Ant Technology

Mobile Analysis Service
User Guide

Document Version: 20250731

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.About Mobile Analysis Service 08
2.Terminology 10
3.Quick start guide 11
4.Integrate MAS into Android 14
4.1. Quick start 14
4.2. Add logs 16
4.2.1. Active device/user log 16
4.2.2. Performance log 17
4.2.3. Crash log 19
4.2.4. Custom event log 19
4.2.5. Automation log 20
4.3. View local logs 23
4.4. Upload logs 24
5.Integrate MAS into iOS 26
5.1. Instructions on MAS integration procedure 26
5.2. Add SDK 26
5.3. Use SDK 26
5.3.1. Configure project 26
5.3.2. Add activity report log 27
5.3.3. Add performance log 28
5.3.4. Add crash log 32
5.3.5. Add custom event log 33
5.3.6. Add automatic log 34
5.4. View local logs 35
5.5. Upload logs 36
6.Basic analysis 38

6.1. Data overview 38
6.1.1. Introduction to data overview 38
6.1.2. Real-time dashboard 38
6.1.3. Historical trends 39
6.2. Behavior analysis 40
6.3. Retention analysis 42
6.4. Page analysis 42
6.5. Device analysis 44
6.6. Component usage analysis 44
6.6.1. Hotfix analysis 44
6.6.2. Offline package analysis 45
6.6.3. Scan analysis 46
6.6.4. Mini Program analysis 47
6.7. Page configuration 48
6.8. Calculation rules for indicators 49
7.Custom analysis 55
7.1. Custom dashboard 55
7.1.1. Create custom dashboards 55
7.1.2. Manage dashboards 56
7.2. User group management 58
7.2.1. About user group 58
7.2.2. Create user group 58
7.2.3. Manage user group 60
7.3. About custom analysis 61
7.4. Configure custom analysis 62
7.5. Configure attributes 63
7.6. Configure events 64
7.7. Event analysis 65

    7.7.1. Add event analysis 65
    7.7.2. Manage event analysis 66
  7.8. Funnel analysis 68
    7.8.1. About funnel analysis 68
    7.8.2. Create funnels 69
    7.8.3. Manage funnels 70
  7.9. Trajectory analysis 71
8.Performance analysis 73
  8.1. Crash report 73
  8.2. Lag report 75
  8.3. Stuck report 76
  8.4. iOS symbol table management 78
9.Log management 80
  9.1. Extract real-time logs 80
  9.2. Query history logs 80
  9.3. Configure upload switch 82
  9.4. Client diganosis 85
    9.4.1. About client diagnosis 85
    9.4.2. Android client diagnosis 86
    9.4.3. iOS client diagnosis 91
      9.4.3.1. Add SDK 91
      9.4.3.2. Use SDK 92
  9.5. Tracking log model 95
    9.5.1. Log tracking 95
    9.5.2. Android/iOS custom event tracking 96
    9.5.3. Android/iOS behavior tracking 102
    9.5.4. Android/iOS performance tracking 120
    9.5.5. HTML5/PC page tracking 138

9.5.6. HTML5/PC click tracking 141
9.5.7. HTML5/PC exposure tracking 145
10.H5 common event tracking 149
10.1. Common tracking types 149
10.2. Configure common tracking 151
10.3. Analyze common tracking logs 153
11.Tutorial 155
11.1. Custom event analysis 155
11.1.1. About this tutorial 155
11.1.2. Android client development 156
11.1.3. iOS client development 157
11.1.4. Create attributes 159
11.1.5. Create an event 159
11.1.6. View event PV and UV 160
11.1.7. Add a custom analysis 160
11.1.8. Add a dashboard 160
11.1.9. Related steps 161
11.2. Funnel analysis 161
11.3. Application startup speed analysis 162
11.4. Crash analysis 163
11.5. Stuck analysis 164
11.6. Lag analysis 165
11.7. Active device/user analysis 166
12.FAQ 167
13.Reference 169
13.1. User ID 169

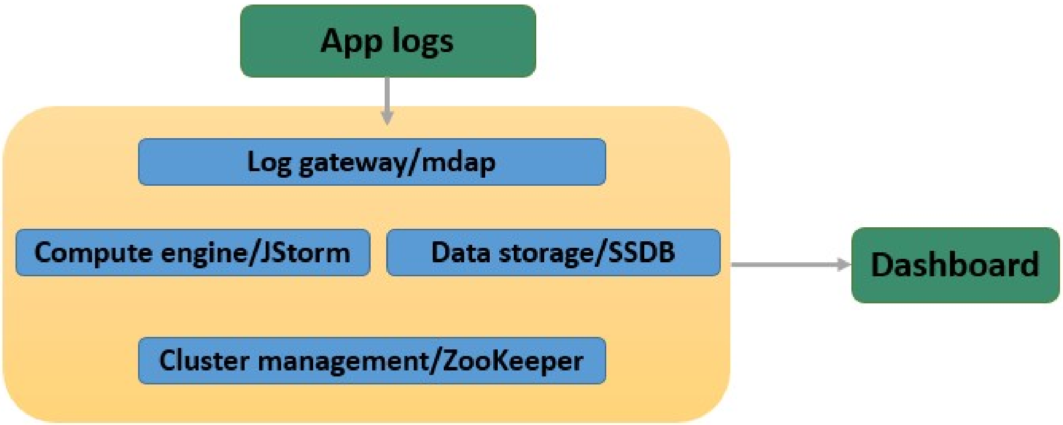
1.About Mobile Analysis Service

The Mobile Analysis Service (MAS) component collects and analyzes the data from client traffic, performance, and user behavior. By using the data that the MAS component provides, you can make informed decisions on products, operations, and promotion campaigns. For example, by reading the crash report that the MAS component provides, you can quickly identify the cause of the crash so that you can improve your system stability.

Principle of component

Introduction to the relevant components involved in MAS:

- **mdap**: Log collection gateway, responsible for collecting client's event logs and transmitting the collected logs to JStorm cluster for computing.
- **JStorm**: Real-time computing engine, perform real-time parsing on logs according to processing rule and store the required data in database.
- **SSDB**: KV (key value) data storage layer with leveldb at bottom layer, with single-table data reaching billions of records.
- **ZooKeeper**: Cluster management, inter-component service discovery.



Features

- **Easy access**: Automatically collecting user behavior logs, network logs, and exception logs.
- **Comprehensive analysis**: Conducting analysis, for example, on user behavior, terminal, traffic, communication links, performance.
- **Multi-dimensional display**: Displaying mobile analytics from multiple dimensions such as terminal type, terminal version, region, network types, and models.
- **Quickly locate problems:** Providing crash and exception logs information such as the interface name where the error occurred, the cause of the exception, and the operating environment, helping developers quickly locate problems.

Functions

- **User behavior analysis**: Providing application usage analysis, including the statistics on active users, user login, new users and other indicators, and support multi-dimensional analysis of and comparison between platforms, versions, regions and time periods, helping you understand your App quickly and conveniently.

- **Stability analysis**: Providing application stability analysis, including crash monitoring, exception monitoring, performance monitoring and user diagnosis, helping App developers to timely find problems and locate the causes.
- **Problem diagnosis**: Providing application problem diagnosis, including individual user diagnosis and diagnostic log collection. Individual user diagnosis refers to obtaining users' client-side behavior in real time; diagnostic log collection works by pushing commands to client to request the client to return local logs.

Scenarios

- **Guide business based on data**: Helping development and operations staff make decisions on products, operations and promotion plans based on mobile analytics.
- **Improve user experience**: Fast determining the location of crash and applying hotpatch to fix the crash rapidly, thus improving user experience and enhancing retention rate.

2.Terminology

The terms are listed in ascending alphabetical order.

Tracking

Tracking refers to the method for capturing, processing and reporting specific user behaviors or events. In the process of implementing event trackig, some information, such as PV, UV, clicks and retention period, is collected from application to track the application usage, and subsequently used as reference data for further product optimization and marketing and operation.

Event

An event records an action performed by a user in the App. You can set up a custom event tracking to log any action (such as button click).

Event ID

An event ID uniquely identifies an event. Events are global in the App. Therefore, event IDs must be unique within the same mPaaS App.

Attribute

An event contains some information, for example, user ID, App version, device model, region and language. These information are known as attributes. MAS provides some preset common attributes. You can also customize some attributes applicable to the entire App based on your business requirements, and use them when configuring specific events.

Attribute ID

An attribute ID uniquely identifies an attribute. Attributes are global in the app. Therefore, attribute IDs must be unique within the same mPaaS App.

Event analysis

Events and their attributes are stored as logs on the local client and reported to the Mobile Analysis server. After finishing related configuration and operations in the console, you can view event analysis reports.

3.Quick start guide

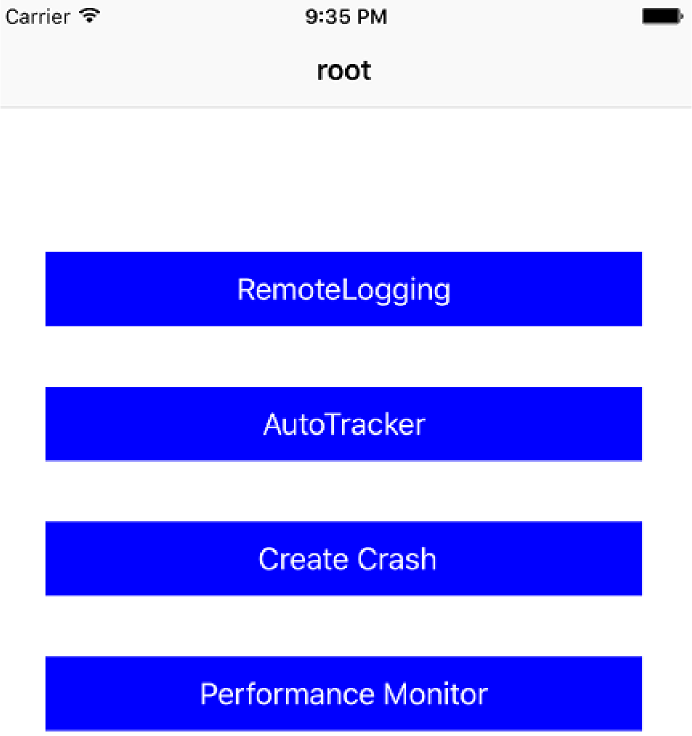
The guide walks you through the steps to run the App for mobile analysis and view the reports generated by MAS.

Procedure

1. Log in to the mPaaS console and create an App.
2. Integrate the event SDK in client, and implement event tracking. For how to integrate the event SDK, see Instructions on MAS integration procedure or Quick start.
3. In the mPaaS console, from the navigation bar on the left, click **Mobile Analysis Service** to view the statistical data.
4. If the iOS App needs crash symbol parsing, you need to upload the corresponding dSYM symbol through the mPaaS console **Mobile Analysis Service** > **Performance Analysis** > **iOS Symbol Table Management**. Symbol reverse parsing is done in real time when the crash log is uploaded with minutes' delay. If no symbol file is uploaded after the App is released, the original log content is displayed by default.

Start and run App

Run the App directly. In the App, when you click a button, switch pages or a crash occurs, an event log is generated and synchronized to the server that the client reports the log to. Then, you can view the analysis reports based on the event data from the Mobile Analysis page on the mPaaS console.



View analysis reports

You can log into the Mobile Analysis Service console to view the analysis reports:

1. Log into mPaaS console, and select an App.

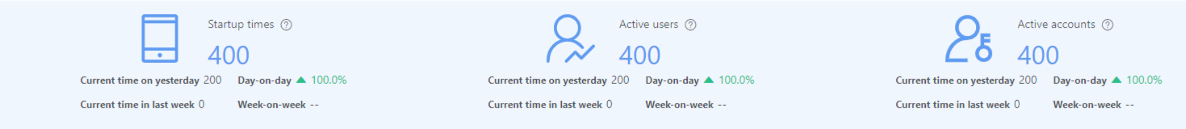
Log into the Console. On the upper left corner of the page, select **Products and Services** > **Mobile PaaS**. Click the App name from the App list.

2. On the left navigation bar, click **Mobile Analysis Service**, then you can see **Basic analysis**, **Performance analysis**, **Component usage analysis**, and other tabs.

The examples of mobile analysis reports are as follows:

- Startup count, Active users, Active accounts, and New users

  You can view the Startup times, Active users, and Active accounts on the **Data overview** > **Real-time dashboard** page.



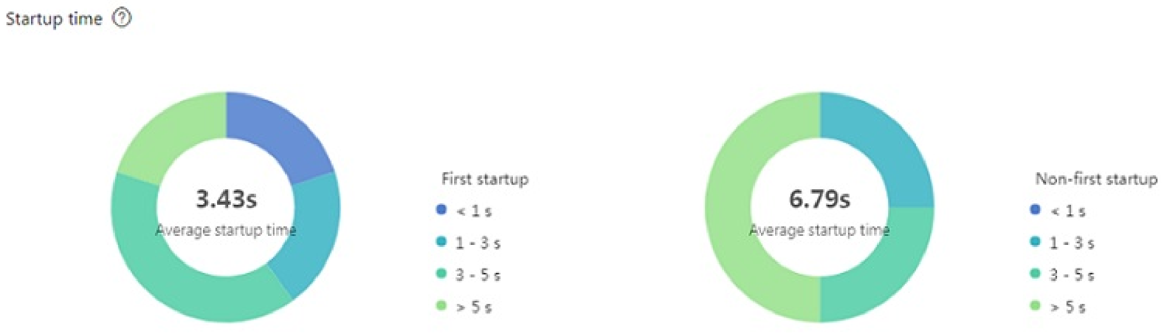
Startup times
400
Current time on yesterday 200
Day-on-day ▲ 100.0%
Current time in last week 0
Week-on-week --

Active users
400
Current time on yesterday 200
Day-on-day ▲ 100.0%
Current time in last week 0
Week-on-week --

Active accounts
400
Current time on yesterday 200
Day-on-day ▲ 100.0%
Current time in last week 0
Week-on-week --

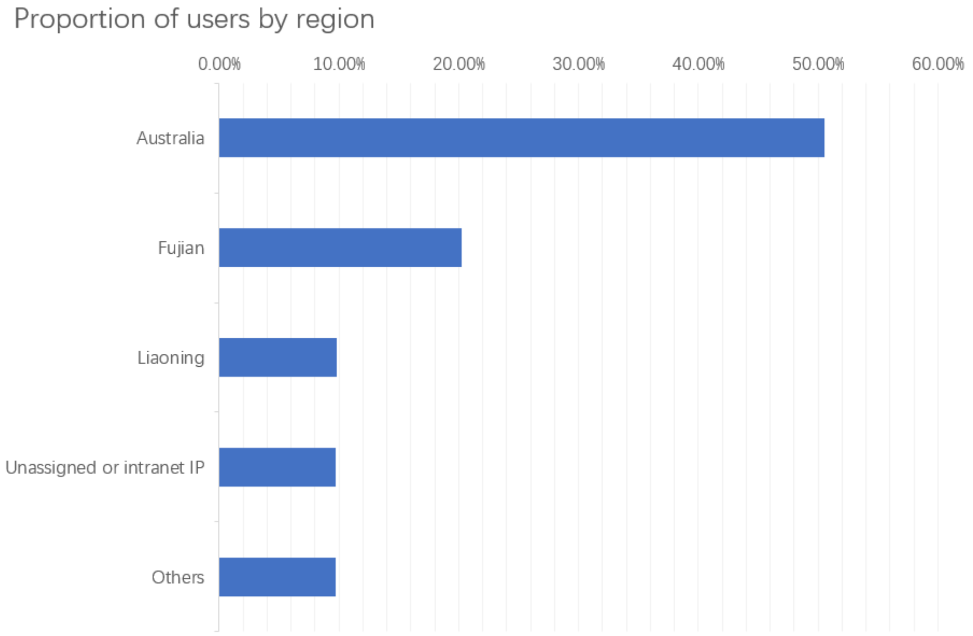
- Startup speed

  You can view the startup speed of App on the **Basic analysis** > **Behavior analysis** page.



- Users by region

  You can view the users by region on the **Basic analysis** > **Behavior analysis** page.

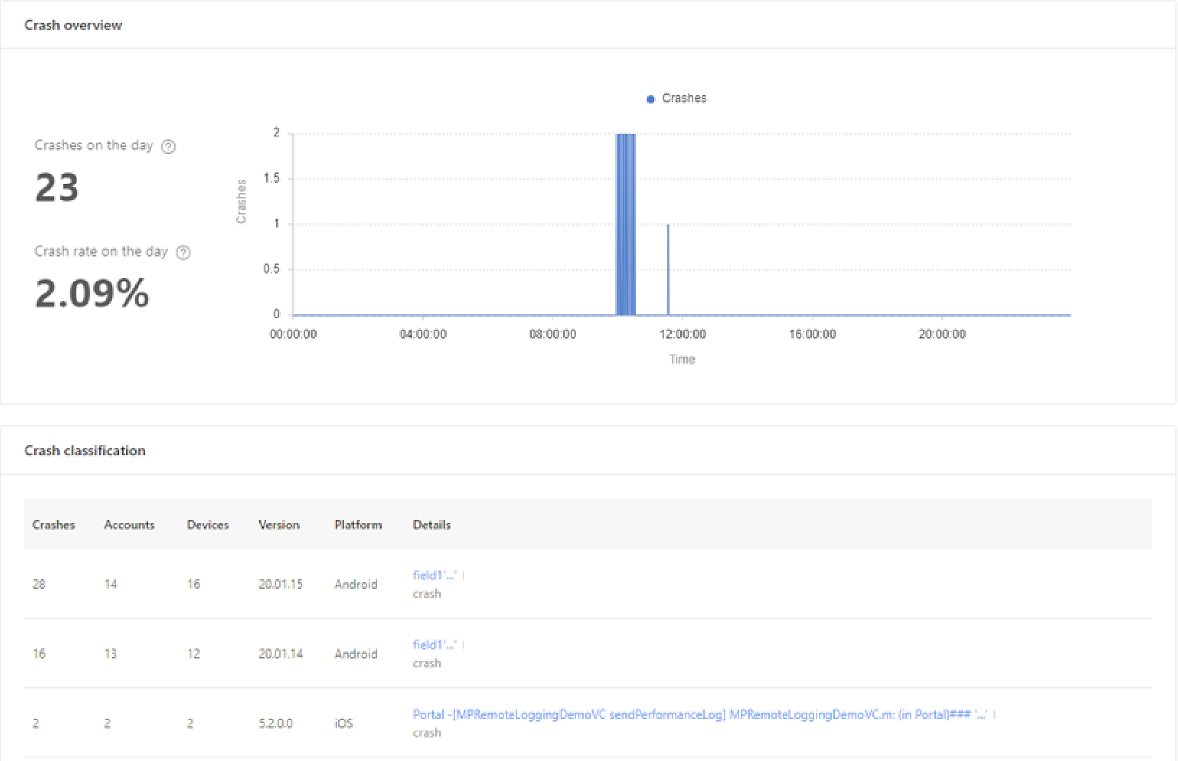


- Retention analysis

  You can view the retention analysis data on the **Basic analysis** > **Retention analysis** page.

- Crash report

  You can view the crash report on the **Performance analysis** > **Crash report** page.



Crash overview

Crashes on the day

23

Crash rate on the day

2.09%

Crashes
2
1.5
1
0.5
0
Crashes
00:00:00
04:00:00
08:00:00
12:00:00
16:00:00
20:00:00
Time

Crash classification

Crashes	Accounts	Devices	Version	Platform	Details
28	14	16	20.01.15	Android	field1'...' crash
16	13	12	20.01.14	Android	field1'...' crash
2	2	2	5.2.0.0	iOS	Portal -[MPRemoteLoggingDemoVC sendPerformanceLog] MPRemoteLoggingDemoVC.m: (in Portal)### '...' crash

- Hotfix report

  You can view the hotfix report on the **Component use analysis** > **Hotfix analysis** page.

