Ant Technology

Mobile Sync Service
User Guide

Document Version: 20250303

蚂蚁集团
ANT GROUP

Legal disclaimer

Ant Group all rights reserved©2022.

No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Ant Group.

Trademark statement

蚂蚁集团 ANT GROUP and other trademarks related to Ant Group are owned by Ant Group. The third-party registered trademarks involved in this document are owned by the right holder according to law.

Disclaimer

The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Ant Group reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through channels authorized by Ant Group. You must pay attention to the version changes of this document as they occur and download and obtain the latest version of this document from Ant Group's authorized channels. Ant Group does not assume any responsibility for direct or indirect losses caused by improper use of documents.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Change history

Document version	Revisions
V20210630	Added the Query data synchronization history section, which provides instructions on how to view synchronization records.

2.About Mobile Sync Service

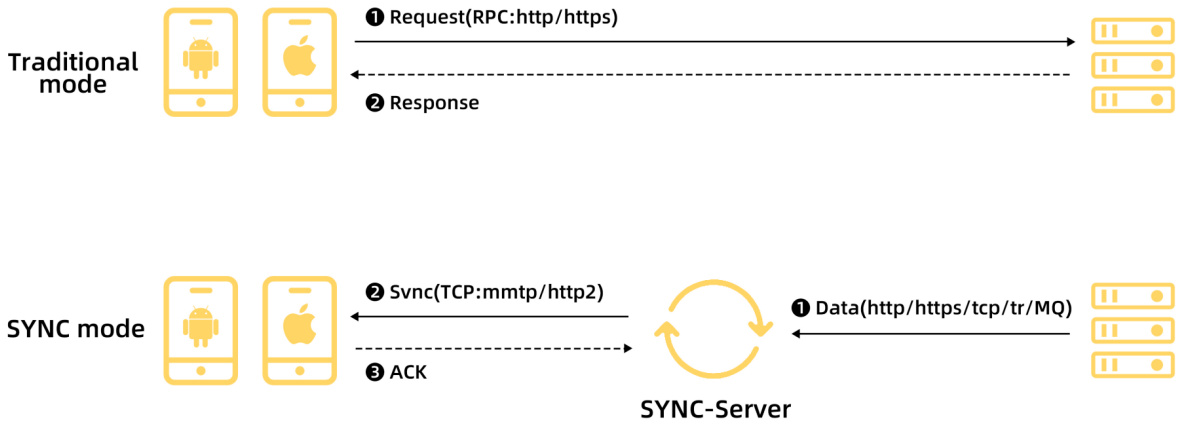
Mobile Sync Service (MSS) is a core basic business component of the mPaaS platform. MSS originates from the E2E solution SYNC of Ant Financial Group, which is oriented to mobile Apps and pushes massive data from the server to the client. This component provides a secure data channel based on the Transmission Control Protocol (TCP) and Secure Sockets Layer (SSL). This data channel can actively synchronize business data from the server to the client App on a timely, accurate, and orderly manner.

Traditional RPC has been applied in the Internet industry for decades and can meet most business scenarios and functional requirements. However, the popularization and development of the mobile Internet have driven the App scale and users' requirements for Apps to a new stage. Traditional RPC requests have many drawbacks due to their own characteristics.

- In certain scenarios, a client needs to call RPC requests to obtain the latest data, but actually no or only little data on the server (cloud) changes.
- As different business modules and functions are designed to be independent of each other, they need to call RPC requests respectively to obtain their business data when the client starts.
- The client cannot be promptly aware of the data changes on the server but needs to call RPC APIs in polling mode to update data.
- Traditional RPC performs data interactions mostly based on HTTP(S) short connections. This type of connection cannot be persistent even if by using features such as keep-alive. In other words, links cannot be reused continuously. Requests for connection creation, certificate exchange, and encryption/decryption will increase the time consumption and compromise the network performance.

MSS is introduced to improve or solve these problems.

Data synchronization



Features

The core features of MSS are described as follows:

- Reliable synchronization

  For business scenarios where the quality of service (QoS) level is arrival guarantee, MSS ensures that the data pushed from your server will be certainly synchronized to the client if the user is active within the data validity period and meets the push requirements of your server.

- Orderly and incremental synchronization

  MSS ensures that messages transmitted in the same channel arrive at the client in the same sequence as your server calls the MSS server, and all messages are synchronized to the client on an incremental basis.

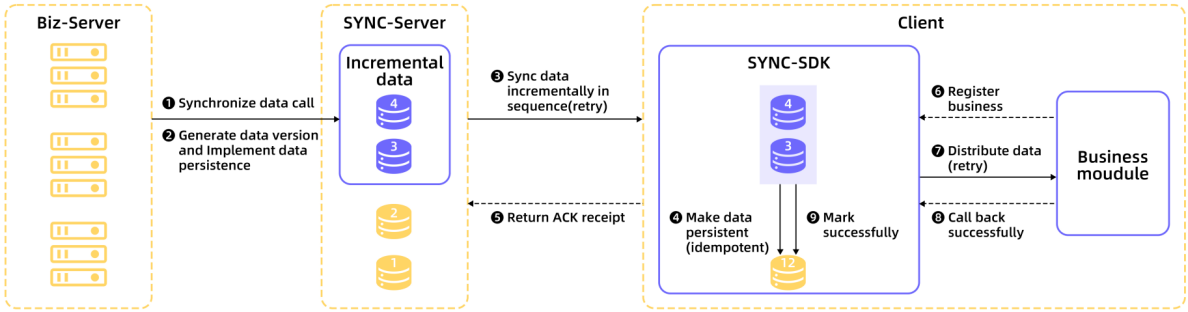
- Highly real-time performance

  When the network connection of the client is good, MSS can ensure highly real-time push performance. The time taken for message synchronization almost equals the time taken for pure data transmission over the network (that is, messages can arrive within 1s).

Basic principles

Similar to the binlog mechanism in MySQL, the basic data unit transmitted between the MSS server and the client SDK is oplog. To synchronize a piece of changed data to a specified user or device, your server needs to call the MSS API. Then the MSS server packages the data as an oplog and stores it in the database. When the client is online, the MSS server synchronizes the oplog to the client. Each oplog has a unique ID. Oplog IDs are unique and monotonically increments (based on the call sequence) among certain users and within a certain business scope. The MSS server synchronizes all oplogs to the client in ascending order of oplog ID. Both the MSS server and the client record the largest oplog ID received by the client, which is called the synchronization point (or understood as the data version number).

SYNC core —— OpLog sync



Advantages

- Merged push

  When the client is successfully initialized, the server can push multiple pieces of business data at a time to reduce the number of requests.

- Incremental push

  Only incremental data is synchronized, reducing the transmission of redundant data and the network costs.

- Reduced requests

  Data synchronization is not requested when there is no incremental data, reducing redundant requests.

- Improved time efficiency

  When the server encounters data changes, the changed data is instantly synchronized to the client, without the need to wait for requests from the client.

- Improved experience

  Data is synchronized imperceptibly and is present before the client UI is rendered, reducing the waiting time of users.

