

Ant Technology

Mobile Sync Service User Guide

Document Version: 20240809



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 <code>cd /d C:/window</code> command to enter the Windows system folder.
<i>Italic</i>	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	This format is used for a required value, where only one item can be selected.	<code>switch {active stand}</code>

Table of Contents

1. Change history	05
2. About Mobile Sync Service	06
3. Terminology	09
4. Client-side development	11
4.1. Android	11
4.2. iOS	13
4.2.1. Add SDK	13
4.2.2. Use SDK	14
5. Access the server	19
5.1. Integrate into MSS	19
5.2. Single data synchronization API	19
5.3. Global data synchronization API	26
5.4. Verify user consistency	33
6. Console operations	36
6.1. Console introduction	36
6.2. Add configuration	36
6.3. Send business data	37
6.4. View configuration details	38
6.5. Change settings	39
6.6. Disable configuration	39
6.7. Query configuration pushes	39
6.8. Manage services	40
6.9. Query MSS operation records	40
7. API reference	43
7.1. Android API	43
7.2. iOS API	48

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

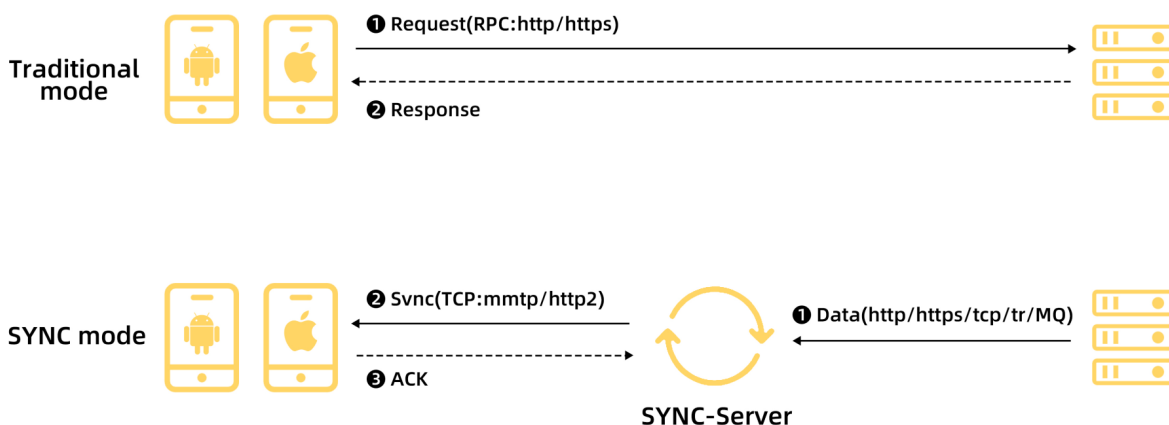
Mobile Sync Service (MSS) is a core basic business component of the mPaaS platform. MSS originates from the E2E solution SYNC of Ant Financial Group, which is oriented to mobile Apps and pushes massive data from the server to the client. This component provides a secure data channel based on the Transmission 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.