4.Integrate MAS into Android

4.1. Quick start

Mobile Analysis Service (MAS for short) depends on the client SDK to implement event tracking, collect user behavior, App performance and other related data to generate logs, and report the logs to the server. The server extracts valid data from the tracking logs uploaded by the client based on the negotiated tracking log format to monitor and analyze client metrics.

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

The complete integration process mainly includes the following six steps:

1. Add SDK
2. Initialize mPaaS
3. Set project
4. Add logs
5. View local logs
6. Upload logs

Prerequisites

- If you integrate MAS through Native AAR, ensure that you have Add mPaaS to your project
- If you integrate MAS through Portal & Bundle, ensure that you have completed the General steps.

Add SDK

Native AAR mode

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

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

Portal & Bundle mode

Install the **LOGGING** component in the Portal and Bundle projects through **Component management**. For more information, see General steps.

Install the **LOGGING** component in the Portal and Bundle projects through **Component management**. For more information, see General steps.

Initialize mPaaS

Native AAR

If you integrate MAS through Native AAR, you must initialize mPaaS.

Add the following codes in the object `Application` :

```
public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        // mPaaS initialization
        MP.init(this);
    }
}
```

For more details, see Initialize mPaaS.

Portal & Bundle mode

To integrate MAS through Portal & Bundle, you don't have to implement mPaaS initialization.

Set project

Log uploading

Network access is required to upload logs. You need to declare the following permissions in the `AndroidManifest` file.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Log diagnosis

To use the log About client diagnosis, you need to declare the following permission in the `AndroidManifest` file, and apply for the permission when the App runs on the Android 6.0+ devices.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

The diagnostic log will be saved to the SD card of the device. With above permission, you cannot obtain the diagnostic log.

Channel number setting

To distinguish the apk data of different channels in the mPaaS console, you can set channel number for the apk.

In the App project, create the `channel.config` file under the directory `assets` and modify the value of `channel_id`.

```
# Identify the current release channel
channel_id=alipay
```

Get IMEI / IMSI

On systems lower than Android 10, the IMEI and IMSI of the device will be obtained by default if the relevant permissions have been obtained. If you need to completely prohibit the behavior of obtaining such information, please add the following configuration to `AndroidManifest`:

```
<meta-data
    android:name="imei.switch"
    android:value="off" />
```

> **Note**
>
> It only takes effect on 10.2.3.6 and above baselines. After the configuration is added, mobile analysis, message push, and data synchronization will no longer obtain the device's IMEI and IMSI.

Add logs

You can add the following logs after integrating the SDK:

- Active report
- Performance log
- Crash log
- Custom event log
- Automation log

View local logs

View local logs to learn the local log information.

Upload logs

Synchronize the local files on a client to the log server. For specific operations, see Upload logs.

4.2. Add logs

4.2.1. Active device/user log

This topic describes how to add active device/user reporting log.

The logs are classified into two categories:

- **Device Install Report**: calculate the number of devices where the App is started.
- **Active User Report**: calculate the number of users who log in to the App.

After these logs are reported, you can view indicators such as the number of active users, number of new users, and number of active accounts on the **Mobile Analysis Service** > **Data overview** page.

Tracking of device install report

The mPaaS framework automatically reports the data of active devices, you don't have to handle it. To stop automatic log reporting, you just need to add the following content (only works in baseline 10.1.68.30 and later versions) in the file `AndroidManifest.xml` .

```
<meta-data
    android:name="report.launch.switch"
    android:value="off" />
```

After you stop automatic log reporting, you can customize the time to report logs, for example:

```
MPLogger.reportClientLaunch();
```

When an App returns to the frontend from the backend, another active device is counted if the interval since the last reporting exceeds the specified interval (30 minutes by default).

You can customize a reporting interval in milliseconds, for example:

```
MPLogger.setReportClientLaunchInterval(long interval);
```

If the active device reporting API is called several times within the reporting interval, only the first call takes effect.

Tracking of active user report

The code for reporting active users is as follows:

```
MPLogger.reportUserLogin(String userId);
```

`userId` uniquely identifies a user who logs in to your App. You can call the active user reporting API after successful user login or after you obtain `userId` in another way.

After you call the active user reporting API, `userId` is set in the code. That is, `userId` is set successfully. In addition, you can also set `userId` separately by calling the following method.

The method for setting a user ID is as follows:

```
MPLogger.setUserId(String userId);
```

After setting a `userId`, you can add the `userId` to a whitelist created under Mobile Delivery Service > Whitelist management in the mPaaS console.

When the user logs out, call `MPLogger.setUserId(null)` to delete the `userId` so as to ensure the accuracy of relevant data.

4.2.2. Performance log

This article describes how to add performance logs for Mobile Analysis Service.

The performance logs that Mobile Analysis Service accesses to Android include:

- Startup time log
- Lag log
- Stuck log

You can Log in to the mPaaS console and choose **Mobile Analysis Service** > **Basic analysis** to view the startup duration; and log in to the mPaaS console and choose **Mobile Analysis Service** > **Performance analysis** to view lag and stuck reports.

Startup time tracking

Application startup time = the time when this method is called - the time when the application starts.

It is recommended to call the following method in the `onCreate()` method of the home page `Activity` to track the startup speed.

```
MPLogger.reportLaunchTime(Context context);
```

Lag tracking

A lag is defined as the Android main thread taking more than 2.25 seconds to complete a method. The lag threshold varies depending on the APK package type:

- When the APK package is a debug package, the lag threshold is 0.75 seconds, so that you can find the potential lagging issues during debugging.
- When the APK package is a release package, the lag threshold is 2.25 seconds.

Enable lag monitoring

Method 1

The interface needs to inherit the `BaseActivity` , `BaseFragmentActivity` or `BaseAppCompatActivity` class provided by mPaaS. All interfaces that inherit these classes will automatically monitor lag.

Method 2

> **Important**
>
> This method is supported only in baseline 10.2.3.50 and later versions.

Manually call the relevant interface in the life cycle method of `Activity` , for example:

```
import android.app.Activity;
import android.app.Application;
import android.os.Bundle;

import com.mpaas.mas.adapter.api.MPLogger;

public class MPLifecycle implements Application.ActivityLifecycleCallbacks {

    private int mVisibleActivityCount = 0;
    private boolean isBackground = false;

    @Override
    public void onActivityCreated(Activity activity, Bundle bundle) {

    }

    @Override
    public void onActivityStarted(Activity activity) {
        mVisibleActivityCount++;
        if (isBackground) {
            isBackground = false;
            // Call it when the application returns to the foreground
            MPLogger.monitorAppForeground();
        }
    }

    @Override
    @Override
    public void onActivityResumed(Activity activity) {
        // Update Activity Context
        MPLogger.monitorActivityResumed(activity);
    }

    @Override
    public void onActivityPaused(Activity activity) {
    }

    @Override
    public void onActivityStopped(Activity activity) {
        mVisibleActivityCount--;
        if (mVisibleActivityCount <= 0) {
            isBackground = true;
            // Call it when the application returns to the background
            MPLogger.monitorAppBackground();
        }
    }

    @Override
    public void onActivitySaveInstanceState(Activity activity, Bundle bundle) {
    }

    @Override
    public void onActivityDestroyed(Activity activity) {
    }

}
```

When the APK is a debug package, the lag monitoring is full statistics; when the APK is a release package, the lag monitoring is sampling statistics with a sampling rate of 10%.

Stuck tracking

A stuck is ANR in the Android system, which usually means that the main thread is unresponsive for more than 5 seconds.

To enable stuck monitoring, please refer to the above article for details: Enable lag monitoring.

4.2.3. Crash log

Crash logs record crashes of Apps. You can log in to the mPaaS console and choose **Mobile Analysis Service** > **Performance analysis** to view crash reports.

The mPaaS framework automatically captures the crash information and uploads logs to the log server.

4.2.4. Custom event log

You can customize event tracking for user behavior analysis based on business needs.

After connecting a client to mPaaS, you need to log in to the mPaaS console, choose **Mobile Analysis Service** > **Custom analysis** > **Custom configurations**, and configure the relevant attributes and events. Then, you can choose **Custom analysis** > **Event analysis** to view relevant data. For more information, see Configure custom analysis.

The event tracking code is as follows:

```
MPLogger.event(String logId, String bizType, Map<String, String> params);
```

- `logId` : specifies the tracking ID, which cannot be empty. It corresponds to the **event ID** of the event created in the console.
- `bizType` : specifies the business type, which can be empty. The default value is `userbehavior` . Cannot contain underscore "_".
  - Custom logs with the same `bizType` are stored in a local file named `Timestamp_packagename-process_bizType` .
  - For more information about the naming rules for log files, see Log file name in View local logs.
- `params` : indicates extended parameters, which can be empty. The key in `params` corresponds to the **property ID** of a property created in the console and the value specifies the value of this property.

You can also use the following code for overloading:

```
MPLogger.event(String logId);

MPLogger.event(String logId, String bizType);

MPLogger.event(String logId, Map<String, String> params);
```

4.2.5. Automation log

Automation logs are used to record page switching and control click events. You can analyze the page view (PV) and unique visitor (UV) data on app functions or operating pages based on automation logs.

Initialize the event tracking

Call the following method to initialize automation log tracking events.

```
  MPLogger.enableAutoLog();
```

- For Portal & Bundle projects, you are suggested to call the method in the `postInit()` method of `MockLauncherActivityAgent` .
- For Native AAR projects, you are suggested to call the method in the `onCreate` method in `Application` , and it must be after calling the mPaaS framework initialization method.

Configure Activity

Activity records the PV of a page from `onResume` to `onPause` , with the page name identified by the Activity class name.

- Activities inheriting the class `BaseActivity` , `BaseFragmentActivity` , or `BaseAppCompatActivity` from mPaaS framework are automatically recorded.
- If not to inherit the base classes from mPaaS framework, you can add the life cycle monitoring code in `BaseActivity` as follows.

```
  public class BaseActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      MPTracker.onActivityCreate(this);
  }

  @Override
  public void onWindowFocusChanged(boolean hasFocus) {
      super.onWindowFocusChanged(hasFocus);
      MPTracker.onActivityWindowFocusChanged(this, hasFocus);
  }

  @Override
  protected void onResume() {
      super.onResume();
      MPTracker.onActivityResume(this);
  }

  @Override
  protected void onPause() {
      super.onPause();
      MPTracker.onActivityPause(this);
  }

  @Override
  protected void onDestroy() {
      super.onDestroy();
      MPTracker.onActivityDestroy(this);
  }
}
```

Configure Fragment

- If you use the `com.mpaas.mas.adapter.api.BaseFragment` provided by mPaaS framework, just inherit the class.
- If you use `Fragment` in the official library `support-v4`, you should enable BaseFragment to implement the `TrackPageConfig` interface, and add the life cycle monitoring configuration as follows.

```
  public class BaseFragment extends Fragment implements TrackPageConfig {

  /**
   * Page ID, it is generally the class name
   * If not passed in, the page analysis data may not be presented on the console
   */
  @Override
  public String getPageSpmId() {
      return this.getClass().getName();
  }

  @Override
  public Map<String, String> getExtParam() {
      return null;
  }

  @Override
  public boolean isTrackPage() {
      return true;
  }

  @Override
  public void onResume() {
      super.onResume();
      MPTracker.onFragmentResume(this);
  }

  @Override
  public void onPause() {
      super.onPause();
      MPTracker.onFragmentPause(this);
  }

  @Override
  public void onHiddenChanged(boolean hidden) {
      super.onHiddenChanged(hidden);
      MPTracker.onFragmentHiddenChanged(this, hidden);
  }

  @Override
  public void setUserVisibleHint(boolean isVisibleToUser) {
      super.setUserVisibleHint(isVisibleToUser);
      MPTracker.onFragmentSetUserVisibleHint(this, isVisibleToUser);
  }

  @Override
  public void onDestroy() {
      super.onDestroy();
      MPTracker.onFragmentDestroy(this);
  }
}
```

Add custom parameters

In baseline 10.1.68.44 and later versions, it is supported to add custom parameters in the automation log by using the following method.

```
MPLogger.addAutoLogCustomParam("test_key1", "test_value1");
MPLogger.addAutoLogCustomParam("test_key2", "test_value2");

Map<String, String> params = new HashMap<>();
params.put("test_key3", "test_value3");
params.put("test_key4", "test_value4");
MPLogger.addAutoLogCustomParams(params);
```

4.3. View local logs

This topic describes how to quickly find the logs you want to view locally.

Log storage path

The log storage path is determined by the `release_type` field in `assets/channel.config`.

- If the value of this field is dev, test, or testpre, logs are stored in **offline mode**. If the value is release, logs are stored in **online mode**.
- If no `assets/channel.config` file is available or the `release_type` field is not specified:
  - The **offline mode** applies when the APK package is a debug package.
  - The **online mode** applies when the APK package is a release package.

Storage path:

- **In online mode**, log files are stored in `/data/data/[PackageName]/files/mdap`. After a log file is uploaded, the local one is deleted.
- **In offline mode**, log files are stored in `/sdcard/Android/data/[PackageName]/files/mdap`. After a log file is uploaded, the local one is moved to `/sdcard/Android/data/[PackageName]/files/mdap/upload` for backup. The timestamp is added as the file name prefix of the backup file.

Log file name

Log files are named in the following format: `(Timestamp_)process name-business code`. The business code (bizType) varies with the log type as follows:

- Active device/user log: `mPaaSAliveAndroid`.
- App startup time log: `mPaaSLaunchAndroid`.
- Crash log: `mPaaSCrashAndroid`.
- Lag log: `mPaaSLAGAndroid`
- Stuck log: `mPaaSANRAndroid`
- Custom event log: `bizType` passed in when tracking is created. If no bizType is passed in, the business code is `UserBehaviorAndroid` by default.
- Automation log: `mPaaSAutomationAndroid`.

Log content format

- Active device log: See Log model > Behavior tracking. In the content, the value of **field 16** (tracking ID) is `reportActive`.

- Active user log: See Log model > Behavior tracking. In the content, the value of **field 16** (tracking ID) is `login` .
- App startup time log: See Log model > Performance tracking. In the content, the value of **field 12** (performance ID) is `performance` and the value of **field 13** (performance category) is `time_startup` .
- Crash log: See Log model > Performance tracking.
- Lag log: See Log model > Performance tracking.
- Stuck log: See Log model > Performance tracking.
- Custom event log: See Log model > Behavior tracking. In the content, **field 16** (event ID) corresponds to the `logId` passed in when a tracking is created.
- Automation log: See Log model > Automation tracking. In the content, the value of **field 09** (behavior ID) is `auto_openPage` which indicates page automation.

What to do next

Upload logs

4.4. Upload logs

Logs written to the local files on the client are synchronized to the log server in the following ways:

- Automatic upload: Logs are automatically uploaded to the log server when certain conditions are met.
- Upload based on the switch: Logs are uploaded to the log server based on the switch value delivered by the server.
- Manual upload: Logs are uploaded to the log server when the log upload API is called.

Automatic upload

The conditions for automatic log upload are as follows:

- Logs are automatically uploaded to the log server when the log quantity meets certain rules:
  - Logs recording active devices/users, startup time, lag, stuck, and crash are uploaded in real time.
  - Custom event logs and automation logs are uploaded when the log quantity reaches 50.
- When an app switches to the backend, the logs are uploaded for once.

Upload based on the switch

You can log in to the mPaaS console, choose **Mobile Analysis Service** > **Log Management** > **Configure upload switch** > **Event tracking configuration**, and modify the parameters to dynamically modify the log upload conditions.

The meanings of the parameters are as follows:

- **Upload**: The settings are effective only when this switch is turned on.
- **Network**: Select **All networks** or **Only Wi-Fi**. This parameter specifies the network through which logs are uploaded.
- **Business code**: The value is the same as the business code (bizType) set when tracking is created. For more information about common business codes, see View local logs.
- **Log count**: specifies the number of logs. Logs of this type are uploaded when the specified number is reached.

- **Log upload rate**: specifies the percentage of users whose logs are uploaded, in per-mil. The value 1000 indicates that logs of all the users are uploaded.
- **Minimum upload level**: Each log has a level when being generated. Logs of a level smaller than or equal to the value specified here are uploaded. For example, if this parameter is set to 2, logs of levels 1 and 2 are uploaded, but logs of level 3 are not uploaded.

For more information, see Configure the log switch.

Manual upload

Call the following code to manually upload logs:

```
MPLogger.uploadAll();
```

5.Integrate MAS into iOS

5.1. Instructions on MAS integration procedure

Mobile Analysis Service (MAS for short) depends on the client SDK to implement event tracking, collect user behavior, App performance and other related data to generate logs, and report the logs to the server. The server extracts valid data from the tracking logs uploaded by the client based on the negotiated tracking log format to monitor and analyze client metrics.

You can integrate MAS to iOS client based on native project with CocoaPods. The procedure is as follows:

1. Complete common mPaaS integration procedure.

   To use the mPaaS component, you must complete the common integration procedure with reference to Integrate by using CocoaPods based on the existing project.

2. Integrate the MAS component.
   i. Use the mPaaS plugin to add SDK.
   ii. Refer to the corresponding SDK development documents to use the SDK, such as configure project and add tracking logs.
   iii. View local logs.
   iv. Report logs.

5.2. Add SDK

This guide introduces how to integrate MAS to iOS client. You can choose the method: Access based on existing projects and using CocoaPods to connect the MAS SDK to the client.

Prerequisites

You have integrated your project to mPaaS. For more information, see: Integrate by using CocoaPods based on the existing project.

What to do next

Use the SDK with reference to the corresponding documentation.

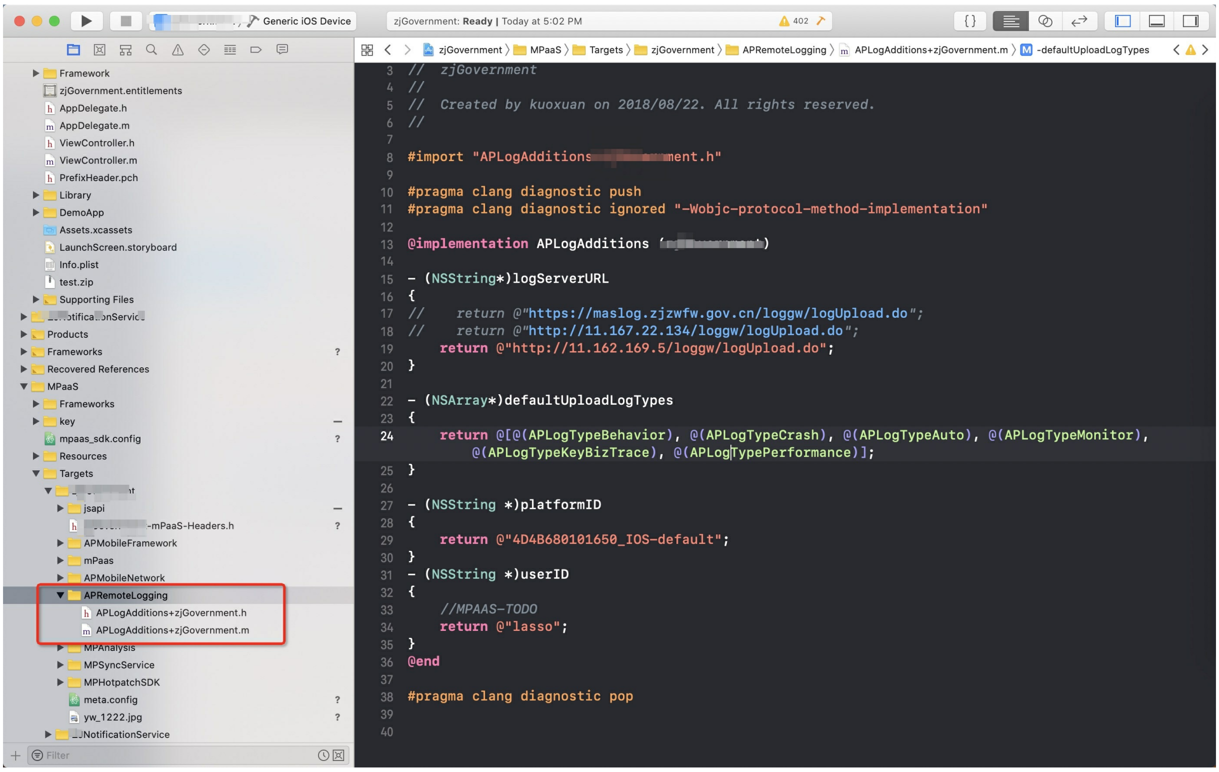
5.3. Use SDK

5.3.1. Configure project

After integrating the SDK, you should do some configurations on the project.