Applicable scenarios

MSS can be applied in business scenarios where data needs to be synchronized in real time to the client, such as transfer result synchronization, payment result synchronization, and message center. You can learn more about MSS capabilities through the following scenarios:

- In instant messaging Apps, MSS provides incremental and reliable message delivery capabilities to synchronize chat messages to specified users based on the message sending order of the sender.
- In Apps requiring dynamic configuration updates, MSS dynamically synchronizes configuration information to all devices. MSS synchronizes information including the app function switch, dynamic parameters, and dynamic configurations to the specified client in real time, or dynamically modifies the business parameters and configurations in batches when the App is running.
- For payment Apps, MSS provides a secure data channel for synchronizing transaction data online, ensuring that the Apps can receive the data in real time when they are online. In addition, MSS provides the data persistence capability. If data is synchronized when an App is offline, the App can receive the data when going online.

3.Terminology

The terms are listed in ascending alphabetical order.

B

Backend

A client App is running in the backend when the mobile phone displays the Home screen or in the screen-saving state, or when the user is operating another App.

BizType

A business type, which is the unique identifier of a business scenario. After data is pushed, the MSS SDK of the client distributes the data to the corresponding business module based on BizType.

Business dimension

There are two business dimensions: user and device. The MSS server pushes data by user or device.

F

Frontend

The client App is running.

I

Idempotence

Operations are applied multiple times based on the **thirdMsgId** field in the **SyncOrder** parameter, and succeed only once with the unique combination of **bizType**, **linkToken**, and **thirdMsgId**. New data will be discarded and not be added to the database. The API returns a success message with the result code “DUPLICAT ED_BIZ _ID”.

M

Multi-device Sync

A message is synchronized to all client-installed devices of a user. After the user logs in to the client that is installed on two or more devices, all the devices can receive the message. If the user uninstalls the client, re-installs it, and goes online again, the message will be pushed again.

MSS data

MSS data needs to be pushed through the MSS server.

MSS push

The MSS server proactively pushes one copy of data to the client. If the client that calls the business is online, data push is triggered immediately. Otherwise, the MSS server will push the data after the client goes online.

O

Online

The client App is connected to the network and maintains a stable TCP connection. When running in the backend, the client App is still online on most Android mobile phones but online for only 3 minutes on iPhones due to the iOS restrictions.

P

Persistence

A mechanism that converts program data between the persistent and transient states. In MSS, the persistence mechanism produces persistent data and non-persistent data.

- Persistent data: If a user is offline, the data will be stored in the database permanently. After the user goes online, the MSS SDK triggers data synchronization to the user.
- Non-persistent data: If a user is online, the data is pushed to the user immediately. If the user is offline, the data is discarded directly and will not be sent to the user after the user goes online.

Push type

There are two push types: designated push and global push.

- Designated push: pushes a piece of data to a designated user or device.
- Global push: pushes a piece of data to all online users or devices. Global push uses multi-device synchronization.

S

Single-device push

A message is pushed only to the device which a user uses for the latest login to the client. The message is pushed only once. If the user uninstalls the client, re-installs it, and goes online again, the message will not be pushed again. If the user logs in to the client from another device, the message will not be pushed to the device.

Sync

Sync refers to the MSS data synchronization service that the MSS server synchronizes data to the client App.

T

Threshold

If a user is offline for a long time and the server keeps on generating data, it may lead to a data backlog in MSS. The threshold specifies the upper limit of the amount of backlog data. When a data backlog occurs, only the latest data within the threshold is retained. Earlier data beyond the threshold will be discarded.

4.Client-side development

4.1. Android

This topic briefly describes how to fast integrate MSS to the Android client. You can access MAS through Native AAR or Portal & Bundle.

> **Important**: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide or mPaaS 10.1.60 upgrade guide.

The complete access process mainly includes the following 2 steps:

1. Add SDK
2. Use SDK

Prerequisites

You have integrated mPaaS to your project.

- If you access MSS through Native AAR, ensure that you have added mPaaS to project.
- If you access MSS in componentized access mode (through Portal & Bundle projects), ensure that you have completed the componentized access process.

Add SDK

Native AAR mode

Follow the instructions in AAR component management to install the **SYNC** component in the project through **Component management (AAR)**.

Componentized access mode

Install the **SYNC** component in the Portal and Bundle projects through **Component management (AAR)**.

For more information, see Manage component dependencies > Add/delete component dependencies.

Use SDK

In baseline 10.1.32 or later version, the `MPSync` class at the mPaaS middle layer encapsulates all APIs of MSS. You can have a quick glance of these APIs in the following table. For more information about the APIs, see Android API reference.

API	Description
setup(Application application)	Initializes basic services on which MSS depends. Call this API before you call the `initialize` method. This API is available only in baseline 10.1.60 and later versions.
initialize(Context context)	Initializes APIs and MSS.

appToForeground()	Notifies the client SDK that the App has been switched to the foreground and it needs to connect to the server. Call this API every time the App is switched to the foreground.
appToBackground()	Notifies the client SDK that the 128344 has been switched to the background and it needs to disconnect from the server. Call this API every time the App is switched to the background.
updateUserInfo(String sessionId)	Call this API when the login information (userId or sessionId) is changed. This API is called at least once.
clearUserInfo()	Clears user information when a user logs out.
registerBiz(String bizType, ISyncCallback syncCallback)	You can call this API to register a `callback` to receive business data. If this API is called, the client SDK will call the `syncCallback` class after receiving synchronized data.
unregisterBiz(String bizType)	Unregisters a specified synchronization configuration. If this API is called, the client SDK will not call the `syncCallback` class when receiving synchronized data.
reportMsgReceived(SyncMessage syncMessag)	After the data is received in the `syncCallback` implementation class, this API is called to notify MSS that the sync data has been received. Before receiving the `reportMsgReceived` message, MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data will be permanently deleted.
isConnected()	Checks whether MSS is running properly.

Code sample

This sample is based on the mPaaS SDK 10.1.32 baseline. The example App provides a button. When a user taps this button, MSS obtains the device ID, and pushes the sync data to the target device specified in the console based on the device ID. In this sample, the sync ID is `bizType`.

Note: This sample is only intended for demonstrating how to call MSS APIs, and is not the best practice of MSS. You can get the best practice code of MSS from Obtain code samples.

```
    public void button1Clicked(View view)
    {
        // Obtain the device ID using the getUtdid method.
        String utdid =UTDevice.getUtdid(MainActivity.this);
        // Print mobile sync data in Logcat.
        Log.e("=========",utdid);
        // Initialize the API and MSS.
        MPSync.initialize(MainActivity.this);
        // Register a callback for receiving service data. If this API is called, the c
lient SDK will call the syncCallback class after receiving synchronized data.
        MPSync.registerBiz("bizType",new SyncCallBackImpl());
        // Set up a network connection with the server.
        MPSync.appToForeground();

    }

    public class SyncCallBackImpl implements ISyncCallback
    {
    @Override
    public void onReceiveMessage(SyncMessage syncMessage) {
    //Print mobile sync data in Logcat.
    Log.e("=========",syncMessage.msgData);
    // Notify the MSS server that the sync data has been received.
    MPSync.reportMsgReceived(syncMessage);
    }
    }
```

Follow-up steps

- Access the server

4.2. iOS

4.2.1. Add SDK

This guide introduces how to integrate Mobile Sync Service (MSS) to iOS client. You can integrate MSS to iOS client based on native project with CocoaPods.