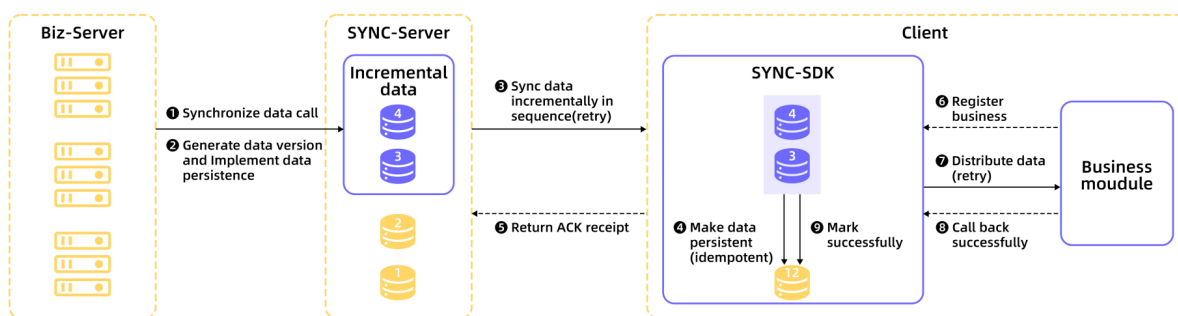
- Highly real-time performance

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

<code>appToForeground()</code>	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.
<code>appToBackground()</code>	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.
<code>updateUserInfo(String sessionId)</code>	Call this API when the login information (userId or sessionId) is changed. This API is called at least once.
<code>clearUserInfo()</code>	Clears user information when a user logs out.
<code>registerBiz(String bizType, ISyncCallback syncCallback)</code>	You can call this API to register a <code>callback</code> to receive business data. If this API is called, the client SDK will call the <code>syncCallback</code> class after receiving synchronized data.
<code>unregisterBiz(String bizType)</code>	Unregisters a specified synchronization configuration. If this API is called, the client SDK will not call the <code>syncCallback</code> class when receiving synchronized data.
<code>reportMsgReceived(SyncMessage syncMessage)</code>	After the data is received in the <code>syncCallback</code> implementation class, this API is called to notify MSS that the sync data has been received. Before receiving the <code>reportMsgReceived</code> message, MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data will be permanently deleted.
<code>isConnected()</code>	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
    // client 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.

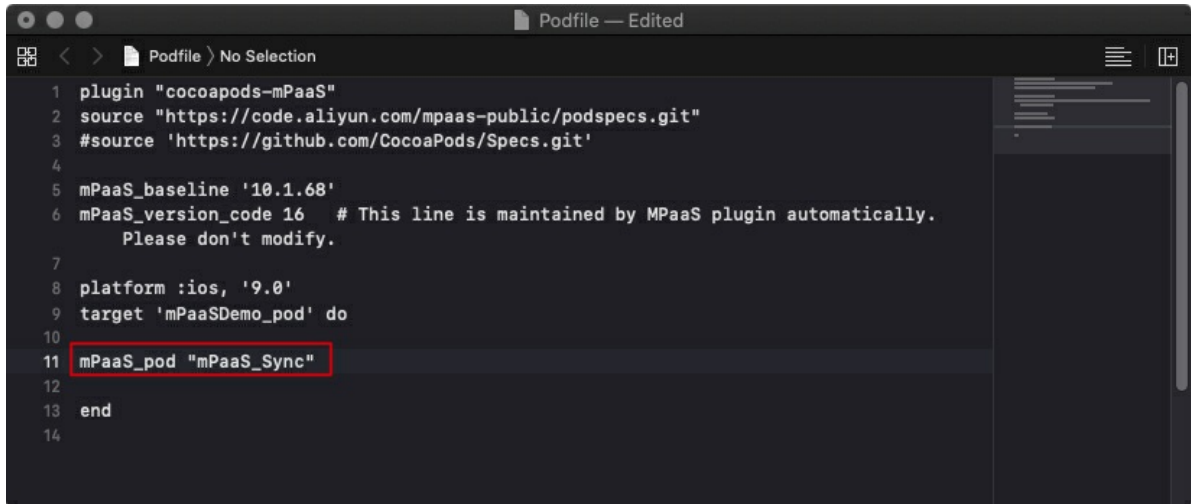
Prerequisites

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.



```
1 plugin "cocoapods-mPaaS"
2 source "https://code.aliyun.com/mpaas-public/podsspecs.git"
3 #source 'https://github.com/CocoaPods/Specs.git'
4
5 mPaaS_baseline '10.1.68'
6 mPaaS_version_code 16 # This line is maintained by MPaaS plugin automatically.
7   Please don't modify.
8
9 platform :ios, '9.0'
10 target 'mPaaS Demo Pod' do
11   mPaaS_pod "mPaaS_Sync"
12 end
13 end
14
```