Configure project after version upgrade

In 10.1.32 and later versions, you don't have to add the Category file of the `APRemoteLogging` class. The middle layer will implement packaging to read the file from the `meta.config`. Therefore, you should check whether there are old-version configurations after upgrading the version. If yes, just remove them. The following figure shows the Category file of the `APRemoteLogging` class that should be removed from the new version.


Generic iOS Device
zjGovernment: Ready | Today at 5:02 PM
402
zjGovernment 〉 MPaaS 〉 Targets 〉 zjGovernment 〉 APRemoteLogging 〉 APLogAdditions+zjGovernment.m 〉 -defaultUploadLogTypes
Framework
zjGovernment.entitlements
AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m
PrefixHeader.pch
Library
DemoApp
Assets.xcassets
LaunchScreen.storyboard
Info.plist
test.zip
Supporting Files
Products
Frameworks
Recovered References
MPaaS
Frameworks
key
mpaas_sdk.config
Resources
Targets
jsapi
-mPaaS-Headers.h
APMobileFramework
mPaas
APMobileNetwork
APRemoteLogging
APLogAdditions+zjGovernment.h
APLogAdditions+zjGovernment.m
MPAnalysis
MPSyncService
MPHotpatchSDK
meta.config
yw_1222.jpg
NotificationService
Filter

```
//  zjGovernment
//
//  Created by kuoxuan on 2018/08/22. All rights reserved.
//

#import "APLogAdditions[redacted]ment.h"

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wobjc-protocol-method-implementation"

@implementation APLogAdditions [redacted])

- (NSString*)logServerURL
{
//    return @"https://maslog.zjzwfw.gov.cn/loggw/logUpload.do";
//    return @"http://11.167.22.134/loggw/logUpload.do";
    return @"http://11.162.169.5/loggw/logUpload.do";
}

- (NSArray*)defaultUploadLogTypes
{
    return @[@(APLogTypeBehavior), @(APLogTypeCrash), @(APLogTypeAuto), @(APLogTypeMonitor),
        @(APLogTypeKeyBizTrace), @(APLogTypePerformance)];
}

- (NSString *)platformID
{
    return @"4D4B680101650_IOS-default";
}
- (NSString *)userID
{
    //MPAAS-TODO
    return @"lasso";
}
@end

#pragma clang diagnostic pop
```

Configure user ID

Configure user ID in the project. The user ID is recorded in tracking logs by default. You can configure the user ID in the category of `MPaaSInterface`.

```
@implementation MPaaSInterface (Portal)
- (NSString *)userId
{
    return @"the-user-id";
}
@end
```

What to do next

You can add the following types of logs:

- Active device/user log
- Add performance log
- Add crash log
- Custom event log
- Automation log

5.3.2. Add activity report log

Activity report logs are used to count the number of App installation and users. You can view indicators such as new users, active users, and active accounts on the **Data Overview** page of the MAS console.

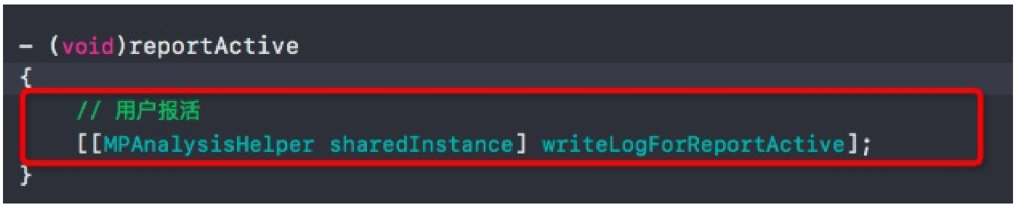
Supports log tracking based on the mPaaS framework and native projects.

Based on the mPaaS framework

The framework automatically reports active state information upon app startup and synchronizes the information to the app analysis console. No manual operation is required.

Based on a native project

An active state reporting API is encapsulated in the SDK. We recommend that you call this API in `didFinishLauncher`. Reference the header file `<MPMasAdapter/MPAnalysisHelper.h>`.



```
- (void)reportActive
{
    // 用户报活
    [[MPAnalysisHelper sharedInstance] writeLogForReportActive];
}
```

5.3.3. Add performance log

Performance logs record the startup time, lag, and stuck information about apps. You can view the startup time under **Mobile Analysis Service** > **Basic analysis**, and view the lag and stuck reports under **Performance analysis**.

Supports log tracking based on the mPaaS framework and native projects.

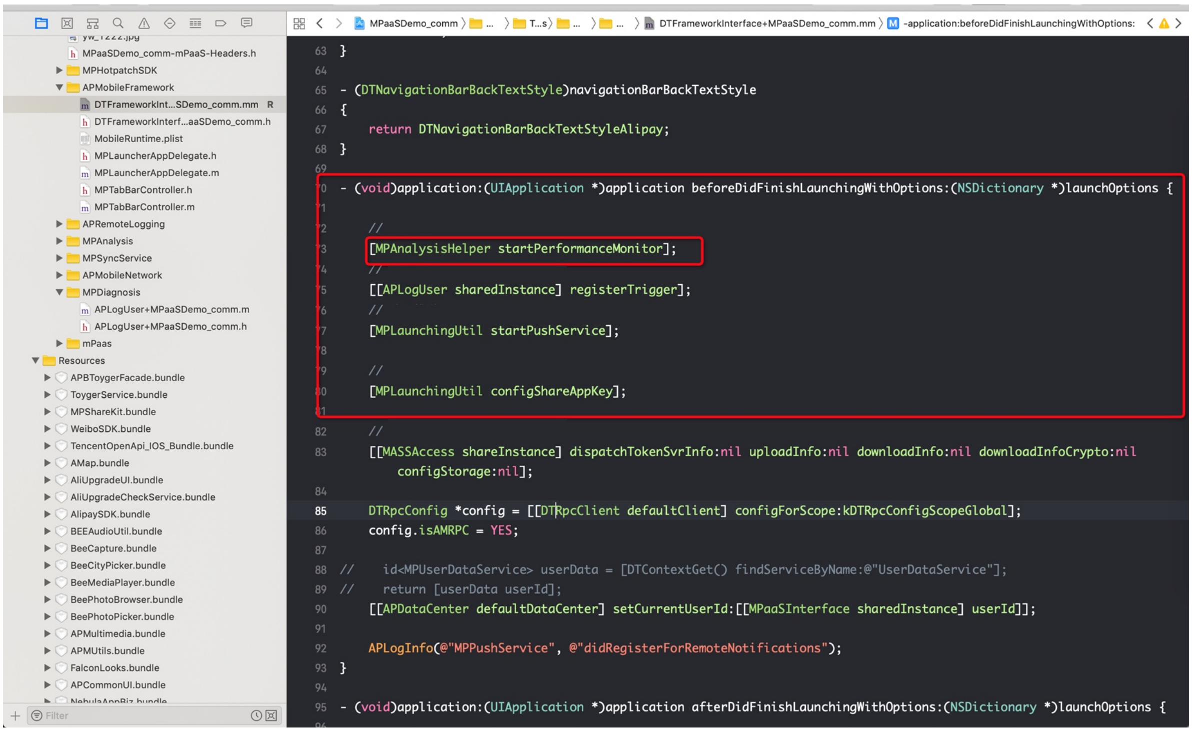
Based on the mPaaS framework

1. Lag monitoring is enabled for 10% devices by default. You can set the percentage through the following API:

   ```
    [MPAnalysisHelper setLagMonitorPercent: 100]; // Lag monitoring is enabled for all devices. The percentage needs to be set before the startPerformanceMonitor API is called.
   ```

   Lag monitoring is enabled only on a real device not in the Xcode debug state.

2. You need to call `[MPAnalysisHelper startPerformanceMonitor]` during App startup. It is recommended that this API be called in the `-(void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` launchOptions method.


MPaaSDemo_comm
T...s
DTFrameworkInterface+MPaaSDemo_comm.mm
-application:beforeDidFinishLaunchingWithOptions:
MPaaSDemo_comm-mPaaS-Headers.h
MPHotpatchSDK
APMobileFramework
DTFrameworkInt...SDemo_comm.mm
DTFrameworkInterf...aaSDemo_comm.h
MobileRuntime.plist
MPLauncherAppDelegate.h
MPLauncherAppDelegate.m
MPTabBarController.h
MPTabBarController.m
APRemoteLogging
MPAnalysis
MPSyncService
APMobileNetwork
MPDiagnosis
APLogUser+MPaaSDemo_comm.m
APLogUser+MPaaSDemo_comm.h
mPaas
Resources
APBToygerFacade.bundle
ToygerService.bundle
MPShareKit.bundle
WeiboSDK.bundle
TencentOpenApi_IOS_Bundle.bundle
AMap.bundle
AliUpgradeUI.bundle
AliUpgradeCheckService.bundle
AlipaySDK.bundle
BEEAudioUtil.bundle
BeeCapture.bundle
BeeCityPicker.bundle
BeeMediaPlayer.bundle
BeePhotoBrowser.bundle
BeePhotoPicker.bundle
APMultimedia.bundle
APMUtils.bundle
FalconLooks.bundle
APCommonUI.bundle
Filter

```
}

- (DTNavigationBarBackTextStyle)navigationBarBackTextStyle
{
    return DTNavigationBarBackTextStyleAlipay;
}

- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    //
    [MPAnalysisHelper startPerformanceMonitor];
    //
    [[APLogUser sharedInstance] registerTrigger];
    //
    [MPLaunchingUtil startPushService];

    //
    [MPLaunchingUtil configShareAppKey];

    //
    [[MASSAccess shareInstance] dispatchTokenSvrInfo:nil uploadInfo:nil downloadInfo:nil downloadInfoCrypto:nil
        configStorage:nil];

    DTRpcConfig *config = [[DTRpcClient defaultClient] configForScope:kDTRpcConfigScopeGlobal];
    config.isAMRPC = YES;

//    id<MPUserDataService> userData = [DTContextGet() findServiceByName:@"UserDataService"];
//    return [userData userId];
    [[APDataCenter defaultDataCenter] setCurrentUserId:[[MPaaSInterface sharedInstance] userId]];

    APLogInfo(@"MPPushService", @"didRegisterForRemoteNotifications");
}

- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
```

Based on a native project

A performance monitoring API is encapsulated in the SDK. We recommend that you call the `[PerformanceHelper performanceMonitor]` API in the `- (BOOL)application:(UIApplication )application didFinishLaunchingWithOptions:(NSDictionary )launchOptions` method of AppDelegate.

```
#import "PerformanceHelper.h"
    #import <MPAnalysis/MPAnalysisHelper.h>

    static NSTimeInterval __start_timestamp = 0;

    @implementation PerformanceHelper

    + (void)load
    {
        __start_timestamp = CFAbsoluteTimeGetCurrent();
    }

    + (void)performanceMonitor
    {
        //start performance monitor

       [MPAnalysisHelper setLagMonitorPercent: 100]; // Lag monitoring is enabled for a
ll devices. The percentage needs to be set before the startPerformanceMonitor API is ca
lled.
        [MPAnalysisHelper startPerformanceMonitor];

        //record the time interval used for the app startup
        NSTimeInterval time = CFAbsoluteTimeGetCurrent() - __start_timestamp;
        [[MPAnalysisHelper sharedInstance] writeLogForStartupWithTime:time];

    }

    @end
```

> **Note**
>
> Lag monitoring is enabled only on a real device not in the Xcode debug state.

Customize performance monitoring thresholds

When the default performance monitoring thresholds cannot meet your needs, you can customize relevant thresholds.

Set lag threshold

```
#Introduce header files
#import <MPMasAdapter/MPAnalysisHelper.h>

/**
 Set the threshold for main thread lag monitoring, in seconds, optional. The default va
lue is 2 seconds.
 */
+ (void)setLagTimeThreshold:(NSUInteger)threshold;

/**
 Set the interval between lag detection. It is recommended that lagTimeThreshold / lagC
heckInterval be equal to an integer
 */
+ (void)setLagCheckInterval:(NSTimeInterval)interval;
```

Set stuck threshold

```
#import <MPMasAdapter/MPMasSettings.h>

#Create the MPMasSettings category and rewrite the following method to customize it


/**
 Get the stuck duration threshold. If it needs to be customized, rewrite it in Category
. It is recommended that anrTimeThreshold / anrCheckInterval be equal to an integer.
 */
- (NSUInteger)anrTimeThreshold
{
    return customized duration;
}


/**
 Get the stuck detection interval. If it needs to be customized, rewrite it in Category
. It is recommended that anrTimeThreshold / anrCheckInterval be equal to an integer.
 */
- (NSTimeInterval)anrCheckInterval
{
    return customized detection interval length;
}
```

Set the startup stuck time threshold

```
#import <MPMasAdapter/MPMasSettings.h>

#Create the MPMasSettings category and rewrite the following method to customize it


/**
 Get the startup stuck time threshold. If it needs to be customized, rewrite it in Cate
gory.
 */
- (NSUInteger)startupAnrTimeThreshold
{
    return customized duration;
}
```

5.3.4. Add crash log

Crash logs record crashes of Apps. You can log in to the mPaaS console, and view crash reports in the **Mobile Analysis Service** > **Performance Analysis** page.

Supports log tracking based on the mPaaS framework and native projects.

Based on mPaaS framework

If you have accessed the framework (`APMobileFramework` repository available in the project), the crash reporting module automatically obtains crash logs and uploads the logs to the server. You only need to ensure that the crash monitoring is enabled after integrating the SDK. To ensure that crash logs are reported in time, it is recommended that you enable crash monitoring in the `main` function.

```
#import <MPMasAdapter/MPMasAdapter.h>

[MPAnalysisHelper enableCrashReporterService];
```

Based on native project

If you haven't accessed the framework (`APMobileFramework` repository unavailable in the project), you need to enable the crash monitoring when startup, and upload the crash logs after startup.

1. Enable crash monitoring in the `main` method.

   ```
   #import <MPMasAdapter/MPMasAdapter.h>

   [MPAnalysisHelper enableCrashReporterService];
   ```

2. Report the crash log in the `didFinishLaunchingWithOptions` method of the startup process.

   ```
   #import <MPMasAdapter/MPMasAdapter.h>

   [[MPAnalysisHelper sharedInstance] writeLogForCrashReporter];
   ```

Disaster recovery switch

By default, disaster recovery processing will be triggered when 4 consecutive crashes occur, and the files in the `Documents` directory will be cleared to avoid crashed problems caused by dirty data. In 10.1.60 and above, you can manually call the following interface to enable or disable disaster recovery processing.

```
#import <MPMasAdapter/MPAnalysisHelper.h>
/**
 * Enable or disable disaster recovery processing, enabled by default.
 */
+ (void)enableDisasterRecovery:(BOOL)enable;
```

Attentions

- Only crash logs of apps running on real devices will be captured and uploaded to the log server. If you need to debug crash monitoring, please disconnect Xcode and do not use the simulator.
- To ensure that the version in the crash log is consistent with the product version, be sure to set the bundle version and product version to the same version number in the project's `info.plist` .

5.3.5. Add custom event log

Custom event logs record button and link click operations, which can be configured when any action is triggered in the App, and used for custom event analysis and funnel analysis. Based on business needs, you can implement user behavior analysis through custom event tracking.

After connecting a client to mPaaS, you need to log in to the mobile analysis service console, and configure the relevant attributes and events in the **Mobile Analysis Service** > **Custom Configuration**. Then, you can select **Custom analysis** > **Event analysis** to view relevant data.

Event tracking API

The custom event tracking API is defined in the `MPRemoteLoggingInterface` class of `MPMasAdapter` . The API is defined as follows:

```
/**
 * Behavior tracking API. The client version, user ID, device ID, operating system
version, network type, device type, and software version are automatically filled in, a
nd no service tracking is required.
 *
 * @param bizType: optional, business type, which defaults to User_behavior_iOS. It is
recommended that you fill in with the business identification.
 * @param eventId: required, event tracking ID.
 * @Param extParam: optional, extended parameter, which can be set by service personnel
as required. The element is a dictionary, the dictionary content can be customized, and
the dictionary will be converted into key-value strings and recorded in logs.
 */
+ (void)writeLogWithBizType:(NSString *)bizType
                    eventId:(NSString *)eventId
                   extParam:(NSDictionary *)extParam;
```

Parameter description

Parameter	说明
bizType	Optional, business type, which defaults to `User_behavior_iOS` . It is recommended that you fill in with the business identifier.
eventId	Event tracking ID, corresponding to **Event ID** on the event creation page in the console.
extParam	Extension parameter. The key in the dictionary corresponds to **Attribute ID** on the attribute creation page in the console, and the value type determines **Data type** of the attribute.

Code sample

```
[MPRemoteLoggingInterface writeLogWithBizType:@"customBiz" eventId:@"customEvent" extPa
ram:@{@"key":@"v"}];
```

5.3.6. Add automatic log

Automatic page tracking records information such as page visits, source and dwell duration of pages. The information is used to analyze indicators such as **PV**, **UV**, and **page traffic**.

Tracking

Manual log tracking is not required for the following reasons:

- Automatic logs use a special behavior tracking method.
- During runtime, the system method is replaced by using the Objective-C method exchange technology, and the monitoring method is inserted.
- After the monitoring method is triggered through touchscreen operations or system notifications, it actively calls the behavior tracking API to upload the tracking logs.

Rules

During the life cycle of `UIViewController` , `viewDidAppear` indicates opening of a page and `viewWillDisappear` indicates closing of a page. When a page is opened and then closed, the number of PVs increases by 1.

Add custom parameters

You can add custom extension parameters for automatic page tracking by implementing the following method. The custom parameters will appear at the extension parameter 4 in the format of `custParm=123^cus2=myinfo` .

```
#import <MPMasAdapter/MPRemoteLoggingInterface.h>

/**
  Set the custom extension parameters for automatic tracking
 */
+ (void)setAutoRemoteLogExtendParam:(NSDictionary *)extParam;
```

Set the switch of automatic click tracking

```
#import <MPMasAdapter/MPMasSettings.h>

#Create the MPMasSettings category and rewrite the following method to customize it


/**
 Whether to enable automatic click tracking. It is enabled by default. If you need to disable it, return NO
 */
+ (BOOL)enableAutoClick
{
 return NO;
}
```

5.4. View local logs

Logs generated by calling the log API are firstly stored in the local sandbox file, and then are uploaded to the log server when the upload logic is triggered.