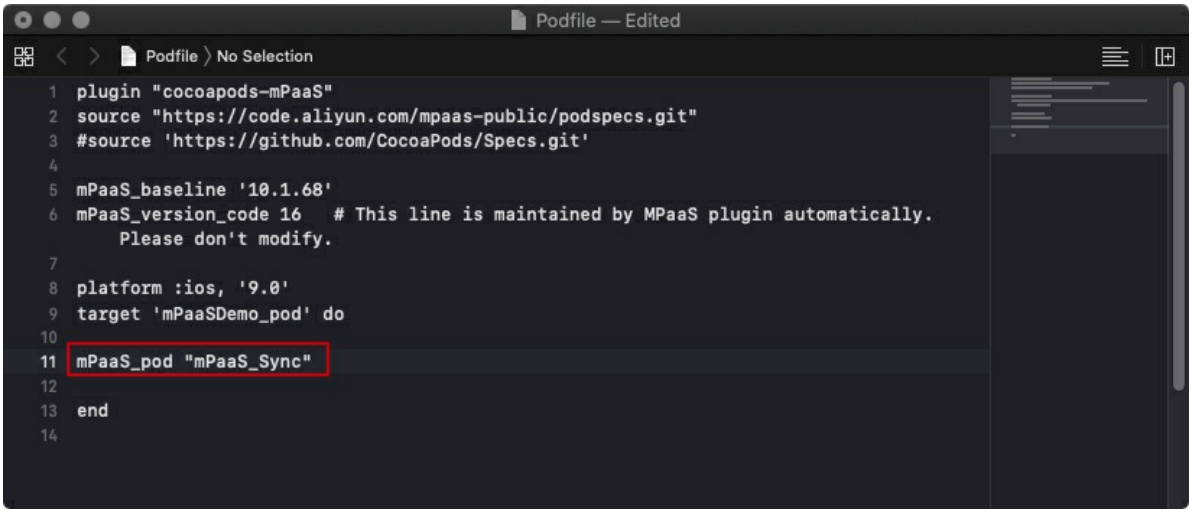
Prerequisites

You have connected your project to mPaaS. For more information, see Access based on native framework and using Cocoapods.

Add SDK

Use CocoaPods plugin to add the MSS SDK. Complete the following steps:

1. In the Podfile file, use `mPaaS_pod "mPaaS_Sync"` to add the MSS dependencies.



2. Run `pod install` to complete integrating the SDK.

4.2.2. Use SDK

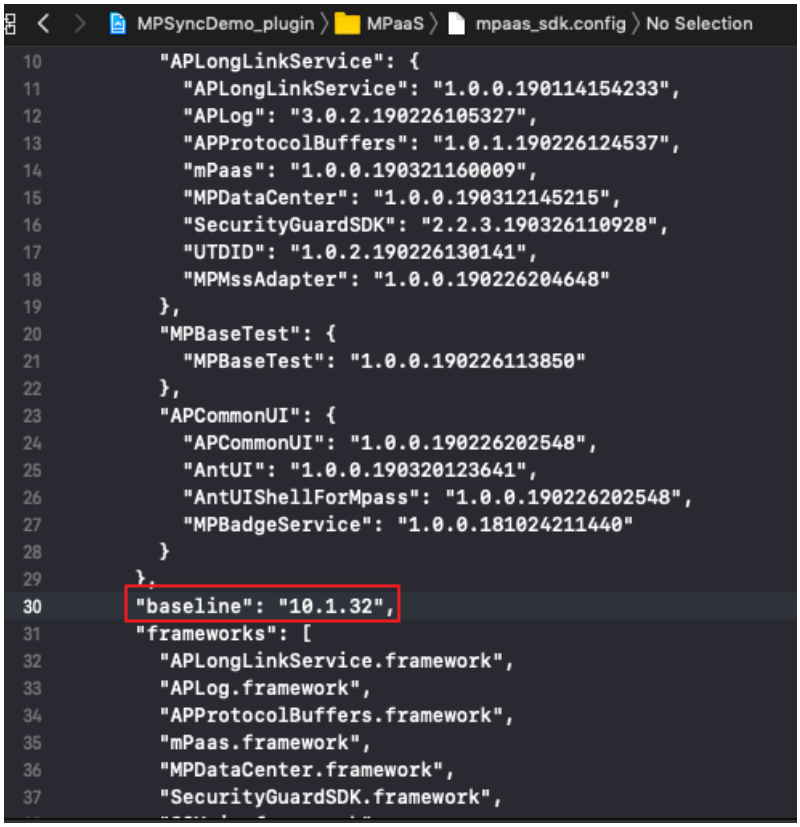
> **Important**: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide or mPaaS 10.1.60 upgrade guide.

After you add the MSS SDK, you must configure the project before using the SDK.

Prerequisites

The SDK version is 10.1.32 or later.

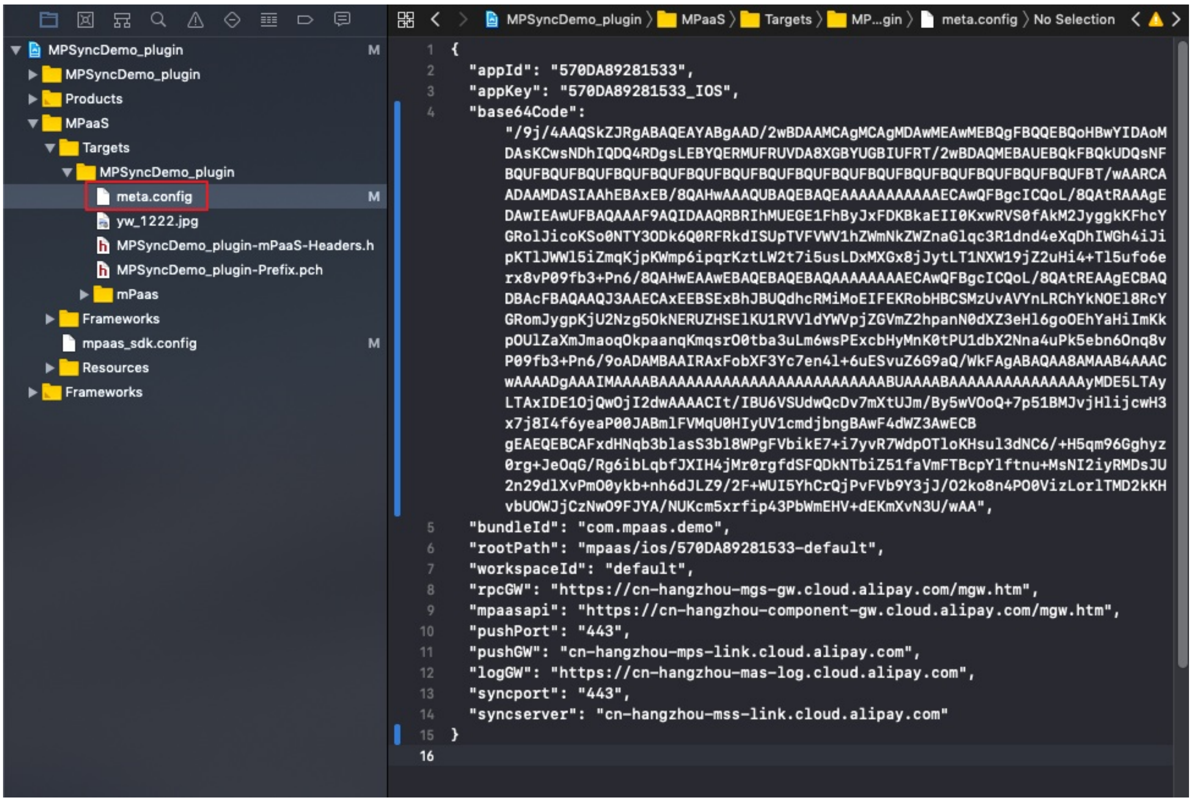
Note: You can view the current SDK version in the `mpaas_sdk.config` file.