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.

```
MPSyncDemo_plugin > MPaaS > mpaas_sdk.config > No Selection
10  "APLongLinkService": {
11      "APLongLinkService": "1.0.0.190114154233",
12      "APLog": "3.0.2.190226105327",
13      "APProtocolBuffers": "1.0.1.190226124537",
14      "mPaas": "1.0.0.190321160009",
15      "MPDataCenter": "1.0.0.190312145215",
16      "SecurityGuardSDK": "2.2.3.190326110928",
17      "UTDID": "1.0.2.190226130141",
18      "MPMssAdapter": "1.0.0.190226204648"
19  },
20  "MPBaseTest": {
21      "MPBaseTest": "1.0.0.190226113850"
22  },
23  "APCommonUI": {
24      "APCommonUI": "1.0.0.190226202548",
25      "AntUI": "1.0.0.190320123641",
26      "AntUIShellForMpass": "1.0.0.190226202548",
27      "MPBadgeService": "1.0.0.181024211440"
28  }
29  },
30  "baseline": "10.1.32",
31  "frameworks": [
32      "APLongLinkService.framework",
33      "APLog.framework",
34      "APProtocolBuffers.framework",
35      "mPaas.framework",
36      "MPDataCenter.framework",
37      "SecurityGuardSDK.framework",
```

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.

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 llOnceToken;

    dispatch_once(&llOnceToken, ^{

        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:nil];
    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. Access the server

5.1. Integrate into MSS

Integrating the business system of a private cloud tenant into a Mobile Sync Service (MSS) server involves two steps: **Configure services** and **Write the call code and process the results**. In business scenarios that have high-security requirements for data synchronization, you must perform user consistency verification after the integration. The step of writing the call code and processing the results is described from the perspective of the single data synchronization API and the global data synchronization API.

Note

In a private cloud, the intranet within your IDC is used. Your business system can be directly connected to the MSS server deployed in the private cloud, without using the OpenAPI system provided by Ant Technology. Modify the domain (IP) address for API calls based on the deployment environment.

Prerequisites

1. You have already deployed the MSS service in a private cloud. All health checks and port checks have passed.
2. You have created an app and obtained the app ID and workspace ID. You understand the client integration progress.
3. You have a server-side application that can initiate HTTP calls.

Configure services

The MSS server communicates with the client based on **BizType**. **BizType** is the core attribute for logic separation between different business data of the same app in an environment. Before you call the server-side API, you must perform push configurations. For more information, see [Add configurations](#).

Write the call code and process the results

The process of writing the call code and processing results varies with synchronization APIs, including the single data synchronization API and the global data synchronization API. For more information, see the following two topics:

- [Use the single data synchronization API](#)
- [Use the global data synchronization API](#)

5.2. Single data synchronization API

You can call this API to synchronize one piece of data to specified users or devices.

- If the target user or device is online, the pushed data is directly synchronized to the terminal.
- However, if the target user or device is offline, the pushed data is forwarded and stored in the database. After the terminal goes online and establishes a connection with the Mobile Sync Service (MSS), the pushed data is then synchronized to the terminal. For more information about data persistence operations, see [Add configurations](#).

API URL example

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

- IP address (domain name): Change it to the IP address of your on-premises MSS server.
- instanceId: The value format is workspaceId_AppId. `instanceID` is the core attribute for separating the app data logic in the MSS.

Code sample

```
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;

/**
 * Single push, which uses the httpclient to simulate a POST request and sends the
 * request to a specified server. You only need to modify the server IP address in apiURL.
 */
public class SingleTest {
    private static String apiURL = "http://11.162.169.36/webapi/sync/single?
instanceId=sit_320C94C171133";

    /**
     * POST request
     *
     * @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");
            // Set contentType, which is required for sending JSON data.
            s.setContentType("application/json");
            httpPost.setEntity(s);
            CloseableHttpResponse res = httpClient.execute(httpPost);
            if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
                HttpEntity entity = res.getEntity();
                // Return data in JSON format.
                String result = EntityUtils.toString(entity);
                response = JSONObject.fromObject(result);
            }
        }
    }
}
```

```
    } catch (Exception e) {
        throw new RuntimeException(t);
    }
    return response;
}

public static void main(String[] args){
    JSONObject params = new JSONObject();
    //bizType
    params.put("bizType", "UCHAT");
    // User ID or device ID.
    params.put("linkToken", "2088102147396225");
    // Message body, which is transparently sent to the client.
    JSONObject payload = new JSONObject();
    payload.put("name", "James");
    payload.put("age", "30");
    payload.put("address", "Hangzhou, Zhejiang");
    // String.
    params.put("payload", JSON.toJSONString(payload));

    // Business ID, which can contain business rules.A maximum of 100 characters i
s allowed. In this demo, a random value is used.
    Double i = Math.random() * 1000000;
    String[] num = i.toString().split("\\.");
    params.put("thirdMsgId", "test_third_msg_id_" + num[0] +
System.currentTimeMillis());

    // Earliest client version supported.
    params.put("appMinVersion", "0.0.0.0");
    // Latest client version supported.
    params.put("appMaxVersion", "100.100.100.100");

    // Current time.
    long today = System.currentTimeMillis();
    // Valid for 30 days.
    long validTimeEnd = today + 30 * 24 * 60 * 60 * 1000L;
    // Validity period.
    params.put("validTimeEnd", validTimeEnd);

    // Submit a POST request.
    JSONObject ret = doPost(apiURL, params);
    System.out.println(ret);
}
}
```