Local log format

Version 10.1.60 & 10.1.68

- Logs in the local sandbox are stored in the **Library** > **atrack** > **logs** directory, which only stores the unreported logs, and the reported logs are no longer stored.
- Log files are named in the following format: `Business code.log`. Logs are classified based on the parameter bizType, and logs of the same type are stored in the same file. At present, the following common tracking types are available:
  - `autotrack`: Automation tracking.
  - `behavior`: Behavior tracking, including active state tracking and custom event tracking. You can define the business codes by the `bizType` parameter for custom event tracking. For more information, see Custom event tracking.
  - `performance`: Performance tracking, such as app startup time tracking.

Version 10.1.32

> **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.

- Logs in the local sandbox are stored in the `Library/log` directory.
- Log files are named in the following format: `Business code.Timestamp.log`. Logs are classified based on the parameter bizType, and logs of the same type are stored in the same file. At present, the following common tracking types are available:
  - `autotrack`: Automation tracking.

  - `behavior`: Behavior tracking, including active state tracking and custom event tracking You can define the business codes by the `bizType` parameter for custom event tracking. For more information, see Custom event tracking.
  - `crash`: Exception tracking.
  - `performance`: Performance tracking, such as app startup time tracking.
- A log file with the file name extension `.log` is generated upon each cold startup of an App.

Tracking log format

- A log file consists of many logs, with one log in a row.
- Each log contains several character strings separated by commas, with different meanings at different locations. The log server splits a log based on location information. The format of a complete log is as follows:

```
  0_     257_1479573031.408824_D-VM,2016-11-20 00:30:31:408,1000533192018_IOS-0000000
001,2.0.0.0,2,-,7542B136-5EA8-4C3A-930D-8BF2CA15F3CA,-,event,-,-,-,-,-,-,startApp,-,u
,c,Launcher,-,NativeApp,-,-,-,-,-,2,-,-,-,-,-,iPhone 6S,9.3.3,WIFI,-,-,follow_system_
zh-Hans-CN,-,-,-,-,VoiceOver=0$$
```

- In version 10.1.60 & 10.1.68, the tag at the beginning of each log no longer marks the upload status of log.
- In version 10.1.32, each log contains a tag at the beginning. The first digit indicates the log upload status. Digit 0 indicates that the log is not uploaded. Digit 1 indicates that the log has been uploaded.
- For the specific meaning of each field, see Log model.

What to do next

Upload logs

5.5. Upload logs

Logs written to the local files on a client are synchronized to the log server in following 3 ways.

- Upload logs automatically: Logs are automatically uploaded to the log server when certain conditions are met.
- Upload logs based on switch: Logs are uploaded to the log server based on the switch value delivered by the server.
- Upload logs manually: Logs are uploaded to the log server when the log upload API is called.

To stop uploading logs, you need to turn off the iOS tracking switch. See Stop uploading logs for more information.

Upload logs automatically

The conditions for automatic log upload are as follows:

- The app triggers the logic of checking log upload upon each cold startup.
- Logs are immediately uploaded when the app is switched to the backend.
- Logs of the same type are uploaded when the quantity reaches 40.
- To ensure timely upload of crash logs, a crash log is uploaded each time the app is restarted after a crash.

Upload logs based on switch

You can log in to the mPaaS console, choose **Mobile Analysis Service** > **Log Management** > **Configure Upload Switch > Tracking Configuration**, and modify the parameters to dynamically modify the log upload conditions.

Dynamically control the log upload through the mPaaS console.

The meanings of the parameters are as follows:

- **Upload**: The settings are effective only when this switch is turned on.
- **Network**: Select **All networks** or **Only Wi-Fi**. This parameter specifies the network through which logs are uploaded.
- **Business code**: The value is the same as the business code (bizType) set when tracking is created. For more information about common business codes, see View local logs.
- **Minimum upload level**: Each log has a level when being generated. Logs of a level smaller than or equal to the value specified here are uploaded. For example, if this parameter is set to 2, logs of levels 1 and 2 are uploaded, but logs of level 3 are not uploaded.
- **Upload quantity**: Specify the number of logs. Logs of current type are uploaded when the specified threshold is reached. For example, you can change the default value 40 to another value.
- **Log upload rate**: Specify the percentage of users whose logs are uploaded, in per-mil. The value 1000 indicates that logs of all the users are uploaded.

For more information, see Configure the log switch.

Upload logs manually

If some logs need to be uploaded in real time, call the following API to upload logs manually:

```
[MPRemoteLoggingInterface upload];
```

Stop uploading logs

Turn off the iOS tracking switch to stop log uploading.

> ⓘ **Important**
>
> This method only works for mPaaS baseline 10.1.68.42 and later versions.

The method of disabling tracking is as follows:

```
#import <MPMasAdapter/MPAnalysisHelper.h>

[MPAnalysisHelper enableRemoteLog:NO];
```

6.Basic analysis

6.1. Data overview

6.1.1. Introduction to data overview

Data overview presents an app’s real-time and historical statistical data including startup count, active users (device ID), active accounts (user ID), new users, cumulative users and other critical indicators from multiple dimensions such as platform, version, and channel, providing support for your business decision making.

Prerequisites

Data in data overview is obtained from the **active device/user tracking logs** reported by the client. Make sure that active device/user log tracking is correctly enabled when you access the client. For more information, see Access Android or Access iOS.

View data overview

1. Log in to the console, choose **Products and Services** > **Mobile PaaS**, and select an app.
2. In the navigation bar on the left, choose **Mobile Analysis Service** > **Data overview**.
3. Click the **Real-time dashboard** or **Historical trends** tab to view real-time or historical data.

Indicator description

You can view the following indicators in data overview:

- **Startup times**: The total number of times that users start the app, including the case where the app is started again after it is stopped or left unattended for over 30 minutes.
- **Active users**: The number of distinct device IDs used to log in to the app.
- **Active accounts**: The number of distinct user IDs logging in to the app.
- **New users**: The total number of users minus the number of old users.
- **Cumulative users**: The total number of users visiting an app after the app connects to Mobile Analysis Service, that is, the total number of distinct device IDs.
- **Cumulative accounts**: The total number of accounts, that is, the total number of distinct user IDs.

Move your mouse cursor over the question mark icon (?) on the upper right of an indicator, and you will see the indicator description and calculation rules.

Related topics

- Real-time dashboard
- Historical trends

6.1.2. Real-time dashboard

Real-time dashboard displays real-time analysis results of App core indicators. For related indicator descriptions, see Calculation rules for indicators.

The steps to view the real-time dashboard of data analysis are as follows:

1. Log in to the console, click **Products and Services** > **Mobile Development Platform mPaaS**, and select an application.
2. On the left side of the navigation bar, click **Mobile Analysis Service** > **Data Overview**.
3. On the right page, click the **Real-time Dashboard** tab to enter the real-time dashboard page.
4. Select the platform, version, and channel to view specific analysis data.

The real-time dashboard is divided into three parts: Indicator overview, Indicator trend, and Indicator detailed data.

Indicator overview

Displays real-time statistical data of various indicators, including real-time data of three indicators: **Startup times**, **Active users**, and **Active accounts**, as well as daily and weekly year-on-year data.

Indicator trend

The aggregated data for each hour is displayed in the form of a line chart, including the indicator comparison curves for today, yesterday, 7 days ago, and 30 days ago.

You can select an indicator from the drop-down list on the upper left of the line chart to view the data change trend of each indicator.

Indicator detailed data

Hourly detailed data of the current day is presented in a table. The table covers three indicators: **Startup times**, **Active users**, and **Active accounts**.

You can click the **Export** button to export the data locally to an Excel file.

6.1.3. Historical trends

This topic presents the historical analysis results of core App indicators in charts. For detailed information about indicators and how to view indicator charts, see Data overview.

The steps to view the historical trends of indicators data are as follows:

1. Log in to the console, click **Products and Services** > **Mobile Development Platform mPaaS**, and select an application.
2. On the left side of the navigation bar, click **Mobile Analysis Service** > **Data Overview**.
3. On the right page, click the **Historical Trends** tab to enter the real-time dashboard page.
4. Select the platform, version, channel and time period to view specific analysis data.

The historical trends is divided into two parts: Indicator trend, and Detailed data.

Indicator trend

Display the indicator data in the selected time period in the form of a line chart:

- Each indicator is displayed separately, and the indicator can be selected through the drop-down list in the upper right corner. The optional indicators include the number of launches, active users, active accounts, new users, cumulative users, and cumulative accounts.
- Select the date dimension for the indicator data display, including daily display, weekly display, and monthly display.
  - **Display by day**: Display the data curve for each day in the selected time range.
  - **Display by week**: Display the data curve for each week in the selected time range. If the selected time range is less than 7 days, the weekly dimension is not available.

  - **Display by month**: Display the data curve for each month in the selected time range. If the selected time range is less than 30 days, the monthly dimension is not available.

Detailed data

Display detailed data in the selected time period in a table, including daily, weekly or monthly data of 6 indicators, including the number of launches, active users, active accounts, new users, cumulative users, and cumulative accounts. Support sorting by different indicators to view data overview.

- Click **Export** to export this data to the local computer in the form of an Excel table.
- Click the sort icon (⇕) to sort in ascending or descending order by the current indicator.

6.2. Behavior analysis

Behavior analysis displays data related to user behavior. It mainly analyzes what operations App users performed at what time and where, through what channels, and how long they took. It can help you understand users' operation patterns, access paths, and behavioral characteristics.

To view the behavior analysis report, perform the following steps:

1. Log in to the mPaaS console, select **Products and Services** > **mPaaS**, and then select an application.
2. On the left navigation bar, click **Mobile Analysis Service** > **Basic analysis** > **Behavior analysis**.
3. Filter analysis data. In the upper right of the page, click the **Add filter** button, select the platform, App version, channel, and date of data analysis to view the corresponding page analysis data.

The behavior analysis page displays the following behavior indicators data through charts:

- Behavior analysis
- Popular pages Top5
- Download channels
- Startup time
- Page traffic
- Users by region

> **Note**
>
> - Hover your mouse cursor on the question mark of an indicator (?), and you can see the description and calculation rule of the indicator.
> - The statistical data is based on the historical daily aggregation. The daily aggregate result of all statistical items is stored in the database, and the data of the current day is not taken into aggregation.
> - For more information about the indicator calculation rule, see Basic indicator calculation rule.

User activity analysis

A user's active hours is subject to the time the user starts to use the App, which is calculated based on the time the log is reported minus the backend duration. Active hours is based on two hours' period starting from 0:00, for example, 2:00 - 4:00 am. User's active hours is based on the time period when the user starts using the App.

- **Daily active time:** The average daily time a user spends using the app is calculated by dividing the total duration of the app by the number of device IDs (UV).
- **Times/day·person (times):** The average daily number of times a user opens the app is calculated by dividing the non-deduplicated data (PV) pushed to the background by the deduplicated data (UV) pushed to the background.

Popular pages Top5

Displays the top 5 most visited pages of the current App. Click a page name to jump to the analysis details page of that page.

Download channels

Based on the active device/user tracking, calculate the top three channels for App downloads. When calculating, deduplicate multiple downloads of the same user in the same channel.

For example, User A downloads the App 1 time through channel C1, and downloads it 2 times through channel C2. When the data is classified, the downloads through channel C1 and C2 are only counted once respectively.

> **Note**
>
> To understand the download channels of Android devices, you can generate and upload different channel packages to the corresponding app market. Fields with different channels have been added to the channel package. After the user downloads and runs the app through the app market, the client automatically reports the event tracking field to the server. For the steps of generating channel package, see **Set channel ID** in Based on mPaaS framework.

Startup time

Startup time counts the average startup time for the first startup and non-first startup, in seconds.

The first startup refers to starting an App for the first time after the App is installed; non-first startup refers to restarting an App after the first-time App startup and exit.

Page traffic

It indicates where the visitors of a specific page come from (source) and go to (destination). Based on the **pid** and the **refer** fields, the system calculates the sources of the current page and the proportion of each source to the total sources, and the destinations of the current page and the total proportion of each destination.

You can switch pages through the drop-down box on the upper right to view the corresponding page data. The pages displayed in the drop-down list are all pages added on the **Page Analysis** > **Page configuration** tab page.

Users by region

The active user tracking log sorts the users by region, and shows the top five regions with the highest proportion of distinct users. At the same time, the proportions of users in the first five regions to the total number of users are displayed.

- For map of China, the proportion of regional distinct users is calculated by province.
- For world map, the proportion of regional distinct users is calculated by country.

6.3. Retention analysis

Retention analysis displays the data about user stickiness by analyzing the number and proportion of users who visit the app again among a group (usually a group of new users). Based on the retention data, you can adjust marketing strategy for the lost users so as to continuously attract more users and improve user retention.

To check the retention analysis report, complete the following steps:

1. Log in to the mPaaS console, select **Products and Services** > **Mobile PaaS**, and then select the target app.
2. On the left navigation bar, click **Mobile Analysis Service** > **Basic analysis** > **Retention analysis** tab.
3. Filter the analysis data you want to view.
    - At the top of the page, select Platform, Version, Channel.
    - Select the time period for data analysis.
    - Select the dimension to display the analytical data, that is, display by device or account.
    - Select the time granularity for displaying analytical data, which can be day, week, or month. The default is daily data, and the default time range is 7 days before yesterday.

The retention analysis page is divided into Retention overview and Retention details.

> **Note**
>
> - Hover your mouse cursor on the question mark of an indicator (?), and you can see the description and calculation rule of the indicator.
> - The statistical data is based on the historical daily aggregation. The daily aggregate result of all statistical items is stored in the database, and the data of the current day is not taken into aggregation. For more information about the indicator calculation rule, see Basic indicator calculation rule.

Retention overview

Retention rate refers to the number of new users who retained in the following 7 days since their first visit to an app on a certain day divided by the total count of new users, shown in the form of line chart.

For daily data, the "N days later" on the horizontal axis refers to the average retention rate in N days within the selected time period. Similarly, the weekly or monthly data refers to the average retention rate in N weeks or months within the selected time period. Support showing data of up to 30 days for daily retention, 6 weeks for weekly retention, and 3 months for monthly retention.

Retention details

The number of new users on a given day who visit the app again each day over the next 7 days is divided by the total number of new users, and the results are displayed in a table.

For example, the count of new users on day T is N, the count of N new users who log in again on day T+1 is M1, and the count of users who log in again on day T+7 is M7. Thus, the retention rate for day T+1 is M1/N while day T+7 is M7/N.

6.4. Page analysis

Page analysis displays the data about the visited pages in the app.

To check the page analysis report, perform the following steps:

1. Log in to the mPaaS console, select **Products and Services** > **Mobile PaaS**, and then select your target app.
2. On the left navigation bar, click **Mobile Analysis Service** > **Basic analysis** > **Page analysis**.
3. Filter analysis data. At the top of the page, select the platform, version, channel, and date of data analysis to filter the data. You can also search for analysis reports of related pages by entering keywords of the page name in the search bar at the top right of the page.

Through the page analysis report, you can view the analysis data of all pages in the app. Page analysis result is presented through charts.

> **Note**
>
> - Hover your mouse cursor on the question mark of an indicator (?), and you can see the description and calculation rule of the indicator.
> - The statistical data is based on the historical daily aggregation. The daily aggregate result of all statistical items is stored in the database, and the data of the current day is not taken into aggregation. For more information about the indicator calculation rule, see Basic indicator calculation rule.

Page analysis overview

- **Users**: Count the devices (deduplicated by device ID) visiting the current page.
- **Accounts**: Count the users (deduplicated by user ID) visiting the current page.

  > **Note**
  >
  > Users who don't log in to the app are not taken into account.

- **PV**: Total page views of the current page.
- **Exit rate**: Proportion of counts that users exit from app from the current page to the total PVs on the current page.

  Exit rate = (total non-deduplicated PVs on the current page - non-deduplicated PVs with current page as source page)/total PVs on the current page.

- **Average time on page**: Total time on the current page divided by total PVs on the current page.

Page analysis details

Click the target page to view the detailed data.

Page view overview

Display the PV (total page views) and UV (deduplicated page views), with day-on-day and week-on-week ratios.

Page view trend

Display the PVs in the past 7 days, with the PV trend shown in the form of line chart.

Page traffic

Indicate where a specific page comes from and goes to. Based on the `pid` and the `refer` fields, the chart shows the proportion of users from each source page to all source pages as well as the proportion of users going to each downstream page to all downstream pages.

6.5. Device analysis

Device analysis shows the critical indicators of an App on different device models.

View device analysis report

Perform the following steps to view the device analysis report:

1. Log in to the mPaaS console, select **Products and Services** > **Mobile PaaS**, and then select your target App.
2. On the left navigation bar, click **Mobile Analysis Service** > **Basic analysis** > **Device analysis**.
3. Filter the analysis data. At the top of the page, select the platform, version, channel, and date of data analysis to filter the data.

> **Note**
>
> The statistical data is based on the daily summary of history. The database records the daily summary results of all statistical items. The data of the current day is not included in the summary results. For more information about the indicator calculation rules, refer to Indicator calculation rules.

Indicators on different device models

The critical App indicators on different device models are presented in a histogram.

Indicators are presented separately. You can select an indicator from the drop-down list on the upper left. Available indicators include startup times, active users, active accounts, and new users.By default, the top 10 device models are displayed.

Detailed data

The critical App indicators on different device models are presented in a table.

- Indicators include new users and proportion, active users and proportion, active accounts and proportion, cumulative users and proportion, and startup times and proportion.
- Data of all device models is included.

6.6. Component usage analysis

6.6.1. Hotfix analysis

Hotfix refers to fixing bugs online without releasing a new version. By viewing the hotfix report, you can learn about the information about Remote Procedure Call (RPC), hotfix and rollback.

The steps to view the hotfix report are as follows:

1. Log in to the mPaaS console, choose **Products and Services** > **mPaaS**, and select target application.
2. From the navigation bar on the left, select **Mobile Analysis Service** > **Component usage analysis** > **Hotfix analysis** to see the hotfix report.
3. Filter data. At the top of the page, select platform, application version, hotfix version, and the date of data analysis to view the corresponding hotfix analysis data.

The following table describes the hotfix analysis indicators.

Indicator	Description
RPC start count	The total number of RPC requests sent over the network to download patch files.
RPC success count	The number of RPC requests sent successfully to download patch files.
Hotfix start count	The number of times patches are loaded on all devices.
Hotfix success count	The number of times patches are successfully loaded on all devices.
Rollback start count	The number of times the current application has been rolled back.
Rollback success count	The number of times the current application has been successfully rolled back to the baseline version or patch version.

6.6.2. Offline package analysis

Offline package refers to the package that includes HTML, JavaScript, CSS, and other in-page static resources. You can download the offline packages in advance, open them on client, and then load them locally so as to get away from the influence of network environment on HTML5 pages. By viewing the offline package analysis report, you can learn how many times the offline package has been delivered, successfully downloaded and used.

To view the offline package report, complete the following steps:

1. Log in to the mPaaS console, choose **Products and Services** > **Mobile PaaS**, and select target Application.
2. From the navigation bar on the left, choose **Mobile Analysis Service** > **Component usage analysis** > **Offline package analysis** to see the offline package report.
3. Filter data. At the top of the page, select platform, app version, offline package, or offline package version, and select a date range for data analysis to view the corresponding offline package analysis data.

Indicator overview

Indicator	Description
Number of deliveries	Refers to the number of times the client successfully requests offline package update reminders. If you have downloaded the offline package but not installed it, you will still receive offline package update reminders.
Number of arrivals	The number of times the client successfully decompresses the offline package.

Number of openings	The number of times you open the offline package on the client.