Configure a project

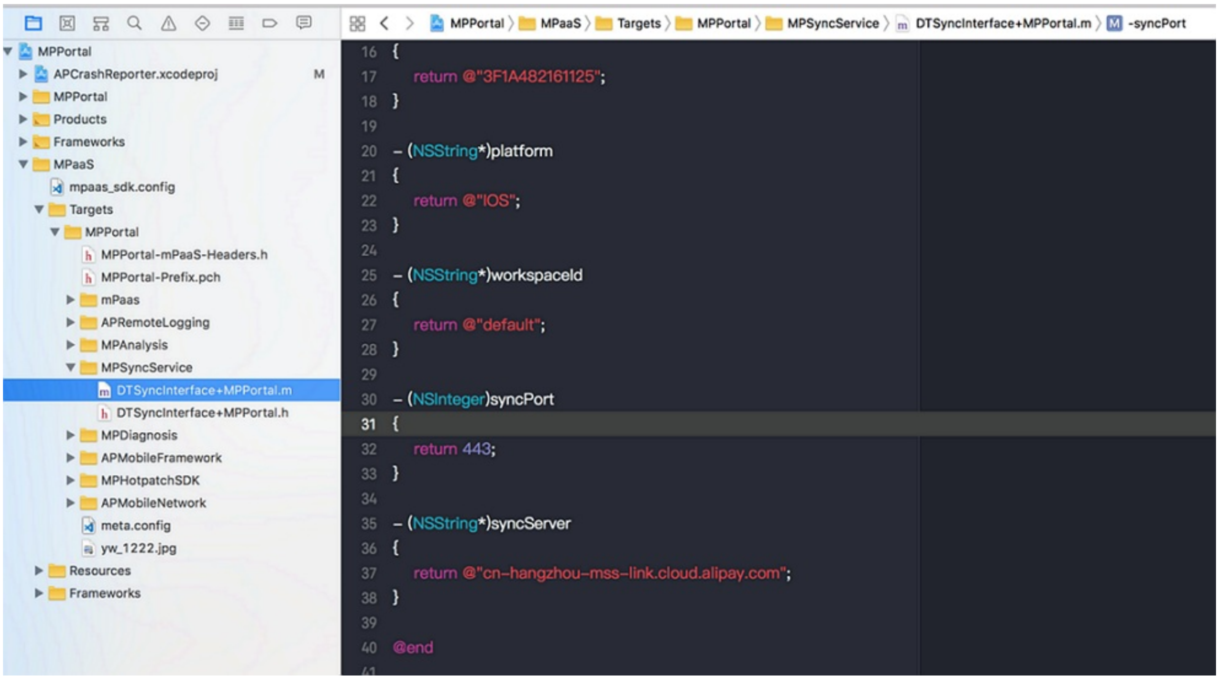
Ensure that the `meta.config` file containing the MSS address and port number has been added to the project.

- If you have used the latest plug-in to add the MSS SDK, the file will be generated automatically.
- If your project does not contain the `meta.config` file, log in to the mPaaS console, choose **Overview** > **Code configuration**, download the `.config` file, rename it to `meta.config`, and add the file to your project.



Upgrade precautions

The `Category` file of the `DTSyncInterface` class does not need to be added since version 10.1.32. The middle tier implements package reading from `meta.config`. After an upgrade, check whether there is any configuration of the earlier version in the project. If yes, remove it. The following figure shows the `Category` file of the `DTSyncInterface` class to be removed from an upgraded version.



Code sample

To realize the logic for listening on the Sync service, you need to create a class, preferably a memory-resident service, to listen on Sync messages. The following code sample creates the `MySyncService` class to listen on the Sync service.

Before listening on the Sync service, you need define a Sync ID for the sync service (the sync ID will also be used when you create a push configuration on the mPaaS console). This sync ID is the link between you as the user and the service provider. The sync ID in the following example is `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_NAME syncObserver:self
selector:@selector(revSyncBizNotification:)];
        [MPSyncInterface bindUserWithSessionId:@"SESSION_DEMO"]; // In this function, *
*User** corresponds to userId that you specify in the console. It specifies the target
to which the console delivers commands, and the value must be the same as that set in t
he userId function of MPaaSInterface. **SessionId** specifies the authorization token c
arried by the client. The user login system returns both userId and sessionId. If eithe
r changes, this function needs to be called again to ensure that a persistent connectio
n is set up correctly.
    }
    return self;
}

-(void)revSyncBizNotification:(NSNotification*)notify
{
    NSDictionary *userInfo = notify.userInfo;
    dispatch_async(dispatch_get_main_queue(), ^{
        // Process business data.
        [MySyncService handleSyncData:userInfo];
        // Call back SyncSDK, indicating that business data has been processed.
        [MPSyncInterface responseMessageNotify:userInfo];
    });
}
```

```
+(void)handleSyncData:(NSDictionary *)userInfo
{
    NSString * stringOp = userInfo[@"op"];
    NSArray *op = [NSJSONSerialization JSONObjectWithData:[stringOp
dataUsingEncoding:NSUTF8StringEncoding] options:NSJSONReadingMutableContainers error:ni
l];
    if([op isKindOfClass:[NSArray class]]){
        [op enumerateObjectsUsingBlock:^(NSDictionary * item, NSUInteger idx, BOOL
*stop) {
            if([item isKindOfClass:[NSDictionary class]]){
                NSString * plString = item[@"pl"]; // Payload of the business data
                if(item[@"isB"]){
                    NSData *dataPl = [[NSData alloc]
initWithBase64EncodedString:plString options:kNilOptions];
                    NSString *pl = [[NSString alloc] initWithData:dataPl encoding:NSUTF8
StringEncoding];
                    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
```

Follow-up steps

- Access the server

5.Server-side development

5.1. Instructions on accessing server

To access your business system to Mobile Sync Service (MSS) server, you must complete the following two steps:

To access your business system to Mobile Sync Service (MSS) server, you must complete the following two steps:

1. Integrate service with Java SDK and compile calling codes: Use Java SDK for access. According to different requirements, the calling codes can be written in two modes: **single data synchronization API** and **global data synchronization API**.
2. Verify user consistency: This verification is generally used in scenarios with high user security requirements for data synchronization.

Prerequisites

You should complete the following preparations before accessing the MSS server:

- You have completed the following operations: Activate mPaaS and Obtain AccessKey ID and Secret from the tenant administrator.
- You have created an App and obtained the appId and workspaceId of the App.
- You have a server-side application.
- You have completed the Maven configuration.

5.2. Integrate service with Java SDK

This topic describes how to access the data synchronization service on the server by using Java SDK.

Import JAR package

After completing the Maven configuration, introduce the following dependencies in the master `pom.xml` file.

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-mpaas</artifactId>
    <version>3.0.10</version>
</dependency>
<dependency>
<groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-core</artifactId>
  <optional>true</optional>
  <version>[4.3.2,5.0.0)</version>
</dependency>
```

Environment Variable Configuration

Configure environment variable MPAAS_AK_ENV and MPAAS_SK_ENV.

- Linux and macOS system configuration methods execute the following commands:

```
export MPAAS_AK_ENV=<access_key_id>
export MPAAS_SK_ENV=<access_key_secret>
```

> **Note**
> `access_key_id` is replaced with the prepared AccessKey ID, and `access_key_secret` is replaced with the AccessKey Secret.