Request parameter

Parameter	Required	Data type	Maximum length	Description
-----------	----------	-----------	----------------	-------------

bizType	Yes	String	30	The business scenario ID. The value must be the same as that of Sync ID set in the mPaaS console. For more information, see Manage the console .
linkToken	Yes	String	100	Set this parameter based on the value of bizType. Set it to the value of userId for user-based business and the value of utdid for device-based business.
payload	No	String	4096	The pushed data, which is a text string or a JSON character string. You must transfer this parameter or the <code>binaryPayload</code> parameter. You cannot leave both parameters empty.

thirdMsgId	Yes	String	100	The ID of a data synchronization request, customized by you for data association and idempotent control. The result of the combination of <code>bizType</code> , <code>linkToken</code> , and <code>thirdMsgId</code> must be unique. Otherwise, the newly pushed data will not be saved to the database and a successful call response is returned.
binaryPayload	No	String	4096	The pushed data, which is a string generated from the original byte array by using the Base64 encoding. You must set this parameter or the payload parameter. Do not leave both empty.
osType	No	String	10	Operating system (iOS or Android) running on the client to which messages are to be pushed. Leave this parameter empty if you do not want to restrict the operating system.

appMinVersion	No	String	20	Earliest client version supported. Messages are pushed only to clients of the specified version or later versions. An example value is 8.6.0.9999. We strongly recommend that you use the version number in the standard format.
appMaxVersion	No	String	20	Latest client version supported. Messages are pushed only to clients of the specified version or earlier versions. An example value is 9.0.0.9999. We strongly recommend that you use the version number in the standard format.
validTimeEnd	No	long	13	End time of the push validity period, after which the MSS server will no longer push expired data to the client. Format: <code>(new Date()).getTime();</code>

Response parameters

The returned data is in JSON format, and the code sample is as follows:

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

Attribute description:

Parameter	Type	Example	Description
success	boolean	true or false	The business call result. Valid values: <code>true</code> and <code>false</code> , where <code>true</code> indicates a successful call and <code>false</code> indicates a failed call. If <code>false</code> is returned, check the value of <code>msg</code> to locate the cause. For more information, see the following Result code table.
msg	String	SUCCESS	Result code.

Result code:

Call result	Result code	Description
true	SUCCESS	The business is successful. Data has been pushed online.
true	DUPLICATED_BIZ_ID	<code>bizId</code> is duplicated. The data push is successful.
true	NOT_ONLINE	The user or device is offline.
false	THIRDMSGID_IS_NULL	<code>thirdMsgId</code> is not specified.
false	BIZ_NOT_ONLINE	The synchronization configuration is not submitted online.
false	ARGS_IS_NULL	A required input parameter is not specified.
false	NOT_SUPPORT_GLOBAL	This API does not support global services.

false	PAYLOAD_LONG	The length of the message body has exceeded the upper limit.
false	THIRD_MSG_ID_LONG	The length of the third-party message ID has exceeded the upper limit.
false	SYSTEM_ERROR	A system error occurred.

5.3. Global data synchronization API

You can call this API to synchronize one piece of data to all users or devices in all networks. The pushed data will be persisted into the database. For more information about configuring data persistence, see [Manage the console](#). Considering the cost-benefit-risk tradeoffs, the pushed data will not be immediately synchronized to users of the application that is currently in the foreground. Instead, when the application switches to the foreground next time, the MSS client SDK will trigger the synchronization process.

API URL example

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

- IP address (domain name): Change it to the IP address of your on-premises MSS server.
- instanceID: The value format is workspaceId_AppId. `instanceID` is the core attribute for separating the app data logic in the MSS.

