送数据统计。进入 **移动开发平台 mPaaS** 控制台后，选择目标应用，完成以下步骤查看各配置的推送统计数据。

操作步骤

1. 在左侧导航栏点击 **数据同步**，进入数据同步页面。
2. 在 **数据查询** 标签页下，您可以查看用户或设备的状态。
3. 在 **用户/设备状态查询** 区域的右上方，从下拉框选择 **用户** 或 **设备** 查询维度，在搜索框中输入用户名或设备名，可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（标识用户是否连接到 MSS 服务）
   - 近 30 天推送次数
   - 近 30 天推送到达次数
   - 推送列表

6.9. 服务管理

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。

6.10. 查询数据同步操作记录

数据同步服务支持查询数据同步操作记录，包括新建配置、修改配置、删除配置、新建同步、上线配置、下线配置、开启签名、关闭签名的变更操作历史，以便进行操作溯源以及使用数据分析。

查询操作历史记录的步骤如下：

1. 登录 mPaaS 控制台，进入目标应用后，在左侧导航栏单击 **数据同步**，进入数据同步页面。
2. 在右侧的 **操作记录** 标签页下，通过操作者账号、操作动作类型或操作日期来筛选操作记录。设置筛选条件后，单击 **搜索**，搜索结果将按时间倒序展示在操作记录列表中，展示每一项操作的操作账号、操作类型以及具体的操作时间（精确到秒）。

   > **说明**
   >
   > 在不指定时间范围的情况下，默认展示最近 7 天的变更操作历史，最多展示 1000 条操作记录。

3. 选择要查看的操作，单击右侧的 **详情** 链接，进入操作详情页面查看该操作的具体信息。
   - **新建配置**：展示新增的同步配置详情。
   - **修改配置**：展示修改的配置信息,修改的内容将标红凸显。
   - **删除配置**：展示被删除的同步配置。
   - **新建同步**：展示新增的同步任务详情，包括推送对象 ID、数据内容、数据唯一 ID、客户端应用版本区间、推送数据有效期。
   - **上线配置**：展示下线后重新上线的同步配置详情。展示的变更详情同新建配置。
   - **下线配置**：展示被下线的同步配置详情。
   - **关闭签名**：展示签名关闭操作记录。
   - **开启签名**：展示签名开启操作记录。

7.API 说明

7.1. Android 接口

> 说明
>
> 自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68 升级指南 或 mPaaS 10.1.60 升级指南 进行基线版本升级。

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。

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

涉及的公共函数列表如下：

返回值	说明
void	`setup(Application application)` 用于初始化移动同步服务依赖的基础服务，在 `initialize` 方法调用前调用。仅限 10.1.60 及以上版本基线。
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。

返回值	说明

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，需至少调用一次。

setup(Application application)

声明

```
public static void setup(Application application)
```

说明

用于初始化移动同步服务依赖的基础服务，在 `initialize` 方法调用前调用。仅限 10.1.60 及以上版本基线。

参数

参数	类型	说明
application	Application	Applicaiton 实例。

返回值

无。

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

参数	类型	说明
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

参数	类型	说明
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

参数	类型	说明
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

参数	类型	说明
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。该接口用于在登录信息 `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

参数	类型	说明
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` 。

7.2. iOS 接口

`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。如果需要监听同步服务端消息的同步标识，最好启动时开始监听。

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。

> **说明** 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。