- Windows system configuration method
  i. Create a new environment variable, add environment variables **MPAAS_AK_ENV** and **MPAAS_SK_ENV**, and write the prepared AccessKey ID and AccessKey Secret.
  ii. Restart Windows system.

API description

Single data synchronization interface

The single data synchronization interface is used to synchronize data to a specified user or device.

Parameters

Business parameters are as follows:

Parameter	Data type	Required	Example	Description
appId	String	Required	ONEX570DA892117	Get App ID from the mPaaS console.
workspaceId	String	Required	PROD	Get Workspace ID from the mPaaS console.
bizType	String	Required	UCHAT	The synchronization identifier configured in the mPaaS console. See Console introduction for more details.
linkToken	String	Required		Push target ID. Enter the user ID if the push is based on users. Enter the device ID if the push is based on devices.

payload	String	Required	testpayload	Actual business message body in custom format, no more than 4,096 characters in length.
thirdMsgId	String	Required	1760339273	Request ID for one data synchronization. Unique for one synchronization configuration. Requests of duplicate IDs will be ignored. The ID must be no more than 100 bytes.
osType	String	No	iOS/Android	Specifies the operating system of the mobile phone to which the data is to be pushed. By default, no parameters will be passed, that is, no specifications, and data will be pushed to both iOS and Android platforms.
appMinVersion	String	No	0.0.0.0	Specify the client version to which the data is pushed. Data is sent only to clients of the specified or later versions.

appMaxVersion	String	No	100.100.100.100	Specify the client version to which the data is pushed. Data is sent only to clients of the specified or earlier versions.
validTimeStart	String	No	1584448493913	Data will be pushed only when the current time is later than or equal to validTimeStart.
validTimeEnd	String	No	1584452093913	Data will be pushed only when the current time is earlier than or equal to validTimeEnd.

Result codes

Result code	Description	Solution
Success	Synchronization succeeded.	Synchronization succeeded.
ARGS_IS_NULL	Required parameters are empty	Check if the parameters have been completely passed according to the non-empty logical operation.
PAYLOAD_LONG	PAYLOAD message body is too long	Check if the length of the playload property parameter exceeds the limit.
THIRD_MSG_ID_LONG	Third-party service ID is too long.	Check if the third-party service ID exceeds the limit.
BIZ_NOT_ONLINE	The synchronization identifier of the service scenario is not submitted.	Go to **mPaaS Console** > **Mobile Sync Service** to check if the bizType synchronization identifier has been configured and submitted.
THIRD_MSG_ID_IS_NULL	Third-party service ID is empty	Check if the third-party service ID is empty.

SYSTEM_ERROR	System error	Contact technical support to confirm the cause of system errors.
INVALID_TENANT_ID	Invalid tenant ID	Check if the App ID is correct and If you have the permission to use the App ID.

Sample code

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.mpaas.model.v20201028.CreateOpenSingleDataRequest;
import com.aliyuncs.mpaas.model.v20201028.CreateOpenSingleDataResponse;
import com.aliyuncs.profile.DefaultProfile;
import org.apache.commons.lang3.builder.ToStringBuilder;
import org.apache.commons.lang3.builder.ToStringStyle;

public class MsyncPopDemo {

    public static void main(String[] args) {
        //Request information, except AccessKey ID AccessKey Secret can be fixed
        DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-
hangzhou.aliyuncs.com");

        // Alibaba Cloud account AccessKey has access rights to all APIs. It is
recommended that you use RAM users for API access or daily operation and maintenance.
        // It is strongly recommended not to save the AccessKey ID and AccessKey Secret
in the project code, otherwise the AccessKey may be leaked, threatening the security of
all resources under your account.
        // This example uses saving the AccessKey ID and AccessKey Secret in
environment variables as an example. You can also save it to the configuration file acc
ording to business needs.
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");

        // Create a DefaultAcsClient instance and initialize it
        DefaultProfile profile = DefaultProfile.getProfile(
                "cn-hangzhou", // Region ID
                accessKeyId,   // AccessKey ID of RAM account
                accessKeySecret);  // AccessKey Secret of RAM account
        IAcsClient client = new DefaultAcsClient(profile);

        CreateOpenSingleDataRequest singleRequest = constructSingleRequest();

        CreateOpenSingleDataResponse singleDataResponse;
        try {
            singleDataResponse = client.getAcsResponse(singleRequest);
            System.out.println("singleDataResponse:" +
                    ToStringBuilder
                            .reflectionToString(singleDataResponse,
ToStringStyle.SHORT_PREFIX_STYLE));
```

```
        } catch (Throwable throwable) {
            throwable.printStackTrace();
        }
    }

    private static CreateOpenSingleDataRequest constructSingleRequest() {

        CreateOpenSingleDataRequest singleRequest
                = new CreateOpenSingleDataRequest();
        //*************Required properties*************/

        //App ID obtained from the mPaaS console
        singleRequest.setAppId("xxxxxxx");
        //WorkspaceId obtained from the mPaaS console
        singleRequest.setWorkspaceId("xxxxxxxx");
        //The synchronization identifier configured during mobile synchronization in
the mPaaS console
        singleRequest.setBizType("TEST-SYNC");
        //User ID or device ID to be pushed (UTDID)
        singleRequest.setLinkToken("testUserId");
        //Actual service message body, custom format with not more than 4096 characters
in length.
        singleRequest.setPayload("testPayload");
         //Service ID, unique, not more than 100 characters in length.
        singleRequest.setThirdMsgId("test_third_msg_id_" + System.currentTimeMillis());

        //*************Non-required properties*************/

        //No restriction on the operating system when the operating system of the targe
t device, iOS or Android, is empty.
        singleRequest.setOsType("IOS");
        //Minimum client version supported, such as 8.6.0.0.9999. If the version
specified here is empty, there will be no limit on the minimum client version.
        singleRequest.setAppMinVersion("0.0.0.0");
        //Maximum client version supported, such as 9.0.0.0.9999. If the version
specified here is empty, there will be no limit on the maximum client version.
        singleRequest.setAppMaxVersion("100.100.100.100");
        //Start of the validity period. If it is empty, there will be no limit on the s
tart of the validity period.
        singleRequest.setValidTimeStart(System.currentTimeMillis());
        //End of the validity period. If it is empty, there will be no limit on the end
of the validity period. The longest validity period is 30 days.
        singleRequest.setValidTimeEnd(System.currentTimeMillis() + (1000 * 3600));

        return singleRequest;