Code sample

```
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;

/**
 * Global push, which uses the httpclient to simulate a POST request and sends the
 * request to a specified server. You only need to modify the server IP address in apiURL.
 */
public class GlobalTest {
    private static String apiURL = "http://11.160.18.15/webapi/sync/global?
instanceId=default_99FB626081956";

    /**
```

```

    * POST request
    *
    * @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");
        // Set contentType, which is required for sending JSON data.
        s.setContentType("application/json");
        httpPost.setEntity(s);
        //httpPost.set
        CloseableHttpResponse res = httpClient.execute(httpPost);
        if (res.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            // Return data in JSON format.
            String result = EntityUtils.toString(res.getEntity());
            response = JSONObject.fromObject(result);
        }
    } catch (Exception e) {
        throw new RuntimeException(t);
    }
    return response;
}

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

    // Message body, which is transparently sent to the client.
    JSONObject payload = new JSONObject();
    payload.put("switchName", "showImage");
    payload.put("switchValue", true);

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

    // Business ID, which can contain business rules. A maximum of 100 characters is
    // allowed. In this demo, a random value is used.
    Double i = Math.random() * 1000000;
    String[] num = i.toString().split("\\.");
    params.put("thirdMsgId", "test_third_msg_id_" + num[0] +
    System.currentTimeMillis());

    // IOS or ANDROID. Do not transfer this parameter if you do not want to restrict
    // the operating system.
    params.put("osType", "IOS");
}
```

```
// Earliest client version supported. Do not transfer this parameter if you do
not want to restrict the version.
params.put("appMinVersion", "0.0.0.0");
// Latest client version supported. Do not transfer this parameter if you do no
t want to restrict the version.
params.put("appMaxVersion", "100.100.100.100");

// Current time.
long today = System.currentTimeMillis();
// Valid for 30 days.
long validTimeEnd = today + 30 * 24 * 60 * 60 * 1000L;
// Validity period.
params.put("validTimeEnd", validTimeEnd);

// Submit a POST request.
JSONObject ret = doPost(apiURL, params);
System.out.println(ret);
}
}
```

Request parameter

Parameter	Required	Data type	Maximum length	Description
bizType	Yes	String	30	The business scenario ID. The value must be the same as that of Sync ID set in the Mobile PaaS (mPaaS) console. For more information, see Manage the console .
payload	No	String	4096	The pushed data, which is a text string or a JSON character string. You must transfer this parameter or the <code>binaryPayload</code> parameter. You cannot leave both parameters empty.

thirdMsgId	Yes	String	100	<p>The ID of a data synchronization request, customized by you for data association and idempotent control. The result of the combination of <code>bizType</code> , <code>linkToken</code> , and <code>thirdMsgId</code> must be unique. Otherwise, keep the data previously used for a successful API call, discard the new data, and a successful call response is returned.</p>
binaryPayload	No	String	4096	<p>The pushed data, which is a string generated from the original byte array by using the Base64 encoding. You must transfer this parameter or the <code>payload</code> parameter. You cannot leave both parameters empty.</p>

osType	No	String	10	Operating system (iOS or Android) running on the client to which messages are to be pushed. Leave this parameter empty if you do not want to restrict the operating system.
appMinVersion	No	String	20	Earliest client version supported. Messages are pushed only to clients of the specified version or later versions. An example value is 8.6.0.9999. We strongly recommend that you use the version number in the standard format.

appMaxVersion	No	String	20	Latest client version supported. Messages are pushed only to clients of the specified version or earlier versions. An example value is 9.0.0.9999. We strongly recommend that you use the version number in the standard format.
validTimeEnd	No	long	13	End time of the push validity period, after which the MSS server will no longer push expired data to the client. Format: <code>(new Date()).getTime();</code>

Response parameters

The returned data is in JSON format, and the code sample is as follows:

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

Attribute description:

Parameter	Type	Example	Description
-----------	------	---------	-------------

success	boolean	true or false	The business call result. Valid values: <code>true</code> and <code>false</code> , where <code>true</code> indicates a successful call and <code>false</code> indicates a failed call. If <code>false</code> is returned, check the value of <code>msg</code> to locate the cause. For more information, see Result code.
msg	String	SUCCESS	The result code.