Indicator trend

Displays the offline package indicator trend within the specified time range, supporting display by minute, hour, and day dimensions.

> **Note**
>
> The data of minute and hour dimensions are only supported when the query time range is one day. For example, when the selected time range is 2020-06-01 ~ 2020-08-01, the hour and minute dimensions are not available.

Detailed data

On the lower part of the page, you can view the detailed data of **Deliveries**, **Arrivals**, and **Opens** in the time period you selected. The time granularity of data display depends on the display time dimension you select.

6.6.3. Scan analysis

Mobile Analysis Service (MAS) supports statistical analysis of the usage data of the mPaaS code scanning component, including the number of calls, number of successful calls, call duration, and call success rate.

To view the analysis report, perform the following steps:

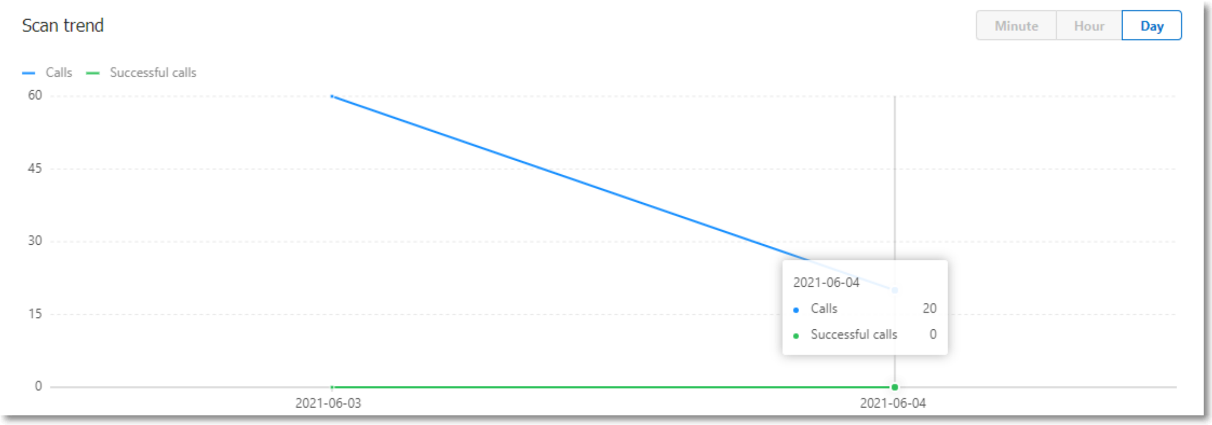
1. Log in to the mPaaS console, choose **Products and Services** > **mPaaS**, and select target application.
2. On the left navigation pane, choose **Mobile Analysis Service** > **Component usage analysis** > **Scan analysis** tab to view the scan statistics report.
3. Filter data. At the top of the page, select the platform, version, and date range for data analysis to view the corresponding scan code analysis data.
4. Select the data display dimension: minute, hour, or day.

   > **Important**
   >
   > The data of minute and hour dimensions are only supported when the query time range is one day. For example, when the selected time range is 2020-06-01 ~ 2020-08-01, the hour and minute dimensions are not available.

Scan trends

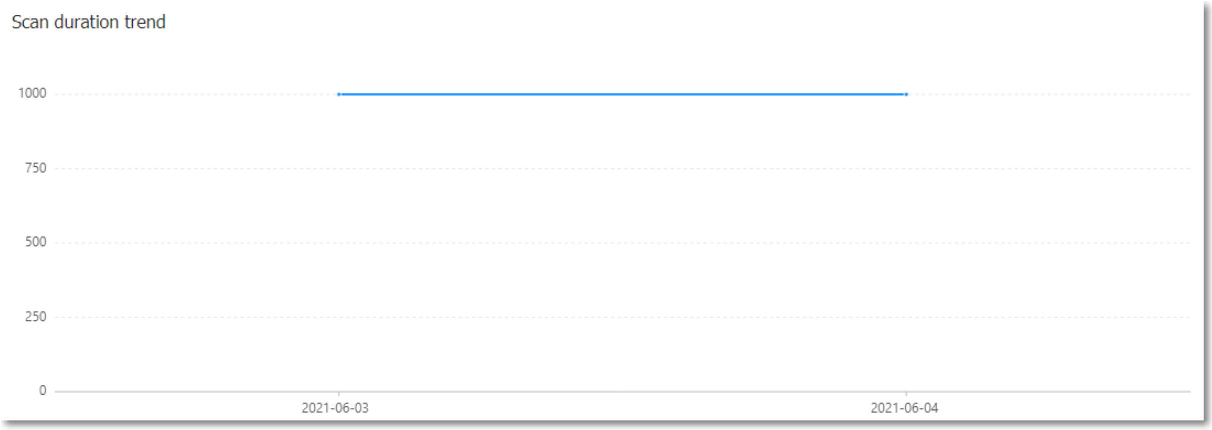
Displays the trends of scan calls within a specified time range.

- **Calls**: Total times of scanning, that is the total number of times the scan component is called.
- **Successful calls**: Times of successful scanning, taht is the total number of times the scan component is successfully called to perform scanning.



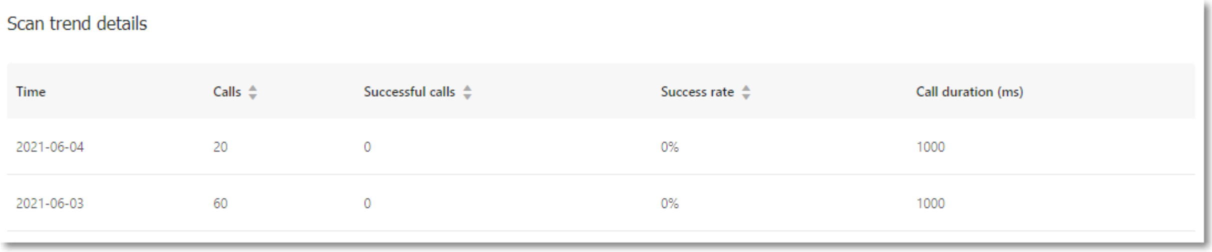
Scan duration trends

Display the trends of scan duration within a specified time range. Scan duration indicates the time taken to scan by calling the scan component. In the scan analysis of MAS, scan duration is measured in milliseconds (ms).



Scan trend details

Display the detailed daily scan data within a specified time range. The time granularity of data display depends on the display time dimension you select.



Scan trend details

Time	Calls	Successful calls	Success rate	Call duration (ms)
2021-06-04	20	0	0%	1000
2021-06-03	60	0	0%	1000

6.6.4. Mini Program analysis

Mobile Analysis Service (MAS) supports analyzing the mini program usage data of the current app. You can view the basic key indicators and page-based analysis data of the mini program on the **Mini Program analysis** page, such as PV, open times, startup times, and active users.

The steps to view the Mini Program analysis report are as follows:

1. Log in to the mPaaS console, click **Products & Services** > **mPaaS** and select an application.

2. Enter **Mobile Analysis Service**> **Component usage analysis**> **Mini Program analysis** page from the left navigation pane.
3. Filter data. In the upper right corner of the page, select a mini program and select the date for data analysis to view the corresponding mini program analysis data.

Indicator overview

Indicator overview provides the following indicator data:

Indicator	Description
Total PV	Total number of visits to all pages of the mini program on the day.
Open times	Startup times of the mini program during the selected date.
Cumulative startup times	The total times that the mini program starts up from the time when it went online to the specified date.
Active users	Total number of users who have used mini program in the selected date, calculated by device ID (distinct).

Page analysis

In page analysis, you can view the analysis data of different pages of the mini program, including page views (PV), number of users, and average stay time.

The data of all pages are displayed by default. You can also enter the page name or related keywords in the search box in the upper right to search for the analysis data of the related pages.

6.7. Page configuration

Page analysis page displays all the pages with user visits in a specified time period in the current app and the visit data of each page. To facilitate the page data query, you can set the page ID respectively shown in iOS system and Android system as well as the page ID's corresponding name through page configuration.

Add page configuration

Complete the following steps to add page configuration:

1. Log in to the mPaaS console, select your target app, and then click **Mobile Analysis Service** > **Basic analysis** on the left navigation bar.
2. Click the **Page configuration** tab on the right.
3. Click **Add**, and add the following page information:
    - **Name**: Page name in report.
    - **iOS**: Page ID in iOS system, which is the VC class name, for example `viewControlleName` .
    - **Android**: Page ID in Android system, which is the activity class name, for example `com.mpaas.demo.launcher.MainActivity` .

Note

- For the same page, the page name shown in iOS system must be the same as that in Android system.
- Every iOS ID and Android ID must be unique.

Modify page configuration

Select the target page configuration you want to modify, click **Modify** in the **Operations** column to edit the page ID and name.

Modify page configuration

Select the target page configuration you want to delete, click **Delete** in the **Operations** column and confirm deletion. This operation will not influence the page report presentation on the **Page analysis** page.

6.8. Calculation rules for indicators

The following tables list the tracking and calculation rules of all the basic indicators that you see in the basic analysis reports. You might see all or some of the following indicators, which is subject to your actual workspace.

Important

If an indicator is time-sensitive, the value of the indicator is subject to the time the server receives the reported logs.

Data overview

Real-time dashboard

Indicator	Tracking	Calculation rule	Distinct or Not	Realtime or Not
Startup times	Active device/user tracking	Total number of times that users start the app, including cold start and the case that the system brings app activity to the foreground.	Not distinct	Realtime
Active users	Active device/user tracking	Total number of distinct device IDs that log in to the app within a specific period.	Distinct	Realtime
Active accounts	Active device/user tracking	Total number of distinct user IDs that are used to log in to the app within a specific period.	Distinct	Realtime

Historical trend

Indicator	**Tracking**	**Calculation rule**	**Distinct or Not**	**Realtime or Not**
Startup times	Active device/user tracking	Total number of times that users start the app, including cold start and the case that the system brings app activity to the foreground.	Not distinct	Not real-time
Active users	Active device/user tracking	Total number of distinct device IDs that log in to the app within a specific period.	Distinct	Not realtime
Cumulative users	Active device/user tracking	Total number of users visiting an app after the app connects to Mobile Analysis Service, that is, total number of distinct device IDs.	Distinct	Not realtime
New users	Active device/user tracking	New distinct device IDs that log in to the app within a specific period.	Distinct	Not realtime
Active accounts	Active device/user tracking	Total number of distinct user IDs that are used to log in to the app within a specific period.	Distinct	Not realtime
Cumulative accounts	Active device/user tracking	Total number of distinct user IDs that are used to log in to the app after the app connects to Mobile Analysis Service.	Distinct	Not realtime

Basic analysis

Behavior analysis

Indicator	**Tracking**	**Calculation rule**	**Distinct or Not**	**Realtime or Not**

Users’ active hours	Backend tracking	A user's active hours is subject to the time the user starts to use the app, which is calculated based on the time the log is reported minus the backend duration. Active hours are based on two hours’ period starting from 0:00, for example, 2:00~4:00 am. For example, if a user starts to use the app at 7:55 am and the app goes to backend at 8:05 am, then the user’s active hours are 6:00~8:00 am.	Not distinct	Not realtime
Daily user engagement: duration/day	Backend tracking	Total duration (seconds) of using the app by users divided by the distinct count of user IDs. The backend tracking records the time when the app page appears on the foreground each time, and calculates the accumulative usage time length of the app based on the time recorded.	Distinct	Not realtime
Download channels	Active device/user tracking	Top 3 channels from which users download the app, aggregated by the channel field based on active device tracking. Downloading the app for multiple times on the same channel by the same user are counted only once. For example, if a user downloads the app once from Channel C1 and twice from Channel C2, C1 and C2 are both counted only once.	Distinct	Realtime
Startup speed	Performance tracking	Average startup time for first-time startup and non-first-time startup, in seconds. The first startup refers to starting an app for the first time after the app is installed; non-first startup refers to restarting an app after the first-time app startup and exit.	Not applicable	Realtime

Page traffic	Page automation tracking	Indicates where a specific page comes from and goes to. Based on the `pid` and the `refer` fields, all the source pages and ratio of each source page to all the source pages are shown; all the next pages and ratio of each next page to all the next pages are shown.	Not distinct	Not realtime
Users by region	Active device/user tracking	Top 5 regions that have most app users (de-duplicated), aggregated by the region field based on active device tracking. Meanwhile, the proportion of users in each top 5 region to the total number of all app users is shown. For map of China, the top 5 provinces are shown; for the world map, the top 5 countries are shown. For example, for map of China, if a user logged in twice in Province A in the morning and once in Province B in the evening, Province A and B are both counted only once.	Distinct	Realtime

Retention analysis

Indicator	Tracking	Calculation rule	Distinct or Not	Realtime or Not

Retention rate	Active device/user tracking	Retention rate refers to the number of new users who retained in the following 7 days since their first visit to an app on a certain day divided by the total count of new users, shown in the form of line chart. For example, the count of new users on day T is N, the count of N new users who log in again on day T+1 is M1, and the count of users who log in again on day T+7 is M7. Thus, the retention rate for day T+1 is M1/N, and the rate for day T+7 is M7/N.	Distinct	Not realtime

Page analysis

Indicator	Tracking	Calculation rule	Distinct or Not	Realtime or Not
Number of users	Page automation tracking	Count the devices (deduplicated by device ID) visiting the current page.	Distinct	Not realtime
Number of accounts	Page automation tracking	Count the users (deduplicated by user ID) visiting the current page.	Distinct	Not realtime
PV	Page automation tracking	Total page views of the current page.	Not distinct	Not realtime
Exit rate	Page automation tracking	Exit rate = (total non-deduplicated PVs on the current page - non-deduplicated PVs with current page as source page)/total PVs on the current page	Not distinct	Not realtime
Time on page	Page automation tracking	Total time on the current page divided by total PVs on the current page.	Not distinct	Not realtime

Device analysis

Indicator	Tracking	Calculation rule	Distinct or Not	Realtime or Not
Startup times	Active device/user tracking	Total number of times that users start the app by device model, including cold start and the case that the system brings app activity to the foreground, counted by device model.	Not distinct	Not realtime
Active users	Active device/user tracking	Total number of distinct device IDs that log in to the app within a specific period, counted by device model.	Distinct	Not realtime
New users	Active device/user tracking	New distinct device IDs that log in to the app within a specific period, counted by device model.	Distinct	Not realtime
Active accounts	Active device/user tracking	Total number of distinct user IDs that are used to log in to the app within a specific period, counted by device model.	Distinct	Not realtime

7.Custom analysis

7.1. Custom dashboard

7.1.1. Create custom dashboards

The dashboard is a report that presents various data analysis results on one page. The custom dashboard of mobile analytics supports displaying custom analysis data in different types of reports according to business needs. By viewing the various indicator data and change trends on the report, users can quickly judge the business situation of the App and make corresponding decisions.

To create a custom dashboard, you need to add reports and complete related configurations. You can add a custom analysis report to the dashboard by creating a new one or selecting an existing analysis report.

Type	Object
Select an existing analysis report	Event
	Funnel
Create a custom analysis report	Event
	Funnel

Prerequisites

- When you add custom analysis, if you use the existing event analysis or funnel analysis, ensure that you have configured the reports through event analysis or funnel analysis. If you choose to create new custom analysis, ensure that you have configured events.
- For events involved in event analysis or funnel analysis, client-defined event tracking has been completed. For more information, see Custom event log or Add custom event log.

Procedure

Log in to the mPaaS console, and complete the following steps: This procedure takes creating a new custom analysis for example:

1. From the navigation bar on the left, click **Mobile Analysis Service > Custom analysis**.
2. On the right page, click **Custom dashboard**.
3. In the **Add a dashboard** window, enter the dashboard name and select a template, and click **OK**.

> ⓘ **Important**
>
> Only **Empty template** is supported currently.

4. Add a custom analysis module. On the dashboard editing page, click **Add custom analysis**, and then select the analysis report creation method: add an existing analysis module or create new custom analysis.
    - **New custom analysis**:
        a. Select the analysis type, and complete the custom analysis configuration accordingly.
            - For event analysis, complete the configuration with reference to Add event analysis.
            - For funnel analysis, complete the configuration with reference to Create funnels.

        b. After completing the event or funnel analysis, click **Submit** to add the analysis module to the dashboard.
    - **Add an existing analysis module**:
        a. Select the analysis type, event or funnel.

        b. Select the event or funnel analysis module from the drop-down list, and click **Submit** to add the analysis module to the dashboard.

On the **Custom dashboard** tab, you can see the dashboard that you added. Click the dashboard, and you can see the custom analysis.

By default, the custom analysis reports display the data in last 4 days.

What to do next

You can Manage dashboards based on your business requirements, for example, to view the data in a specific time period by selecting a start and end date.

7.1.2. Manage dashboards

You can manage the custom analysis of the dashboards that you created. The management includes the following operations:

You can manage the custom analysis of the dashboards that you created. The management includes the following operations:

- Adjust dashboard page layout
- Filter the report data to show
- Modify the reports on dashboard
- Delete the reports on dashboard
- Copy the dashboard
- Delete the dashboard

Adjust dashboard page layout

Log in to the mPaaS console, and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
2. On the right page, click **Custom dashboard**.
3. Click the dashboard that you want to adjust the page layout for.
4. Click **Edit layout** in the upper right of the page.
5. Drag the analysis reports to the appropriate place on the page.
6. After you adjust the layout, click **Save layout** in the upper right of the page.

Filter the report data to show

Log in to the mPaaS console, and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
2. On the right page, click **Custom dashboard**.
3. Click the dashboard that you want to filter custom analysis on.
4. Under **Add custom analysis**, filter by analysis time period, time unit (**Day** or **Hour**), and filter label. The results are shown on the current page.

Modify the reports on dashboard

When a custom analysis report is created, a card is generated. To modify the configurations of the custom analysis, you can conduct uniform operation on the **Card management** tab.

In addition to the **Card management** tab, you can click the dashboard that you want to modify on the **Custom dashboard** tab, and then click the modification icon on the upper right of the custom report to modify the configuration.

> **Important**
>
> You cannot modify the funnel analysis.

Before you begin

- You have created event analysis or funnel analysis.
- You have the permission to modify custom analysis.

Procedure

Log in to the mPaaS console, and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
2. On the right page, click the **Card management** tab.
3. Select the custom analysis indicator, and then click **Modify** on the **Operation** column.
4. On the **Edit custom analysis** page, modify the configurations of the custom analysis.
5. After you modify the report, click **Submit**.

Delete the reports on dashboard

Delete event analysis reports

You can delete an event analysis report in the following two ways.

- On the **Card management** tab:
  i. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
  ii. On the right page, click **Card management**.
  iii. Select the custom analysis indicator, and then click **Delete** on the **Operation** column.
  iv. Click **OK**.
- On the **Custom dashboard** tab:
  i. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
  ii. On the right page, click **Custom dashboard**.
  iii. Select the dashboard that the event report to be deleted is located in.
  iv. Select the event report, and then click the deletion icon in the upper right of the report.
  v. Click **OK**.

Delete funnel analysis reports

You can delete a funnel analysis report in the following two ways.

- On the **Funnel analysis** tab:
  i. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
  ii. On the right page, click **Funnel analysis**.
  iii. Select the funnel that you want to delete, and click **Delete** on the **Operation** column.
  iv. Click **OK**.
- On the **Custom dashboard** tab:
  i. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
  ii. On the right page, click **Custom dashboard**.
  iii. Select the dashboard that the funnel report to be deleted is located in.
  iv. Select the funnel report, and then click the deletion icon in the upper right of the report.
  v. Click **OK**.

Copy the dashboard

Supports copying existing dashboards and quickly creating new custom dashboards based on existing dashboards.