    }
```

Important

Please make sure that your AccessKey has AliyunMPAASFullAccess permission. For details, please refer to Application-level access control for RAM users.

Global data synchronization interface

Global data synchronization interface is used to synchronize data to all devices.

Parameters

Business parameters are as follows:

Parameter	Data type	Required	Example	Description
appId	String	Required	ONEX570DA892117	Get App ID from the mPaaS console.
workspaceId	String	Required	PROD	Get Workspace ID from the mPaaS console.
bizType	String	Required	UCHAT	The synchronization identifier configured in the mPaaS console. See Console introduction for more details.
payload	String	Required	testtestatapalayd	Actual service message body, custom format with not more than 4096 characters in length.
thirdMsgId	String	Required	1760339273	One data synchronization request ID. Unique for one synchronization identifier. Requests from duplicate IDs will be ignored. The ID must be no more than 100 bytes.

osType	String	No	IOS/ANDROID	Specifies the operating system of the mobile phone to which the data is to be pushed. By default no parameters will be passed, that is, no specifications, and data will be pushed to both iOS and Android platforms.
appMinVersion	String	No	0.0.0.0	Specify the client version to which the data is pushed. Data is sent only to clients of the specified or later versions.
appMaxVersion	String	No	100.100.100.100	Specify the client version to which the data is pushed. Data is sent only to clients of the specified or earlier versions.
validTimeStart	String	No	1584448493913	Data will be pushed only when the current time is later than or equal to validTimeStart.
validTimeEnd	String	No	1584452093913	Data will be pushed only when the current time is earlier than or equal to validTimeEnd.

maxUid	Long	No	99	The maximum Uid in the synchronization range. Uid is the second last character and the third last character of the user ID or device ID. If the Uid is not alphabetic, you need to convert the Uid to ASCII.
minUid	Long	No	00	The minimum Uid in the synchronization range. Uid is the second last character and the third last character of the user ID or device ID. If the Uid is not alphabetic, you need to convert the Uid to ASCII.
uids	String	No	01,02,99	The priority is higher than maxUid and minUid. The discrete Uid segment. Uid is the second last character and the third last character of the user ID or device ID. If the Uid is not alphabetic, you need to convert the Uid to ASCII.

Result codes

Result code	Description	Solution
Success	The task is successful.	The task is successful.

ARGS_IS_NULL	Required parameters are empty	Check if the parameters have been completely passed according to the non-empty logical operation.
PAYLOAD_LONG	PAYLOAD message body is too long	Check if the length of the playload property parameter exceeds the limit.
THIRD_MSG_ID_LONG	Third-party service ID is too long.	Check if the third-party service ID exceeds the limit.
BIZ_NOT_ONLINE	The synchronization identifier of the service scenario is not submitted.	Go to **mPaaS Console** > **Mobile Sync Service** to check if the bizType synchronization identifier has been configured and submitted.
THIRD_MSG_ID_IS_NULL	Third-party service ID is empty	Check if the third-party service ID is empty.
SYSTEM_ERROR	System error	Contact technical support to confirm the cause of system errors.
NOT_SUPPORT_GLOBAL	Does not support calls with global service synchronization identifier	According to BizType, go to **mPaaS Console** > **Mobile Sync Service** to check if the synchronization identifier is user-based or device-based.
INVALID_TENANT_ID	Invalid tenant ID	Check if the App ID is correct and If you have the permission to use the App ID.

Sample code

```
public static void main(String[] args) {

        //Request information, fixed except AccessKey ID and AccessKey secret
        DefaultProfile.addEndpoint("cn-hongkong", "mpaas", "mpaas.cn-
hongkong.aliyuncs.com");
        // Create and initialize a DefaultAcsClient instance.
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hongkong", // Region ID
            "xxxxxx", // AccessKey ID of the RAM account
            "xxxxxx"); // AccessKey secret of the RAM account
        IAcsClient client = new DefaultAcsClient(profile);

        CreateOpenGlobalDataRequest globalDataRequest = constuctGlobelRequest();
```

```
        CreateOpenGlobalDataResponse globalDataResponse;
        try {
            globalDataResponse = client.getAcsResponse(globalDataRequest);
            System.out.println("globalDataResponse:" +
                ToStringBuilder
                    .reflectionToString(globalDataResponse,
ToStringStyle.SHORT_PREFIX_STYLE));

        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            e.printStackTrace();
        } catch (com.aliyuncs.exceptions.ClientException e) {
            e.printStackTrace();
        } catch (Throwable throwable) {
            throwable.printStackTrace();
        }
    }

 private static CreateOpenGlobalDataRequest constuctGlobelRequest() {

        CreateOpenGlobalDataRequest globalRequest
            = new CreateOpenGlobalDataRequest();

        //*************Required properties*************/
        //App ID obtained from the mPaaS console
        globalRequest.setAppId("BE9C457161429");
        //WorkspaceId obtained from the mPaaS console
        globalRequest.setWorkspaceId("sit");
        //The synchronization identifier configured during mobile synchronization in
the mPaaS console
        globalRequest.setBizType("test-global");
        //Actual service message body, custom format with not more than 4096 characters
in length.
        globalRequest.setPayload("testtestata");
        //Service ID, unique, not more than 100 characters in length.
        globalRequest.setThirdMsgId("test_third_msg_id_" + System.currentTimeMillis());

        //*************Non-required properties*************/

        //No restriction on the operating system when the operating system of the targe
t device, iOS or Android, is empty.
        globalRequest.setOsType("IOS");
        //Minimum client version supported, such as 8.6.0.0.9999. If the version
specified here is empty, there will be no limit on the minimum client version.
        globalRequest.setAppMinVersion("0.0.0.0");
        //Maximum client version supported, such as 9.0.0.0.9999. If the version
specified here is empty, there will be no limit on the maximum client version.

        globalRequest.setAppMaxVersion("100.100.100.100");
        //Maximum Uid
        globalRequest.setMaxUid(Long.valueOf(99));
        //Minimum Uid
        globalRequest.setMinUid(Long.valueOf(1));
```

```
        //Uid 00-99 to be pushed for the phased-release, which is a string array.
        globalRequest.setUids("01,02,99");

        globalRequest.setValidTimeStart(System.currentTimeMillis());
        globalRequest.setValidTimeEnd(System.currentTimeMillis() + (1000 * 3600));

        return globalRequest;