The following table describes the result codes:

Result	Result code	Description
true	SUCCESS	The business is successful. Data has been pushed online.
true	DUPLICATED_BIZ_ID	<code>bizId</code> is duplicated. The business is successful.
false	THIRDMSGID_IS_NULL	<code>thirdMsgId</code> is not specified.
false	BIZ_NOT_ONLINE	The synchronization configuration is not submitted online.
false	ARGS_IS_NULL	A required input parameter is not specified.
false	NOT_SUPPORT_GLOBAL	This API does not support global services.
false	PAYLOAD_LONG	The length of the message body has exceeded the upper limit.
false	THIRD_MSG_ID_LONG	The length of the third-party message is too long.
false	SYSTEM_ERROR	A system error occurred.

5.4. Verify user consistency

In some cases, the business has high-security requirements for data synchronization. Make sure that the target users of the push are the current logon users and have not been forged. For that, Mobile Sync Service (MSS) provides user consistency verification, which can be turned on by the user when needed.

The general principle of this feature is:

- The client reports `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 API implemented by the tenant. The tenant verifies the consistency by checking `userId` and authorization token (`sessionId`) through the API. MSS records a consistency identifier for determining the session status during subsequent data synchronization. When the MSS server determines that a session is valid, it will synchronize data to the target user. When it determines that a session is invalid, it will not synchronize data to the target user.
- 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. For more information about data synchronization configuration, see [Manage the console](#).

Configure the user consistency verification API

Configure the RPC API for verifying the logon session in the Mobile PaaS (mPaaS) console.

Operation page

In the selected app, choose **Backend service management** > **Mobile gateway service**, and configure the API. For more information, see [Configure an API](#).

API name

The MSS server regularly calls this API. Therefore, you must set `operationType` of the API to be added to `com.antcloud.session.validate`. In addition, the API must have the following request parameters.

Parameter	Required	Data type	Maximum length	Description
<code>instanceId</code>	Yes	String	100	The business scenario identifier. The value format is <code>workspaceId_AppId</code> .
<code>userId</code>	Yes	String	100	The unique identifier of the user.
<code>sessionId</code>	Yes	String	100	Authorization token carried by the client.

Response

To check the custom consistency verification logic in the system, the data returned must be in JSON format.

- Parameter description

Parameter	Type	Example	Description
success	boolean	true or false	The business call result. Valid values: <code>true</code> and <code>false</code> , where <code>true</code> indicates a successful call and <code>false</code> indicates a failed call. If <code>false</code> is returned, use <code>returnCode</code> to locate the cause. For more information, see the result codes in the following table.
returnCode	String	SUCCESS	The result code.
resultMsg	String	SUCCESS	Result information.
result	JSONString	{ "sid": "kkdddd", "valid": "true/false", }	The result object. For parameter descriptions, see the result codes in the following table.
sid	String	kkdddd	The authorization token or sessionId.
valid	boolean	true or false	The consistency verification result.

- Result codes

Result	Result code	Description
true	OK	The business is successful.
false	OPERATION_ERROR	The operation fails. Only the <code>com.antcloud.session.validate</code> API is called.

- Code sample

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

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 scenario 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 configurations](#)
- [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.

6.9. Query MSS operation records

Mobile Sync Service (MSS) allows you to query operation records of data synchronization, including the operation change records of creating a configuration, modifying a configuration, deleting a configuration, creating a synchronization, publishing a configuration, disabling a configuration, enabling a signature, and disabling a signature. MSS is used for operation traceability and usage data analysis.

To query the operation records, perform the following steps.

1. Log on to the mPaaS console to enter the target application, and click **MSS** in the left-side navigation pane to enter the MSS page.
2. On the right-side **Operation record** tab, filter the operation records by operating account, operating type, or operating date. After you set the filter criteria, click **Search**, and the search results will be shown in the operation record list in descending order. The result includes the operation account, operation type, and specific operation time (accurate to second) for each operation.

 **Note** If you do not specify a time range, a maximum of 1,000 operation change records in the latest 7 days are shown by default.



操作者	操作动作	操作时间	操作
mpaas_public_test	新建同步	2021-06-28 10:54:44	详情

3. Select the operation you want to view and click the right-side **Details** link to enter the operation details page to view the specific information of the operation.
 - **Create configuration**: shows the details of the new synchronization configuration.