On the custom dashboard page, select the dashboard you want to copy and click the **Copy** menu on the card. After copying, you can rename the dashboard and modify the dashboard information as needed.

Delete the dashboard

Delete unnecessary dashboards.

On the custom dashboard page, select the dashboard you want to copy, click the **Delete** menu on the card, and confirm the deletion.

7.2. User group management

7.2.1. About user group

Custom user grouping function refers to grouping users from the user perspective. You can group users by certain characteristics (who) and events (what) that they have done during a certain period of time (when).

Mobile Analysis Service (MAS) provides the ability to create crowds based on events. The created user group can be directly called by components such as Mobile Content Delivery Platform (MCDP) or exported for use by other components.

Usage scenarios

- Through the user grouping function, you can single out desired user groups for targeted advertisement delivery, precise marketing, and personalized push, meeting different needs of different users.
- You can also analyze the data of a certain group by using the basic and advanced analysis functions, so as to refine operation and marketing related analysis.

7.2.2. Create user group

User grouping refers to dividing users into different groups according to their behavior characteristics so as to better analyze the group attributes and behavior characteristics of different groups, and figure out the differences of different groups in user conversion in critical path. User grouping helps operators better locate the fundamental problems in product, and thus take effective improvement and optimization measures.

This topic describes how to create a user group.

Prerequisites

Related events have been created on the page displayed after you choose **Custom analysis** > **Custom configuration**. For more information, see Configure an event.

Procedure

Log on to the mPaaS console and perform the following steps:

1. From the navigation bar on the left, choose **Mobile Analysis Service** > **Custom analysis**.
2. Click the **User group** tab.
3. On the upper left of the page, click **Create a group** to enter the user group creation page.
4. Set the following information for grouping:
    - **Group name**: Name of the user group, which must be a string with no more than 20 characters.
    - **Description**: Brief description of the user group, which must be a string of not more than 100 characters.
    - **Calculation method**: You can select **Once** or **Daily routine**.
        - **Once**: Grouping is manually performed once each time when a user group is created or edited.
        - **Daily routine**: Grouping is automatically performed in the early morning on a daily basis.
    - **User dimension**: It indicates the dimension of grouping users and directly affects exported group content. You can select **User ID** or **Device ID**.
    - **Group user events**: A maximum of 10 events can be added. You can separately set the occurrence time period, occurrence times, and other filter conditions for each event.
        - In the **Time** selection box, select the time period you want to filter.
            - When **Calculation method** is **Once**, you can select any time period.
            - When **Calculation method** is **Daily routine**, you can select only a couple of fixed time periods: **Yesterday**, **Last 7 days**, **Last 14 days**, and **Last 30 days**.
        - In event selection box, select an event created in **Custom configurations**.
    - **Group tags**: Mark the group with tags. Only the groups calculated by user ID can have tags.

    > **Note**
    >
    > - Once group: A maximum of 10 once groups can be created every day.
    > - Daily routine group: A maximum of 10 daily routine groups can be created for a single app.

5. Click **Submit** to complete the group creation. After submission, the current grouping task is performed as an offline task, and calculation results are generated within a maximum of

about 2 hours.

7.2.3. Manage user group

The created user groups will be displayed in a list on the group management page. You can manage the created groups, including searching, deleting, editing user groups, exporting group user information, and viewing user behavior tracks and user details within the group.

View groups

Log in to the mPaaS console, select the target App, and enter the **Mobile Analysis Service > Custom analysis > User group** page from the left navigation pane.

The user group list displays all the calculated groups by default. You can choose the calculation method from the upper-right drop-down list to display **All**, **Once** or **Daily routine** groups, or enter a keyword in the text box to search.

The user group information includes:

- **Group name**: Click the group name to enter the group detail page where you can view the group details and edit the group information. Grouping calculation will be performed again once the filter condition is modified.
- **Number of users**: Correspond to the quantity of users in the group.
    - If the calculation method is **Once**:
        - The user count is displayed after grouping calculation is complete.
        - **Calculating...** is displayed when the calculation is in process.
    - If the calculation method is **Daily routine**:
        - The latest calculation results are presented if calculation has been performed at least once. The user quantity and time are updated upon next calculation.
        - If the first calculation is not completed, **Calculating...** is displayed.
- **User dimension**: The dimension for grouping and counting users, user ID or device ID.
- **Update time**: Display the time of latest calculation. If the latest calculation has not been finished, a dash (-) is displayed.
- **Calculation method**: The calculation method used for grouping.
    - **Once**: Grouping will only be performed once. Each time when a user group is created or edited, the group will enter the calculation queue immediately. The calculation result will be output within at most 2 hours.
    - **Daily routine**: Grouping is automatically performed once every day. When a group is created, it will enter the calculation queue immediately for the first calculation. The calculation result will be output within at most 2 hours. The subsequent calculations will be done every morning.

Edit groups

On the user group list, click the target group name to enter the group detail page, click **Edit**, modify the group information, and then click **Submit** to save the modification.

Only group name, description, and group user event can be modified. The edited group will enter the calculation queue immediately. The calculation result will be output within at most 2 hours.

Delete groups

On the user group list, select the target group, click **Delete** in the **Operations** column, and confirm deletion to remove the group.

After a group is deleted, the ongoing calculation task is stopped.

Export group user information

You can export grouping result by user ID or device ID to an Excel file. A maximum of 100 user records can be exported.

On the user group list, select the target group, click **Export** in the **Operations** column to export the corresponding user information.

> **Important**
>
> The user information cannot be exported for the group being calculated.

View user tracks/user details

On the user group list, click the number of users to go to the group user detail page where all the device/user IDs in the group are displayed. You can search users by device/user ID.

- **View user tracks**

  On the group detail page, click **View user tracks** to go to the track analysis page where you can check the user behavior track by the user/device ID. For more information, see Trajectory analysis.

- **View user details**

  On the group detail page, click **View user details** to go to the **Log playback** page where you can query the user-related history logs by the user/device ID. For more information, see Query history logs.

7.3. About custom analysis

Based on the mPaaS custom tracking points, custom analysis performs multi-dimensional instant custom analysis of client custom events through online analytical processing (OLAP) to meet the special needs of different businesses on the cloud. At the same time, different reports can be generated and saved on the front end according to different defined query conditions and query scenarios.

Multi-dimension, real-time analysis provides real-time analysis based on the client custom event model, which includes the following parts:

- Event metadata configuration management
- Custom analysis model
- Custom analysis mode
- Front-end custom analysis report

Where:

- **Event metadata configuration management** maintains the current App's various indicator data that need to be analyzed in real time. Such data is not only the foundation for analysis, but also the basis for OLAP database table creation and query.
- **Custom analysis model** and **custom analysis mode** together represent a special type of custom analysis function, such as event analysis and funnel analysis. Each model has its own particular query mode.
- **Front-end Custom analysis report** generates different reports for front-end presentation based on different models and query conditions, thus facilitating your analysis and processing.

You can create a custom dashboard to present the results of custom analysis in different types of reports. The current version supports the following types of analysis:

- **Custom event analysis:** You can upload custom analysis events based on your business conditions. mPaaS provides a powerful OLAP engine to implement multi-dimensional custom event analysis.
- **Behavior analysis:** mPaaS behavior analysis encapsulates various general behavior analysis indicators, such as retention rate, user/device access trend, proportion of devices, proportion of versions, hot page, and the distribution of visit time.

7.4. Configure custom analysis

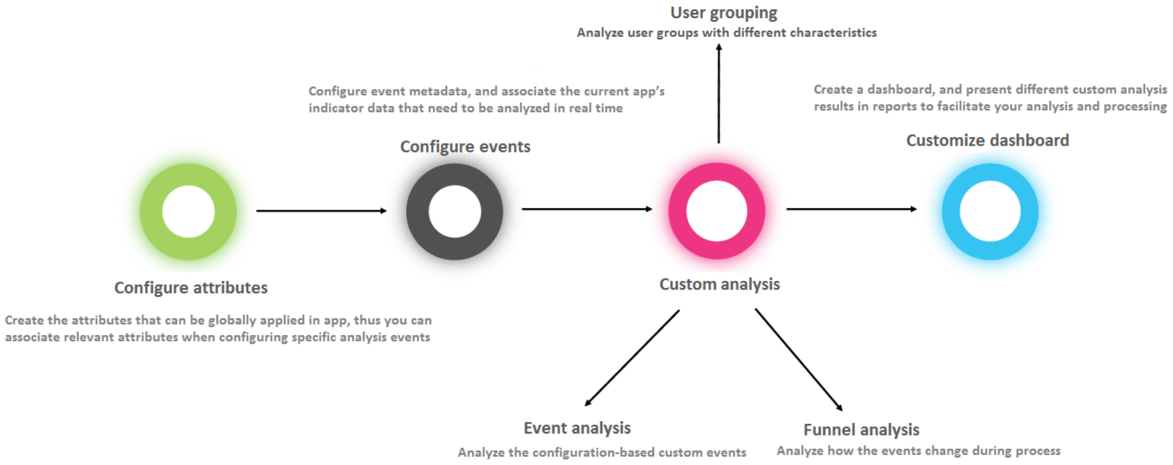
To perform custom analysis by using mPaaS Mobile Analytics Service, you must log in to the mPaaS console and complete relevant operations.

Prerequisite

You have completed the client custom event tracking. For more information, see Android custom event tracking or iOS custom event tracking.

About this task

The following flow chart illustrates the high-level procedure to conduct custom analysis on the mPaaS console:


User grouping
Analyze user groups with different characteristics
Configure event metadata, and associate the current app's indicator data that need to be analyzed in real time
Create a dashboard, and present different custom analysis results in reports to facilitate your analysis and processing
Configure events
Customize dashboard
Configure attributes
Custom analysis
Create the attributes that can be globally applied in app, thus you can associate relevant attributes when configuring specific analysis events
Event analysis
Analyze the configuration-based custom events
Funnel analysis
Analyze how the events change during process

1. Configure attributes: Create the attributes that can be globally applied in the app, thus you can associate relevant attributes when configuring specific analysis events.
2. Configure events: Configure event metadata, and associate the current app's indicator data that need to be analyzed in real time.
3. Customize analysis:
    - Event analysis: Analyze the configuration-based custom events.
    - Funnel analysis: Analyze how the events change during process.
    - User grouping: Analyze user groups with different characteristics.
4. Custom dashboard: Create a dashboard, and present different custom analysis results in reports to facilitate your analysis and processing.

7.5. Configure attributes

An event contains some information, such as the user ID that triggered the event, the App version, the device model, etc. These information are called attributes. The mobile analytics platform presets some common attributes, and also supports customizing some attributes that are applied to the global App according to business needs, so that they can be associated when configuring specific analysis events.

Create an attribute

The steps are as follows:

Log in to the mPaaS console and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
2. On the right page, click **Custom configuration**. Switch to the **Attribute** tab and click **New**.
3. In the **New attribute** window, complete the following configurations:

Field	Required or not	Description
Attribute name	Required	The Chinese name corresponding to the attribute ID, which can be customized.
Attribute ID	Required	For the correspondence between the attribute ID and the client tracking point, see Custom event log or Add custom event log.
Data type	Required	Select the data type corresponding to the current attribute from the drop-down menu. The available data types include character, integer, and floating point. ◦ In event analysis, you can count the number of times character attributes are deduplicated, and calculate the sum, average, maximum, or minimum value of floating point and integer attributes. ◦ For floating point or integer attributes, the data reported by the client will be converted to the corresponding type.
Unit	Not	Fill in the unit of the current attribute
Enumerable or not	Required	Sets whether the attribute value is enumerable.
Display status	Required	Set whether to display the current attribute, enabled by default.

4. Click **OK** to complete the creation. The attribute list will display the new attribute.

Modify attribute

In the attribute list, select the target attribute and click **Modify** in the action column to modify the attribute configuration.

Delete attribute

In the attribute list, select the target attribute and click **Delete** in the action column.

> **Important**
>
> - Preset attributes cannot be deleted.
> - If an attribute is associated with an event, the attribute will be deleted from the event. If an attribute is associated with a dashboard, the attribute cannot be deleted.

Configure attribute value translation

It not only supports the display of a single configuration attribute value, but also supports batch import of attribute value translation configuration by uploading a local Excel file containing attribute value translation records. At the same time, it supports exporting attribute value translation tables.

Add individually

1. On the **Custom Configuration** > **Attribute** page, click **Attribute Value Translation Table**.
2. On the Attribute Value Translation Table page, click **Add Attribute Value Translation** and configure the attribute translation information in the pop-up window, including the attribute, attribute value, and the display content corresponding to the attribute value.

   The displayed content cannot exceed 20 characters.
3. After completing the configuration, click **OK**. The attribute value translation table will display the new attribute value translation records.

Batch import

1. On the Attribute Value Translation Table page, click the **Import** button in the upper right corner to download the template file in the prompt window.
2. After entering the attribute value translation information based on the template, click **Upload** in the Import window to upload the file.

   After the import is successful, the attribute value translation table will display the new attribute value translation records.

What to do next

Configure events.

7.6. Configure events

An event is a description of a user's interactive behavior when using an App, such as clicking to register, adding an item to a shopping cart, completing a payment, etc. It describes what the user did and when and where.

Because custom event analysis is multi-dimensional, real-time event analysis based on custom client events, before you conduct custom event analysis, you must configure event metadata and associate the data from the attributes for realtime analysis.

Create an event

The steps are as follow:

1. Log in to the mPaaS console, from the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**, click **Custom configuration**.
2. Switch to the **Event** tab and click **New**.
3. In the **Add Event** window, complete the following configurations:
    - **Event ID**: Required. For more information about the relationship with client event tracking, see the descriptions in Custom event log or Add custom event log.
    - **Display status**: To display the status of the event analysis list or not, If you select Do not show, the event will not be displayed in the event analysis list.
    - **Event name**: Required, enter the name of added event.
4. In the **Manage attribute** column, click **Add**. In the pop-up **Add attributes** window, select one or more attributes to be associated. Click **OK**.

> **Note**
>
> All platform preset attributes are added automatically without any additional work.

5. In the **Manage attribute** column of **New event** page, check and confirm the attribute information, and then click **Submit**.

Manage events

On the event list page, you can perform the following operations on the event:

- Change **Display status** : You can manually turn the display status of this event on or off by sliding the switch.
- View event details: Click the event name to enter the event details page and view the basic information and associated properties of the event.
- Modify event information: Click the event name to enter the event details page, where you can modify the event name and add or delete associated attributes.
- Delete an event: In the **Operation column** to the right of the event, click **Delete** and confirm the deletion operation.

What to do next

On the **Event analysis** page, Add event analysis.

On the **Funnel analysis** tab, Create funnels.

7.7. Event analysis

7.7.1. Add event analysis

The Event Analysis page displays the events configured and reported in the Event Configuration page. To add a custom event analysis report, select the event you want to analyze in the **Event Analysis** tab and complete the relevant configuration.

Procedure

Log in to the mPaaS console and perfrom the following steps to add custom event analysis report:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom Analysis**.
2. On the right of the page, click **Event analysis**. Click the event name that you want to add the custom analysis for. Click **Add custom analysis**.

3. On the **Add custom analysis** page, perfrom the following configurations:
    - **Analysis name**: Enter the name of the custom analysis.
    - **Display indicators**: Select the attributes to be displayed on the report for the event. To add filter rules for indicators, perfrom the following steps:
        a. Click **Show filter rule**, and then click **+ AND** to add a filter rule.
        b. Set the parameters and the corresponding filter rules, and then click **Save**.
    - **Comparison indicators**: Select a comparative event, set its indicators, and choose the calculation method if applicable. The data is shown in the report. You can add filter rule for the comparative indicators:
        a. Click **Show filter rule**, and then click **+ AND** to add a filter rule.
        b. Set the parameters and the corresponding filter rules, and then click **Save**.
    - **Aggregation rule**: Select the dimension that the preceding indicators are aggregated from. You can add multiple aggregate rules.

      The aggregate rule equals the SQL `Group by` statement, which is used to split a dataset to multiple areas and process the data in those areas. For example, for the daily active devices indicator, if you select **Device brand** as aggregate rule, the same device brand corresponds to one curve of daily active devices.
    - **Chart type**: Select how the event analysis is presented.
4. Click **OK**.
5. To add more analysis reports for the current event, repeat the steps 3 ~ 4.

On the **Event Analysis** page, click the event name and you will see the analysis report for the event. Each time an event analysis report is created, a card will be generated, and these cards will be displayed uniformly on the **Custom Analysis** > **Custom Configuration** > **Card Management** page.

After creating a new event, wait for one minute. After the client reports the relevant logs, you can see the analysis report, but the report does not contain data before the event was created.

Related operations

For more information about managing the added custom analysis based on your business requirements, see Manage custom analysis.

7.7.2. Manage event analysis

You can manage the event analysis that you have created.

The management operations include:

- View events
- Adjust page layout
- Filter event data
- Modify event analysis reports
- Delete event analysis reports

View events

The event analysis list shows the events configured and reported in the **Custom Configuration** > **Events** page.

The steps to view reported events are as follows:

1. Log in to the mPaaS console, select the target application, and go to the **Mobile Analysis** > **Custom Analysis** > **Event Analysis** page from the left navigation bar.
2. At the top of the event analysis list, select the query time range, click **Advanced Filter** to add filter conditions and save. The event list below will show the access data of the queried reported events.

   Alternatively, enter the event name in the search box on the upper right of the page to search for events.

Adjust page layout

Log in to the mPaaS console, and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis** > **Event analysis**.
2. In the event analysis list, select the event that you want to adjust the page layout for.
3. Click **Edit layout** in the upper right of the window. Drag the analysis reports to the appropriate place on the page.
4. Click **Save layout** on the upper right to save the changes.

Filter event data

On the event analysis details page, select the analysis time period and time aggregation dimension (month, week, day, hour, the optional aggregation dimension varies depending on the length of the time period) to perform simple data filtering.

If you need to further filter data, click **Advanced Filter** and set the filtering conditions. The filtered custom analysis report results are displayed on the current page.

Modify event analysis reports

Modify the event analysis report After each event analysis report is created, a card is generated. If you want to modify the configuration of the created event analysis, you can perform unified operations on the Card Management tab.

In addition to modifying the event analysis report on the **Card Management** tab, you can also modify the configuration by clicking the edit icon in the upper right corner of the event analysis report on the **Event Analysis** tab.

- Through the Card Management tab:
  i. Go to the **Mobile Analysis Service** > **Custom Analysis** > **Custom Configuration** page from the left navigation bar.
  ii. Click the **Card Management**.
  iii. Select the indicator name of the custom analysis to be modified and click **Modify** in the action bar.
  iv. Modify the configuration on the **Modify Custom Analysis** page.
  v. After the modification is completed, click **Submit**.
- Through the Event Analysis tab:
  i. Go to the **Mobile Analysis Service** > **Custom Analysis** > **Event Analysis** page from the left navigation bar.
  ii. Select the event where the report to be modified is located to enter the event details page.
  iii. Select the event report to be modified, click the edit icon in the upper right corner of the report, modify the configuration, and click **Submit**.

Delete event analysis reports

You can delete the unnecessary analysis reports.

- On the **Manage cards** tab:
  1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
  2. On the right of the page, click the **Custom configuration** >**Manage cards** tab.
  3. Select the custom analysis indicator and click **Delete** on the operation column, and then click **OK**.
- On the **Event analysis** tab:
  1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom analysis**.
  2. On the right of the page, click **Event analysis**.
  3. Select the event that the report to be deleted is in.
  4. Select the event report and click the deletion icon (🗑) on the upper right of the report, and then click **OK**.