    }
```

> **Important**
>
> Please make sure that your AccessKey has AliyunMPAASFullAccess permission. For details, please refer to Application-level access control for RAM users.

5.3. Check user consistency

In some cases, the business system has high-security requirements on data synchronization, namely, the target users of the push must be the current logon users and not fake. For that, the MSS provides user consistency verification, which can be turned on by the user when needed. The general principle of this function is:

In some cases, the business system has high-security requirements on data synchronization, namely, the target users of the push must be the current logon users and not fake. For that, the MSS provides user consistency verification, which can be turned on by the user when needed. The general principle of this function is:

- The client reports user ID (userId) and authorization token (sessionId) when the client connects to the server. Both userId and sessionId are the data returned after the user logs on to the system. When userId and sessionId change, the relevant APIs need to be called to ensure that the persistent connection is established correctly.
- The server calls a consistency verification interface implemented by the tenant, and the tenant checks the consistency through this interface. The data synchronization service records an identifier indicating whether the consistency requirement is met.
- For synchronization configuration with high security requirements, the tenant can enable user consistency verification, and data is pushed only to devices of users who have passed the consistency verification. If user consistency verification is not enabled, the consistency verification results are ignored.

Configure user consistency verification interface

The following section describes how to configure the consistency validation interface `com.antcloud.session.validate` and explains the interface usage.

Note that after configuring the consistency verification interface in the mPaaS console, you need to disable the **signature verification** feature of this RPC. Otherwise, the logic of consistency verification for mobile synchronization will not work properly.

Operation path

After you log on to the mPaaS console, select the target App and choose **Mobile Gateway Service** > **Manage API** to add the API. For more information, see Mobile Gateway > Manage APIs.

API description

The `operationType` of the API to be added must be `com.antcloud.session.validate`. The request parameters are as follows:

Parameter	Type and length	Required	Example	Description
InstanceId	String	Required	instancedemo	String of workspaceId_appId
userId	String	Required	20880939	User ID.
sessionId	String	Required	kkdddd	Authorization token carried by the client.

Returned parameters

The data returned after implementing the consistency verification logic is in JSON format, as shown in the following example:

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

Attribute description:

Parameter	Data type	Example	Description

success	boolean	true/false	The business call result. Valid values: `true` and `false`, where `true` indicates a successful call and `false` indicates a failed call. If the call fails, check the value of `returnCode` to locate the cause. For more information, see Result codes as follows .
returnCode	String	ERROR	The result code.
resultMsg	String	SYSTEM-ERROR	Result information.
sid	String	kkdddd	The authorization token or sessionId.
valid	boolean	true/false	Verification result.

Result codes

Result	Result code	Description
true	OK	Business call succeeded.
false	OPERATION_ERROR	The operation fails. Only the `com.antcloud.session.validate` API is called.

6.Console operations

6.1. Console introduction

The **Mobile Sync** console allows you to manage push configurations and perform data push actions. A push configuration defines the basic application scnenario of the push service. And the actual data push actions can be realized based on the push configuration.

You can perform the following actions in the **Mobile Sync** console:

- Add cnfigurations
- Send business data
- View configuration details
- Modify configurations
- Disable configurations
- Query statistics on configuration pushes
- Service management

6.2. Add configuration

A synchronization configuration defines the basic application scenario of data push. And the actual data push actions can be realized based on the synchronization configuration. Therefore, you need to add synchronization configuration before sending data.

Log in to mPaaS console, click the mPaaS App for which you want to add configuration, and complete the following steps.

Procedure

1. On the left navigation pane, choose **Mobile Sync Service** under **Backend connection**.
2. Click the **Configuration management** tab, and then click **+ New sync configuration**. The **New sync configuration** page appears.
3. Set parameters.

   The following table describes the parameters.

Parameter	Description
Sync ID	Identifies a specific data push business scenario. The format of uppercase letters with a hyphen (-), such as DEVICE-LOCK, is recommended.
Description	Describes the business scenario corresponding to the configuration.

Push scope	Indicates the range of users or devices receiving data in the data push process. The value **Global** indicates that all users or devices can receive data, and the value **Appointed** indicates that only the appointed user or device can receive data.
Target	Indicates whether data is pushed by user or by device.
Multi-device sync	This parameter is required only when **Target** is set to **User**. If you select **Yes**, data will be synchronized between multiple devices of a single user. That is, when the user uses a device to log in to the client, the user can receive the data that the user has received on another device.
Data persistence	Pushed data will be saved to the database for a maximum of 30 days by default. If a user is offline when data is pushed, the user will receive the data when going online.
Re-push mode	Specifies the policy for processing the backlog data on the server. This parameter is available only when **Data persistence** is set to **Yes**. When **All** is selected, all the backlog data on the server will be pushed to the client. When **Threshold** is selected, only the latest backlog data within the threshold will be pushed to the client.
Re-push threshold	This parameter is available only when **Data persistence** is set to **Yes** and **Re-push mode** is set to **Threshold**.
User consistency check	This parameter is available only when **Target** is set to **User**. If you set this parameter to **Yes**, MSS will verify user consistency when pushing data and push data only when user consistency check is successful. For more information, see Verify user consistency.

4. After setting the above information, click **OK** to complete adding the synchronization configuration. The newly added synchronization configuration becomes online by default.

   Once a configuration is taken online, you can push data by calling APIs or performing actions in the console.

6.3. Send business data

This topic describes how to send business data in the mPaaS console. Enter your target App and complete the following steps.

Prerequisites

One push configuration record exists in the console and is online.

Procedure

1. On the left-side navigation pane, choose **Mobile Sync Service**.
2. Under the **Configuration management** tab, click **Operate** of a configuration record in the configuration list. The **Create synchronization** window appears.
3. Set parameters, and click **OK**.

   The following table describes the parameters.

Parameter	Description
User ID/Device ID	Indicates the user or device to which the business applies.
Content	Indicates the text content of the data, in String format.
Unique data ID	Uniquely identifies the data content. This parameter is required only for the data persistence business. When two data records with the same unique data ID are pushed, the second record will be ignored.
OS	Indicates the operating system type of the data receiving client. The options are **Android** and **iOS**.
Version range	Indicates the range of data receiving client app versions. This parameter is optional.
Validity period	Indicates the maximum validity period of the pushed data. The default value is 30, in days.

6.4. View configuration details

This topic describes how to view configuration details in the **mPaaS** console.

Enter your target App and complete the following steps to view the configuration details:

1. In the left navigation pane, click **Mobile Sync**.
2. Under the **Configuration management** tab, click the ID of a configuration record in the configuration list to view the details.

6.5. Change settings

This topic describes how to modify push configurations in the **mPaaS** console.

Enter your target App, and complete the following steps to modify a piece of push configuration:

1. In the left navigation pane, click **Mobile Sync**.
2. Under the **Configuration management** tab, click the ID of a configuration record in the configuration list.
3. On the displayed configuration details page, click **Modify** in the upper right corner. Modify parameters as required, and click **Save**.

Note: **Sync ID** and **Target** cannot be modified.

6.6. Disable configuration

In case data synchronization needs to be suspended due to data problems or other reasons, you can do it by disabling the synchronization configuration in the mPaaS console.

In case data synchronization needs to be suspended due to data problems or other reasons, you can do it by disabling the synchronization configuration in the mPaaS console.

In the mPaaS console, select your App, and complete the following steps to disable the synchronization configuration:

1. On the left navigation pane, click **Mobile Sync Service**, and then go to the **Configuration management** tab page.
2. In the synchronization configuration list, click **Offline** right to the target configuration, and confirm to take the configuration offline.

   Once the synchronization configuration is disabled, all the corresponding synchronization business will be disabled accordingly. To use the configuration again, you just need to click **Online** to take the configuration online.

6.7. Query configuration pushes

MSS displays pushed statistical data by user and device status.

MSS displays pushed statistical data by user and device status.

This topic describes how to view pushed statistical data in the mPaaS console. Enter your target App and complete the following steps.

Procedure

1. On the left navigation pane, click **Mobile Sync Service**.
2. Click the **Data query** tab to view user or device status.
3. Select **User** or **Device** in the upper right of the **User/device status** area, and enter a user name or device name in the search box accordingly to view the status of the user or device.

MSS provides the following user or device data on this page:

- User/device name
- Status of whether the user connects to MSS
- Pushes in the last 30 days
- Arrivals in the last 30 days
- Push list

6.8. Manage services

On the **Service Management** tab page, a switch is available for enabling or disabling signature. The setting is effective globally. You can temporarily enable or disable all signature verification related functions as needed.

7.API reference

7.1. Android API

Important: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide or mPaaS 10.1.60 upgrade guide.

In baseline 10.1.32 or later versions, the `MPSync` class in the mPaaS middle layer encapsulates all APIs of the Mobile Sync Service (MSS). You can use the `MPSync` object to implement all functions of MSS.

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

Related public functions are shown as follows:

- setup(Application application)
- appToBackground()
- appToForeground()
- clearUserInfo()
- initialize(Context context)
- isConnected()
- registerBiz(String bizType, ISyncCallback syncCallback)
- reportMsgReceived(SyncMessage syncMessag)
- unregisterBiz(String bizType)
- updateUserInfo(String sessionId)

Return value type	Methods and description
void	setup(Application application) Initializes basic services on which MSS depends. Call this API before you call the `initialize` method. This function is available only in baseline 10.1.60 and later versions.
void	appToBackground() Notifies the client SDK that the App has been switched to the background and it needs to disconnect from the server. Call this function every time the App is switched to the background.
void	appToForeground() Notifies the client SDK that the App has been switched to the foreground and it needs to connect to the server. Call this function every time the App is switched to the foreground.

Return value type	Methods and description
void	clearUserInfo() Clears user information when a user logs out.
void	initialize(Context context) Initializes MSS.
boolean	isConnected() Checks whether MSS is running properly.
void	registerBiz(String bizType, ISyncCallback syncCallback) Registers a callback to receive business data. If this API is called, the client SDK will call the syncCallback class after receiving synchronized data.
void	reportMsgReceived(SyncMessage syncMessag) Notifies MSS of the data synchronization success after data is received in the syncCallback class. Before receiving reportMsgReceived, MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data is permanently deleted.
void	unregisterBiz(String bizType) Unregisters a specified synchronization configuration. If this API is called, the client SDK will not call the syncCallback class when receiving synchronized data.
boolean	updateUserInfo(String sessionId) Call this API at least once when the login information (userId or sessionId) is modified.

setup(Application application)

Declaration

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

Description

Used to initialize the base service that MSS depends on. This function needs to be called before the initialize method is called. This function is available only in baseline 10.1.60 and later versions.

Parameters

Parameter	Type	Description
application	Application	An application instance.

Returned value

None.

appToBackground()

Declaration