配置新增详情 ×

基础配置

同步标识: 88 推送说明: 88 推送范围: 指定推送

推送对象

推送对象: 用户 多设备同步[?]: 是

数据持久化[?]

数据持久化: 是 重推方式: 阈值[?] 重推阈值: 8

安全控制

用户一致性[?]: 否

关闭

- **Modify configuration:** shows the modified configuration information and highlights the modified information in red.

配置修改详情 ×

基础配置

同步标识: deviceSync1 推送说明: 设备全局同... 推送范围: 全局推送

推送对象

推送对象: 设备 多设备同步[?]: 否

数据持久化[?]

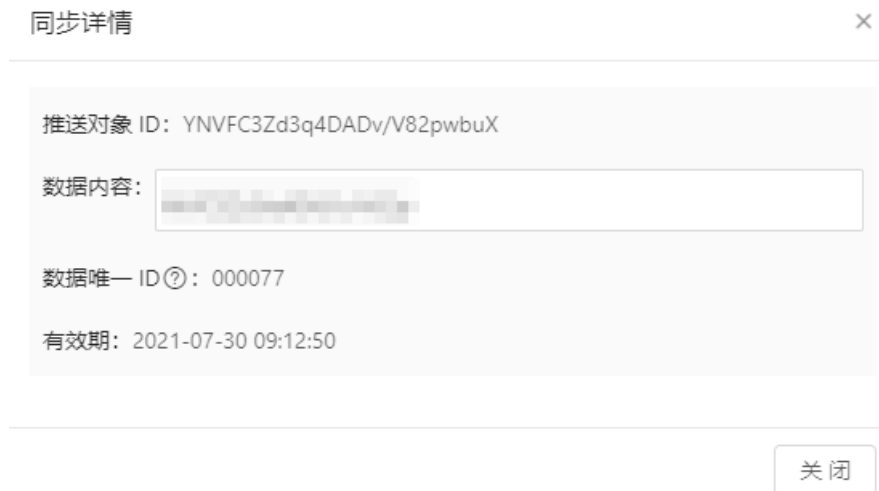
数据持久化: 是 重推方式: 阈值[?] 重推阈值: 10

安全控制

用户一致性[?]: 否

关闭

- **Delete configuration:** shows the deleted synchronization configuration.
- **Create sync:** shows the details of the new synchronization task, including push object ID, data content, unique data ID, app version interval of the client, and push data validity.



- **Publish configuration:** shows the details of synchronization configuration after disabling a configuration and publishing it again. The details of the changes shown are the same as creating a configuration.
- **Disable configuration:** shows the details of the disabled synchronization configuration.
- **Disable signature:** shows the operation records of disabling a signature.
- **Enable signature:** shows the operation records of enabling a signature.

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\(SyncMessage syncMessage\)](#)
- [unregisterBiz\(String bizType\)](#)
- [updateUserInfo\(String sessionId\)](#)

Return value type	Methods and description
void	<code>setup(Application application)</code> Initializes basic services on which MSS depends. Call this API before you call the <code>initialize</code> method. This function is available only in baseline 10.1.60 and later versions.
void	<code>appToBackground()</code> 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	<code>appToForeground()</code> 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	<code>clearUserInfo()</code> Clears user information when a user logs out.
void	<code>initialize(Context context)</code> Initializes MSS.
boolean	<code>isConnected()</code> Checks whether MSS is running properly.
void	<code>registerBiz(String bizType, ISyncCallback syncCallback)</code> Registers a callback to receive business data. If this API is called, the client SDK will call the <code>syncCallback</code> class after receiving synchronized data.
void	<code>reportMsgReceived(SyncMessage syncMessage)</code> Notifies MSS of the data synchronization success after data is received in the <code>syncCallback</code> class. Before receiving <code>reportMsgReceived</code> , MSS attempts to resend the data for a maximum of six times. If all resending attempts fail, the data is permanently deleted.
void	<code>unregisterBiz(String bizType)</code> Unregisters a specified synchronization configuration. If this API is called, the client SDK will not call the <code>syncCallback</code> class when receiving synchronized data.
boolean	<code>updateUserInfo(String sessionId)</code> Call this API at least once when the login information (<code>userId</code> or <code>sessionId</code>) 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 <code>Context</code> .

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.