7.8. Funnel analysis

7.8.1. About funnel analysis

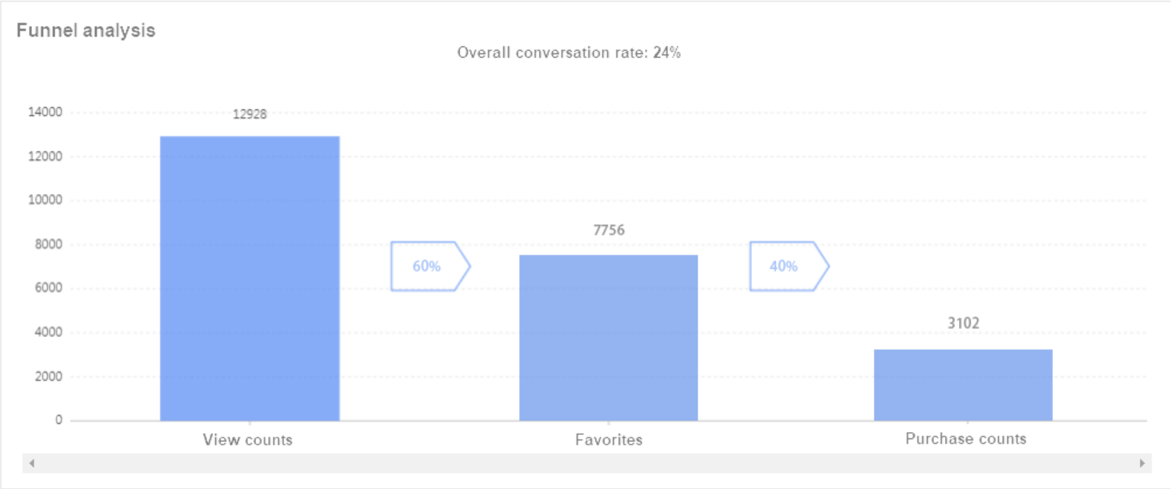
Funnel analysis shows how an event changes in a defined process by using funnel chart. The funnel analysis is an effective way to calculate conversion rates on specific user behaviors. Data and business staff can utilize the analysis to improve enterprise operations and marketing strategy.

For example, when users shop in an App, they mainly go through the following steps: search goods, browse goods, add the goods to shopping cart, and make payment. To analyze their step-by-step conversion rate in a specific period from searching goods to making payment, you can create a funnel, add steps to conduct data analysis, and view the analysis result in the funnel chart and line chart.

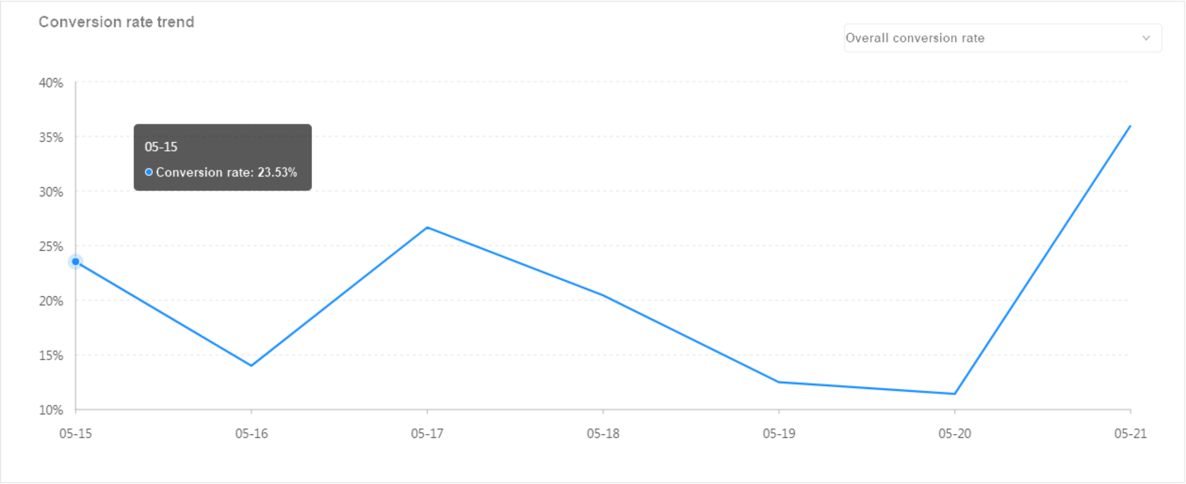
- The funnel chart shows the data at each step and conversion rates among several steps.

  Overall conversion rate = data in final step ÷ data in first step

  For example, the following funnel chart shows the UVs (distinct count) in three steps and the conversion rates among the steps.



- The line chart shows the event conversion rate in a specific time period.



- The detailed data table shows the number of occurrences of the current funnel events on each device type and the specific conversion data of the funnel.

7.8.2. Create funnels

The funnel model goes through a multi-step process and mainly analyzes the conversion and loss between different steps. It can help you locate the stage where you lose your users, figure out the cause of loss, and then improve the user conversion rate through product optimization or marketing campaigns.

This topic introduces how to create a funnel, add steps and define time window and dimension for the funnel.

Prerequisites

Because funnel analysis indicates how an event changes in a specific process by using a funnel and line chart, you must configure the associated events first. For information about configuring events, see Configure events.

About this task

- Because funnel analysis calculates the conversion rate between steps, you must configure at least two steps when you create a funnel.
- After you create a funnel, the charts show the data in T+1 days.

Procedure

Log in to the mPaaS console, and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service**> **Custom Analysis** > **Funnel analysis**.
2. Click **Create a funnel**. On the **Create a funnel** page, complete the following configurations:
   - **Funnel name**: Required, enter the funnel name.
   - **Funnel description**: Optional, enter the description about the funnel.
   - **Time window**: Required, select the time span within which all the steps in the funnel are completed.

> **Note**
>
> If users go through all the steps in the funnel within the selected time span, they reach the conversion.

- **Calculation dimension**: Required, select **User ID** or **Device ID**.

> **Note**
>
> Select the dimension that the funnel calculates and displays the data (distinct) from.

- **Step 1**: Required, select the event of step 1 from the pre-configured event list.
- **Step 2**: Required, select the event of step 2 from the pre-configured event list.

> **Note**
>
> If you want to associate more steps under the funnel, you can click **Add Step** and select the 3rd or more step events from the list of configured events.

3. Click **Submit** to create a funnel.

The funnel charts show data in T+1 days. On the **Funnel analysis** page, click **Details** on the operation column to see the funnel chart and conversion rate line chart.

7.8.3. Manage funnels

You can manage the funnels that you created. The management involves viewing funnel data, filtering funnel data and deleting funnels.

The funnel list includes not only the funnels created through the funnel analysis module, but also the funnels synchronized from Mobile Content Delivery Platform. The funnel analysis module of mobile analysis is connected with the Mobile Content Delivery Platformy component, which supports the one-click synchronization of the mobile analysis event conversion funnel data in the Mobile Content Delivery Platform activity effect dashboard to Mobile Analysis Service for subsequent conversion analysis.

About this task

You cannot modify the created funnels.

View funnel data

Log in to the mPaaS console, and perform the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Custom Analysis** > **Funnel Analysis**.
2. Select the funnel that you want to view, and click **Details** on the operation column. On the page that opens, you can see the settings of the current funnel, including the time window, calculation dimensions, and events for each step.

View and filter funnel data

1. On the funnel analysis page, select the funnel you want to view. On the page that opens, you can see the funnel conversion analysis chart, conversion rate trend chart, and detailed data.
2. In the funnel list, select the name of the funnel you want to view. On the page that opens, you can see the funnel analysis chart and conversion rate trend chart.

3. Filter and display the data displayed in the funnel analysis report.
    - In the date column at the top of the page, select the data statistics time period. For example, if you select 2022-01-10 ~ 2022-01-13, then during this time period, all data related to the step 1 event will be counted.
    - Click **Advanced Filter** on the right side of the date column, select **Parameters**, **Rules**, enter **Values**, and then **Save**. You can also click **+ OR** or **+ AND** to add more filter conditions.

Delete funnels

In the funnel list on the Funnel Analysis page, select the funnel you want to delete, click **Delete** under the action bar and confirm.

> **Note**
>
> If you delete a funnel, all the associated reports are deleted.

7.9. Trajectory analysis

By the combination of automated tracking and real-time computing, Mobile Analysis Service draws users' behaviors in an app together and arranges the behaviors in time sequence (by client time) to form user behavior trajectories. By viewing the user behavior trajectory, you can understand the users' operation path in the app, for example, when the app was started, which pages were browsed, and what operations were performed on the pages.

Trajectory analysis shows the behavior trajectory of a specific user over a period of time. Each trajectory node displays the behavior-related information according to the tracking type, including the time when the behavior occurred, platform, system version, channel, device (model), and specific behavior. You can query the user’s behavior trajectory by user ID or device ID.

View trajectory analysis report

Complete the following steps to view the trajectory analysis report:

1. Log in to the mPaaS console, and enter the **Mobile Analysis Service** > **Custom analysis** > **Trajectory analysis** tab page from the left navigation bar.
2. On the trajectory analysis page, select the query dimension (**User ID**/**Device ID**), enter the user ID or device ID, specify the time range, and then click the **Query trajectory analysis** button to view the corresponding user trajectory result.
3. 10 trajectories are displayed each time the trajectory analysis result is loaded. Click **View more trajectory** at the bottom to load the subsequent 10 trajectories. The trajectory analysis results are all displayed in one page. You can scroll the mouse to view the trajectory graph if there are too much content.

Mobile Analysis Service supports translating page addresses into user-defined page aliases through page configuration (see Page configuration). Therefore, in the behavior trajectory graph, pages with configured aliases will use page aliases instead of page addresses.

Each trajectory node displays different behavior descriptions based on the tracking type:

- Active user/device tracking: Record the behavior of starting the app.
- Page automatic tracking: Record the behavior of opening a page.
- Click automatic tracking: Record the behavior of clicking the controls on a page.

Related tracking logs

- Active device/user tracking log

Key fields: Log identifier `reportActive` (the 17th field)

Sample:

```
  2020-07-05
12:30:33.002,ip=183.xx.xx.119^country=China^province=Hubei^city=Wuhan^district=,2020-
07-05 12:30:32:532,363452B111425_IOS-
default,2.1.1,3,XElQBnytt2YDAJfdBoO/uAAq,C3638BAE-C601-4D68-ADCF-
8AB52E3D539C,368800339590,event,-,-,-,-,-,-,reportActive,1,mPaaSAliveiOS,c,-,-,-,mp_b
aseline=10.1.60.27^mp_module=mPaaSAliveiOS,-,-,XElQBnytt2YDAJfdBoO/uAAq,-,-,-,-,-,-,1
000,iPhone 11,13.3,LTE|China UNICOM,-,-,follow_system_zh-Hans-CN,-,-,-,-,VoiceOver=0^
TimeZone=Asia/Shanghai,-,750*1624,-,3,-
```

- Page automatic tracking log

  Key fields: Log identifier `auto_openPage` (the 10th field), current page ID (the 17th field)

  Sample:

```
  2020-07-05 12:30:33.007,ip=117.xx.xx.128^country= China
^province=Shanghai^city=Shanghai^district=,2020-07-05 12:30:32:902,84EFA9A281942_IOS-
default,2.4.01,2,Wo4BWJSmXjoDAOGmjiPm1HdH,AD2B567B-B0F9-4A78-A1C5-
FFFCEDB2975B,13681855793,auto_openPage,-
,SHM_H5WebViewController|SHM_H5WebViewController__Wo4BWJSmXjoDAOGmjiPm1HdH__NCTFKzA_,-
,NCTFMp8,//SHM_H5WebViewController,-,SHM_H5WebViewController,2,autotrack,c,-,1278,NCT
FMp8,referSpm=^respond=83565^openpagetime=2020-07-05 12:30:31:625^staytime=1278,-,127
8,Wo4BWJSmXjoDAOGmjiPm1HdH,-
,SHM_H5WebViewController__Wo4BWJSmXjoDAOGmjiPm1HdH__NCTFMp8_,SHM_H5WebViewController,SH
H5WebViewController,https://__bridge_loaded__/|flash,A93ac1e3a7d2c9d60bbe895acabea8bf9,
00,iPhone 6S Plus,13.3,LTE|China Mobile,-,-,follow_system_zh-Hans-CN,-,-,-,-,VoiceOve
r=0^TimeZone=Asia/Shanghai,-,1242*2208,-,298,-
```

- Click automatic tracking log

  Key fields: Log identifier `auto_click` (the 10th field), current clicked control ID (the 17th field)

  Sample:

```
  2020-07-05 12:30:33.023,ip=223.xx.xx.231^country=China^province=Shanghai^city= Shan
ghai^district=,2020-06-25 12:27:44:106,84EFA9A281942_ANDROID-
default,2.0.55,2,460001605536199|865611032563288,d5e39965-2def-45d3-8f61-44efdbb0fea6
,13501677084,auto_click,-
,MyWalletAct|MyWalletAct__W/OMuBtMJZ4DAMBf8XDfluoI__NBfkkVv_,ActivityApplicationStub,-
,//com.app.shanghai.metro.ui.mine.wallet.MyWalletAct/LinearLayout[0]:-
/FrameLayout[0]:-/TextView[0]:tvLeftTitle|-1,-,tvLeftTitle,2,autotrack,c,-,-,-,header
=D-AM,-,-,W/OMuBtMJZ4DAMBf8XDfluoI,-,-
,com.app.shanghai.metro.ui.activities.H5ActivitiesActivity,com.app.shanghai.metro.ui.mi
.wallet.MyWalletAct,com.app.shanghai.metro.ui.mine.wallet.MyWalletAct|tvLeftTitle,A5473
8d535a5eff6c8ebf3425d4d2f8,1000,1711-A01,7.1.1,LTE|cmnet,-,-,-
,20190424130858,8,1401,3602,romVersion=v076^brand=360,-,1920x1080,main,171
```

8.Performance analysis

8.1. Crash report

Crash refers that an app stops functioning properly and exits unexpectedly. When a crash occurs, the client uploads the crash data in real time, and the data is displayed on the console with seconds or minutes delay.

Crash report shows app crash data such as crash count, crash rate, and crash trend over a period of time. The crash report aggregates crash data by the causes for crash, and displays statistics on the crash count, device count, and main device models of a specific type of crash.

> **Important**
>
> To view the crash report, ensure that you have integrated the MAS SDK and configured client tracking. For more information, see Quick start and Add SDK.

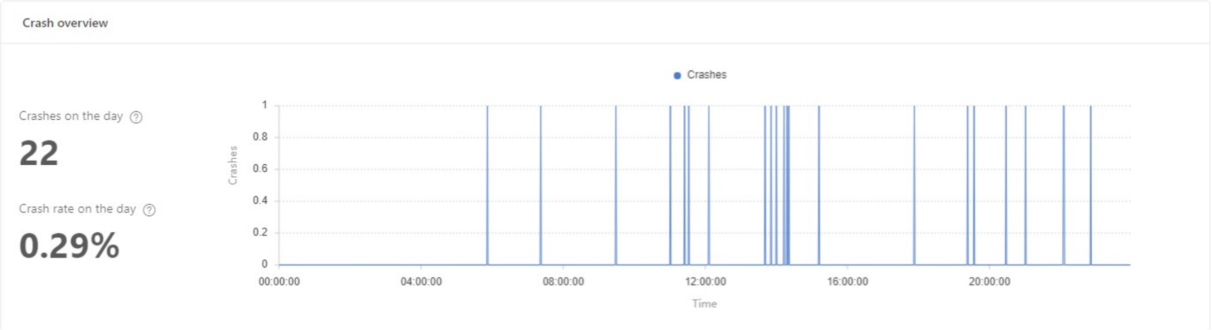
To view the crash report, complete the following steps:

1. Log in to the console, click **Products and Services** > **mPaaS**, and then select an application.
2. From the navigation bar on the left, click **Mobile Analysis Service** > **Performance analysis** > **Crash report**.
3. Filter the crash analysis data by platform, app version, and time.

Crash overview

Present the minute-level crash count and crash rate in the line chart.

- **Crashes on the day**: Total count of crash log files of the day (not deduplicated).
- **Crash rate on the day**: Total crash log files (not deduplicated) / total active user/device logs that the client uploaded.



Crash overview

Crashes on the day: 22

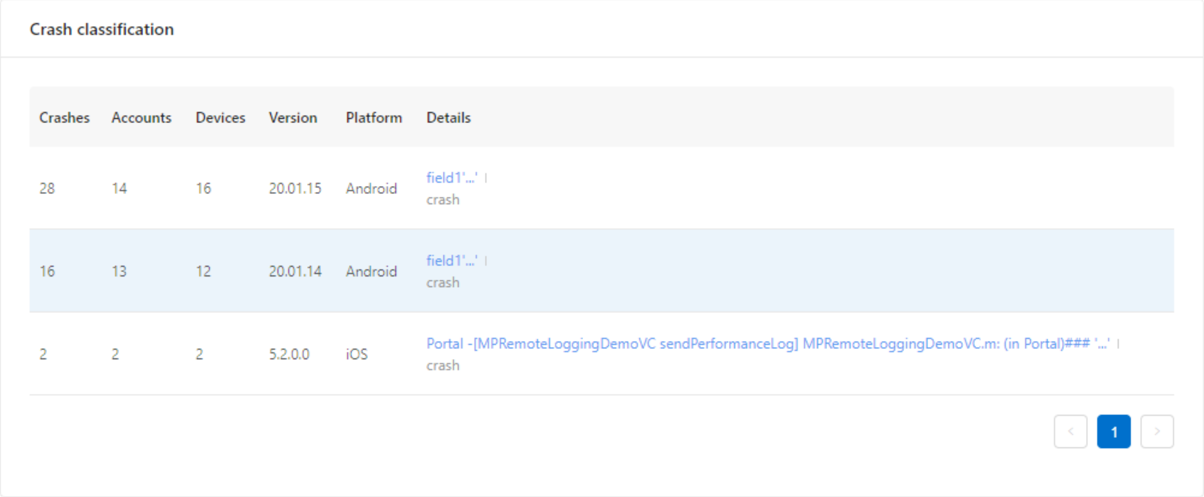
Crash rate on the day: 0.29%

Crash details

The data report in this area shows the crash data of the selected date, and classifies and counts the logs according to the reasons for the crash. mPaaS supports symbolic crash logs. iOS crash logs need to be symbolicated, but Android crash logs do not.

- **Crashes**: the total number of crashes of the same type (number of crash logs).
- **Accounts**: The number of different userIDs in the same type of crash (counted based on the userID field in the log). If the userID field is not filled in the log, the default value is 1.
- **Devices**: The number of different userIDs in the same type of crash (counted based on the device ID field in the log). If the userID field is not filled in the log, the default value is 1.

- **Version**: The version number recorded in the crash log.
- **Details**: The crash call stack recorded in the crash log.



Crash classification

Crashes	Accounts	Devices	Version	Platform	Details
28	14	16	20.01.15	Android	field1'...' | crash
16	13	12	20.01.14	Android	field1'...' | crash
2	2	2	5.2.0.0	iOS	Portal -[MPRemoteLoggingDemoVC sendPerformanceLog] MPRemoteLoggingDemoVC.m: (in Portal)### '...' | crash

> **Note**
>
> Both Android and iOS devices support viewing the corresponding stack information in the detailed data of the crash report.

Crash classification details

In the **Crash classification** section, click the content in the **Details** column to enter the crash details page which displays the error group, error sample and other information.

- **Error group**:
  - **Crashes**: Total count of the crashes of the same type (total number of log files of the same-type crash).
  - **Affected devices**: Total count of the devices on which the crash occurred, deduplicated by device ID.