```
public static void appToBackground()
```

Description

Notifies the client SDK that the App has been switched to the background and it needs to disconnect from the server. Call this function every time the App is switched to the background.

We recommend that you call this API inside the `onStop()` method of the home page. If this API is not called when the App is switched to the background, the network connection between the App and the server cannot be released in a timely manner, increasing power consumption and traffic usage.

Parameters

None.

Returned value

None.

appToForeground()

Declaration

```
public static void appToForeground()
```

Description

Notifies the client SDK that the App has been switched to the foreground and it needs to connect to the server. Call this function every time the App is switched to the foreground.

We recommend that you call this API inside the `onResume()` method of the home page.

Parameters

None.

Returned value

None.

clearUserInfo()

Declaration

```
public static void clearUserInfo()
```

Description

Clears user information when a user logs off.

Parameters

None.

Returned value

None.

initialize(Context context)

Declaration

```
public static void initialize(Context ctx)
```

Description

You can call this API to initialize MSS. Your App can use MSS only after you call this API.

During the life cycle of the App (from the time the App is started to the time the App is stopped), this API needs to be called only once.

Parameters

Parameter	Type	Description
ctx	Context	A non-empty `Context`.

Returned value

None.

isConnected()

Declaration

```
public static boolean isConnected()
```

Description

Checks whether MSS is running properly.

Parameters

None.

Returned value

Returns true if the service is normal, and returns false if the service is abnormal.

registerBiz(String bizType, ISyncCallback syncCallback)

Declaration

```
public static void registerBiz(String biz, ISyncCallback callback)
```

Description

Used to register a callback for receiving service data. If this API is called, the client SDK will call the syncCallback class after receiving synchronized data.

This API needs to be called once for each synchronization configuration.

Parameters

Parameter	Type	Description
bizType	String	Synchronization identifier
syncCallback	ISyncCallback	Callback implementation class

Returned value

None.

reportMsgReceived(SyncMessage syncMessag)

Declaration

```
public static void reportMsgReceived(SyncMessage msg)
```

Description

After the synchronously pushed data is received in `syncCallback`, call this API to notify MSS that the synchronized data has been received successfully. Before receiving the `reportMsgReceived`, MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data will be permanently deleted.

Parameters

Parameter	Type	Description
syncMessag	SyncMessage	Message synchronization

Returned value

None.

unregisterBiz(String bizType)

Declaration

```
public static void unregisterBiz(String biz)
```

Description

Unregisters a specified synchronization configuration. MSS will not call `syncCallback` after MSS receives the synchronization configuration data.

Parameters

Parameter	Type	Description
biz	String	Synchronization identifier

Returned value

None.

updateUserInfo(String sessionId)

Declaration

```
public static boolean updateUserInfo(String sessionId)
```

Description

Calling inside the method is based on the `LongLinkSyncService.getInstance().updateUserInfo(String userId, String sessionId)` API, in which `userId` indicates the user ID specified in `MPLogger` .This API is called when `userId` or `sessionId` changes and will update user login information.

Both parameters are required for logon. If `userId` is not specified, this method returns `false` , indicating a calling failure.

This method must be called upon session expiration or each successful automatic logon. Note that the automatic logon function is enabled after a user logs on to the client once. The general calling principle is that this method must be called when `userId` or `sessionId` changes.

Parameters

Parameter	Type	Description
sessionId	String	Session ID.

Returned value

Returns true if the user information is updated successfully, and returns false if userId is not set at logon.

7.2. iOS API

The `MPSyncInterface` class in `MPMssAdapter.framework` provides all MSS APIs. All methods in the class are class methods that can be called by the class name.

+(void)initSync;

Initializes MSS. An app can use MSS only after calling this API.

During the life cycle of the app (from the time the app is started to the time the app is stopped), this API needs to be called only once.

+(MPSyncNetConnectType)connectStatus;

Checks the connection status of MSS.

Return value: connection status specified by `MPSyncNetConnectType` .

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

Registers the notification listener which works on the business name `bizName` , and calls `[[NSNotificationCenter defaultCenter] addObserver:observer selector:selector name:bizName object:nil];` to listen on notifications.

The value of **bizName** is the same as that in the server console. If this API is not called, the specified biz messages will not be distributed but stacked in the database of the client SDK. We recommend that you start listening on specified sync messages sent to the server upon server startup.

Return value: registration result `YES` or `NO`.

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

Notifies the MSS client SDK that message listening on a synchronization configuration has been disabled and that sync messages related to the synchronization configuration will no longer be received.

The internal `[[NSNotificationCenter defaultCenter] removeObserver:observer name:bizName object:nil];` API is called to remove the listener.

After this API is called, messages of the biz will not be distributed but stacked in the SyncSDK database. This API matches the `registerSyncBizWithName` API.

Return value: result `YES` or `NO`.

+(void)removeSyncNotificationObserver:(id)observer;

Disables listening on the synchronization notification. This API is usually called in the `dealloc` function of a listening class. The internal `[[ NSNotificationCenter defaultCenter] removeObserver:observer];` API is called to remove the listener.

Return value: none.

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

Notify a `callback` after a message has been processed. The parameter is `userInfo(notify.userInfo)` in the notification.

Calls back `SyncSDK`, indicating that the business data has been processed in the notification processing function registered using the `registerSyncBizWithName` API, when data processing is completed.

Return value: none.

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

This method is called when the value of the login parameter `userId` or `sessionId` changes.

This API is called during login. The value of `userId` is the `-(NSString\*)userId` function of `MPaaSInterface`.

This method must be called upon `sessionId` expiration or each successful automatic login, which is enabled after a user logs in to the client once.

The overall calling principle is that this method must be called when the value of `userId` or `sessionId` changes.

When the value of `userId` changes, `unBindUser` is called to unbind the user account and then `bindUserWithSessionId:` is called to rebuild a connection.

sessionId is used with the server to verify the validity of a session. If this parameter is set to **nil** on the server, the default value `@”SESSION_DEMO”` is used.

Return value: none.

+(void)unBindUser;

Called to unbind the currently connected user when the user logs out.

Return value: none.

+(NSString *)getSyncDeviceId;

Obtains the device ID, which is used when pushing device-based sync data.

Return value: device ID.

Important: If the value of `sessionId` in the API is invalid, the user consistency option in the console must be disabled, or sync messages will fail to be pushed due to verification failure. Enable or disable signature verification by referring to Service management.