    > **Note**
    >
    > If the device ID is empty or "-", the number of devices is not accumulated.

  - **Proportion of affected devices**: Total devices affected by such type of crash/total devices experience crash
  - **Device models**: The percentage of crashes on different models is displayed from high to low.
- **Error sample**: Displays the device details, log details and other information of the current sample. You can switch samples using the <and> buttons on both sides.
  - **Device details**: Displays the device ID, platform, user ID, device model, operating system version, UUID and other information of the current sample.

- **Symbol parsing**: Only for iOS devices. Displays the decompression status of the current crash log. For logs that failed to be decompressed, the failure reason will also be displayed, such as symbol table file not found, symbol table file invalid, UUID mismatch, etc. Supports iOS symbol table management for manual decompression.

  > **Note**
  >
  > Supports symbolic decompression of iOS App crash logs through symbol tables. If the symbol table file has not been uploaded after the App is released, the original log content will be displayed by default; if the dSYM symbol table file has been uploaded, the crash log symbolization will be performed in real time after the crash log is uploaded (delayed by minutes).

- **Log details**: Shows the crash log of the current sample. The log can be exported. See Crash tracking.

8.2. Lag report

Lag means that the main thread fails to complete a method for a certain period of time (2.25 seconds for Android and 2 seconds for iOS). When lag occurs, the client will upload lag information in real time. This information will be displayed on the console. The overall time delay is generally from a few seconds to a few minutes.

> **Important**
>
> To view the lag report, ensure that you have integrated the MAS SDK and configured client tracking. For more information, see Quick start and Add SDK.

You can learn the number of device lags, device lag rate, number of affected devices, and view details of each lag category.

1. Log in to the console, click **Products and Services** > **Mobile PaaS**, and then select the target App.
2. From the navigation bar on the left, click **Mobile Analysis Service** > **Performance analysis** > **Lag report**.
3. Filter the lag analysis data by platform, App version, and time.

Lag overview

Present the minute-level lag statistical data in the line chart.

- **Lag count**: The total number of lags that occurred on devices with lags monitoring enabled. The sampling rate of lags monitoring devices is 10%.
- **Lag rate**: Number of lags/total PVs of devices with lags monitoring turned on.
- **Affected devices**: Number of lags, deduplicated by user.

Lag details

The data report in this area can display the lag data of the selected date and classify the logs according to the lag causes.

- **Lag count**: Total number of lags of the same type (total number of lag logs).
- **Accounts**: Total distinct user IDs (counted by `userID` field in the logs) in the lags of the same type. If no `userID` field is available in the logs, the value is 1 by default.
- **Devices**: Total distinct device IDs (counted by device ID in the logs) in the lags of the same type. If no device ID is available in the logs, the value is 1 by default.

- **Version**: Version No. in the lag log file.
- **Details**: Lag call stack in the log file.

Lag classification details

In the lag classification list, click the content in the **Details** column to view the error details of this type of lag, including error group information and error samples.

- **Error group**:
  - **Lag count**: Total number of lags of the same type (total number of lag logs).
  - **Affected devices**: Total count of the devices on which the lag occurred, deduplicated by device ID.

    > **Note**
    >
    > If the device ID is empty or “-“, the device will not be counted.

  - **Proportion of affected devices**: The number of devices affected by this type of lag / the total number of devices experiencing lag.
  - **Device models**: The percentage of lags on different models is displayed from high to low.
- **Error sample**: Shows the device details, log details and other information. You can switch the samples by clicking <and> button.
  - **Device details**: Shows the **Device ID**, **Platform**, **User ID**, **Device model**, and **Operating system version**.
  - **Log details**: Shows the lag log of the current sample. The log can be exported. SeeLag tracking.

8.3. Stuck report

Stuck falls into startup stuck and ANR stuck. When a device gets stuck, the client uploads the stuck data in real time, and the data is displayed on the console with seconds or minutes delay.

Stuck type	Android	iOS
Startup stuck	The app fails to leave welcome page and enter home page within 30 seconds after startup.	When the app starts, the main thread does not finish executing a method within 15 seconds (30 seconds for iPhone6 and below).
ANR stuck	System ANR stuck, see ANRs on Android Official Website for definition.	When the app starts, the main thread does not finish executing a method within 10 seconds (20 seconds for iPhone6 and below).

> **Important**
>
> To view the stuck report, ensure that you have integrated the MAS SDK and configured client tracking. For more information, see Quick start and Add SDK.

The lag report allows you to learn the total number of lags, lag rate, number of affected devices, and view the details of startup lags and ANR lags by category.

To view the stuck report, complete the following steps:

1. Log in to the console, click **Products and Services** > **mPaaS**, and then select an application.
2. From the navigation bar on the left, click **Mobile Analysis Service** > **Performance analysis** > **Stuck report**.
3. Select platform, version, and time to view the stuck statistics data.

Stuck overview

Present the minute-level statistical data of startup stuck and ANR stuck in the line chart.

Indicators	Startup stuck	ANR stuck
Stuck count	Startup stuck times on the day.	ANR stuck times on the day.
Stuck rate	Number of times the application startup stuck/Number of times the application is started.	Number of times the application ANR stuck/number of times the application is started.
Affected devices	Total number of devices that have startup stuck, deduplicated by device ID.	Total number of devices that have ANR stuck, deduplicated by device ID.

Stuck details

The data report in this area shows the **Startup stuck** and **ANR stuck** data on the selected date.

- **Stuck count**: Total stuck count of the same type (total number of stuck logs).
- **Accounts**: Total distinct user IDs (count by `userID` field in the logs) in the stuck of the same type. If no `userID` field is available in the logs, the value is 1 by default.
- **Devices**: Total distinct device IDs (count by device ID in the logs) in the stuck of the same type. If no device ID is available in the logs, the value is 1 by default.
- **Version**: Version No. in the stuck log file.
- **Details**: Stuck call stack in the log file.

> **Note**
>
> - For Android models, you can view the relevant stack information in the details of the stuck report. The stack information of the startup stuck provides the stack content of all current threads for developers to troubleshoot.
> - For iOS models, you can only view the stack information corresponding to non-startup stuck in the details of the stuck report. The stuck report does not provide the stack information of the startup stuck.

Stuck classification details

In the stuck classification section, click the content in the **Details** column to enter the stuck details page which displays the error group, error sample and other information.

- **Error group**:
  - **Stuck count**: Total stuck count of the same type (total number of stuck logs).
  - **Affected devices**: Total count of the devices on which the stuck occurred, deduplicated by device ID.

    > **Note**
    >
    > If the device ID is empty or “-“, the device will not be counted.

  - **Proportion of affected devices**: Total devices affected by current stuck type/total stuck devices.
  - **Device models**: The percentage of stuck times of different models is displayed from high to low.
- **Error sample**: Displays the device details, log details and other information of the current sample. You can switch samples by the <and> buttons.
  - **Device details**: Shows the **Device ID**, **Platform**, **User ID**, **Device model**, and **Operating system version** of the current sample.
  - **Log details**: Show the stuck log of the current sample. The log can be exported. See Stuck tracking.

8.4. iOS symbol table management

Mobile Analysis Service (MAS) supports reverse parsing of iOS App crash logs through symbol tables to locate problematic code in the App, helping to improve the efficiency of troubleshooting and resolving online exceptions. It also provides iOS symbol table management functions to import and query symbol tables and perform symbol table reverse analysis tests.

About symbol table

A symbol table records the mapping relations between memory addresses and functions, file names, and line numbers. The elements in the symbol table are as follows:

```
<start address> <End address> <function> [<file name:line number>]
```

When an iOS App crashes, the crash stack in the crash log is the obfuscated binary information. The binary stack information can be reversely parsed through the symbol table, and converted into readable function names and line numbers, so the problem codes can be easily located.

Import iOS symbol table

Before you perform symbol reverse parsing on the crash log, you should upload the symbol table first. In the iOS platform, the symbol table is saved in the dSYM file which is usually named `xxx.app.dSYM`. It is recommended to back up the dSYM file every time you build or release an App.

Complete the following steps to upload the iOS symbol table:

1. In the current directory of the dSYM file, use the Linux command `tar -czvf symbol.tgz ./xxx.app.dSYM` to compress the dSYM file into a tgz package.
2. Log in to the mPaaS console, select the target App, and enter the **Mobile Analysis Service > Performance analysis > iOS symbol table management** page from the left navigation pane.

3. Click **Import**, enter the symbol table information, and upload the corresponding symbol table in the pop-up **Import symbol table** window.
    - **Version**: App version number.
    - **Module Name**: The name of the iOS App binary file, which is stored in the symbol table to identify the corresponding App binary file, so that users can match the App binary file with the App symbol table file.

      Fill in the moduleName of the App main module here. For example, if the package is `Produce.app`, then the moduleName is `Produce`.
    - **UUID**: Universally Unique Identifier (UUID), a unique identifier generated by machine.

      iOS App generates a UUID every time compilation occurs. To ensure that the log can be successfully parsed reversely, the UUID in the stack must be consistent with the UUID in the symbol table, that is, both UUIDs are from the same compilation. The crash stack information can be accurately parsed and restored only when the UUID in the imported symbol table is consistent with the UUID in the crash log.

      Fill in the UUID of the App main module here. If there are more than one UUIDs, you can use any one of them. For example, if there are armv7 and arm64 two architectures, then there will be two UUIDs, and you can fill in any one of them. In the UUID string, the “-“ must be removed and all characters must be lowercase, for example: `b7583434dc5e377bb4d8e7b69bf4c1fb`.
    - **Upload from URL**: Enter the URL of the tgz file compressed from the symbol table. If there is no symbol table file under the specified URL, an error message will be returned.
4. Click **Import** to import the symbol table.
5. Check the import status of the symbol table on the iOS symbol table management page. The status **Done** indicates that the symbol table has been successfully imported while **Failed** indicates import failure. If the symbol table import failed, you can try again according to the error message.

All the imported symbol tables are displayed on the symbol table management page, you can query them by App version.

Parsing test

Verify whether the imported symbol table file is valid through parsing test.

Complete the following steps to perform parsing test:

1. In the iOS symbol table list, select the target symbol table that was successfully imported, and click **Parsing test** under the **Operations** column.
2. In the **Raw log text** column, enter the crash log to be parsed, and click **Parse log**. Then, the **Log parsing result** column will display the parsed log text. If the reverse parsing failed, the corresponding error message and failure reason will appear, such as "UUID mismatch".

9.Log management

9.1. Extract real-time logs

The client SDK of mPaaS provides an interface to write diagnostic logs. Diagnostic logs are logs written by calling tracking interface based on your development and troubleshooting requirements. By default, the diagnostic logs are recorded only on disk and not uploaded to server.

When you troubleshoot problems, you can assign a diagnosis task to the client to extract logs. After receiving the task, the client uploads the logs to server. Then, you can download logs from the server logs through the MAS console.

prerequisites

Before you extract logs, ensure that you have completed client diagnostic log tracking. About log tracking, see iOS client diagnosis and Android client diagnosis.

Download server logs

Complete the following steps to download server logs:

1. Log in to the mPaaS console, and enter the **Mobile Analysis Service** > **Log management** > **Extract real-time logs** page.
2. Click **Add** to create a diagnosis task and complete the task configuration.

   For Android diagnosis task, if you select the log type as **Custom log path**, then you should keep in mind the following two points when filling in the log path:

   - Make sure that the custom path points to a specific file, not a directory. If it is a file under the mobile SD card, then `/storage/emulated/0/` must be added in front of the path, for example `/storage/emulated/0/Android/data/com.mpaas.aar.demo.analytics/files/mdap/upload/log.txt` .
   - The file path must include the app package name, for example, `com.mpaas.aar.demo.analytics` . In consideration of the code of conduct for apps, the package name must be consistent with the Package Name you filled in the Code configuration. No matter it is the file stored inside the app or the file in SD card, you should only pull log files in the app’s own directory.
3. After you complete the task configuration, click **OK** to generate a diagnosis task.
4. In the diagnosis task list, select the newly created task, and select the trigger channel which can be **Sync** or **Push**, and then click **Trigger** to trigger the task. When the task is successfully assigned, you can see the task status on the page updated.

   The trigger channel should be consistent with your client’s diagnosis method. Namely, if the client adopts Mobile Sync Service to pull diagnostic logs, the trigger channel must be Sync; if the client adopts Message Push Service to pull diagnostic logs, the trigger channel must be Push.
5. After the client receives the diagnosis task, the client uploads the logs to the server and updates the task status to **Processed**. Then, you can click **View** to go to the subtask page, and click **Download** to download the logs.

9.2. Query history logs

Based on your business requirements, you can query the logs that the client uploaded. The logs that you can query include behavior logs, automation logs, exception logs and performance logs. When you query the logs, you can quickly locate the logs by selecting a keyword type and applying advanced filter conditions, and then download the query result.

Prerequisites

Ensure that you have access to query the logs. For access permission issues, please contact your system administrator.

About this task

You can query any logs mentioned above by keywords.

- The query adopts full text indexing and word segmentation, so you can enter the following keywords to query the logs:
  - Letters and numbers. For example, the full text `hello Peter how are you` is segmented into `hello` , `Peter` , `how` , `are` , `you` . Enter any of the preceding words and you can get the query result that includes the keyword in the logs.
  - URLs that do not contain `path` . For example, you can search the URL of `www.google.com` . However, if you enter `www.google.com.hk/webhp` , `/webhp` cannot be indexed, so the query of `/webhp` doesn't work.
- Support searching multiple keywords at a time, with keywords separated by space (press Space key or Tab key).

> **Important**
>
> - You can query the original logs in the last 32 days (excluding diagnosis logs) and diagnosis logs in the last 7 days. However, the time range cannot be longer than 48 hours.
> - As for the analysis results, the real-time data will remain in server for 90 days, and the offline data will remain for 360 days.
> - For the client logs not uploaded, 1/4 of the backlog will be deleted to free up storage space when it reaches the upper limit. The limit is as follows:
>   - In the Android client, the total volume of tracking logs exceeds 15 MB, or the total volume of applog exceeds 50 MB.
>   - In the iOS client, the total volume of tracking logs or applog exceeds 30 MB.

Supports querying the original logs of the last 32 days (excluding diagnostic logs) and the diagnostic logs of the last 7 days, but the time span of each query cannot exceed 2 days.

Procedure

Log in to the mPaaS console and complete the following steps:

1. From the navigation bar on the left, click **Mobile Analysis Service** > **Log management** > **Query history logs**.
2. Select the log type and time order of log query, enter the keyword, and then click **Search**. By default, the logs only in the last 12 hours are searched. Under **Log list**, check the log query result.
   - Log query supports ascending and descending order.
   - To conduct quick search, click the keyword that you entered previously in the **Search history** section.

3. To specify the query time, click **Time condition** to expand the query conditions, set the start and end time of query, and select the time range and the number of logs displayed per page.

   > ? **Note**
   >
   > If the time range conflicts with the start time and the end time, the start and the end time is automatically adjusted based on the time range.

4. To add an advanced filter rule, click the plus sign (⊕) on the right of the keyword input box.
5. In the **Condition** section, select the rule type and enter the corresponding content:
   - The supported rule types are **Seed**, **Platform**, and **Version**.
   - You can add multiple advanced filter rules that work together. To remove a rule, click the delete button (⊖) on the right of the rule.
6. Under **Log list**, check the query results, where the keywords are highlighted. By default, all logs are collapsed:
   - Click the query result of a single log to expand and check the full content.
   - Select **Show all** to expand the query results of all the logs.
7. Click the download button (↓) to download the query result that is saved in an `.xlsx` file.

Related links

- Android tracking
- iOS tracking
- Log tracking

9.3. Configure upload switch

The switch configuration feature allows you to modify the trigger condition for automatic log uploading based on the switch value delivered by the server, to dynamically control log uploading.

Mobile Analysis Service (MAS) implements tracking based on the client SDK, collects user behaviors, app performance data, and other related data to generate logs, and then reports the logs to the server. Then, mPaaS generates various metrics reports and dashboards based on real-time or offline computing for your referrence.

mPaaS charges the traffic consumed for tracking log uploading. To prevent unnecessary costs, you can manage log uploading by using the switch configuration feature. For more information about traffic fees for log uploading, see Pay-as-you-go mode.

Add tracking configuration

1. In the mPaaS console, go to the **Mobile Analysis Service** > **Log management** > **Configure upload switch** page, and click **Tracking configuration** in the upper-left corner to enter the log switch list page.
2. Click **New business**, and configure the tracking information.

  - **Business code**: Enter a business code. For custom behavior tracking, the business code is as follows:
    - Android: Correspond to the `behaviourPro` set by calling `demoBehavior.setBehaviourPro("Pay")` on the client.
    - iOS: It defaults to `behavior`. You can set a custom value for the client using the `bizType` parameter in the `writeLogWithActionId` API. For more information, see Android business codes or iOS business codes.
  - **Business name**: Enter the description of the tracking business. The value is customizable.
  - **Log header**: Enter the value of **field 01** in the log model. This parameter is used to distinguish different log types. For example, the log header for custom behavior log is `D-VM`. For more information, see Log tracking.

3. Click **Add**. The tracking configuration is added. To specify more configurations, see Configure upload switch.

Modify tracking configuration

The new tracking configuration is displayed on the log switch list page. You can set the log upload switch and network, modify and delete the tracking configuration.

Set log upload switch

Enable or disable log uploading. Once the log upload switch is turned on, the tracking logs of the business will be automatically uploaded.

Set upload network

Set the network environment where tracking logs are uploaded. The network can be **All networks** (2G, 3G, 4G, Wi-Fi and others) or **Wi-Fi** only.

Modify tracking configuration

To modify tracking configurations, click **Modify** in the **Operations** column. You can modify the following configurations of the tracking point:

- **Business code**: Correspond to the business code that you set when adding the tracking.
- **Description**: Correspond to the business name that you set when adding the tracking.
- **Log header**: Correspond to the log header that you set when adding the tracking.
- **Minimum upload level**: Logs are classified into levels 1, 2, and 3, which are sorted by importance in descending order. For example, if this parameter is set to 2, logs of levels 1 and 2 will be uploaded, but logs of level 3 will not. For custom behavior event tracking, the default value is 2. For iOS apps, you can set the log level when configuring events for the client. For more information, see iOS custom behavior event log.
- **Uploaded logs**: Tracking logs of the client are first written to local files. When the number of local logs with the specified business code reaches the threshold defined by this parameter, the logs are automatically uploaded to MAS. For more information, see Automatic log upload for Android or Automatic log upload for iOS.
- **Log upload rate**: Specify the rate of users whose logs are to be uploaded, in permillage. For example, the value 1000 indicates that logs of all users are to be uploaded.
- **Strategy**: If the **Upload in background** strategy is selected:
  - Android: Logs are automatically uploaded after an app runs in the background for more than half an hour.
  - iOS: Logs are automatically uploaded when an app switches to the background. For more information, see Log upload for Android or Log upload for iOS.

Initialize business

Logs written by the client in local files are automatically uploaded to the MAS server when certain conditions are met. Such conditions depend on the default tracking configuration. After creating an app, you can use the **Initialize business** function to synchronize the default tracking configuration to the App. The procedure is as follows:

1. Click **Initialize business**. In the pop-up confirmation dialog box, click **OK**.
2. When the initialization is successfully done, you can see the default tracking configuration list.

> **Note**
>
> The default tracking configuration will apply to your app even if you do not click **Initialize business** to synchronize it. The only difference is that it will not be displayed in the tracking configuration list.

Mock

In addition to adding and modifying tracking configurations on pages, you can use `JSON` to add and configure tracking. However, we **do not recommend** you to use this function for the following reasons:

- The mock function only facilitates tracking configuration but does not provide any other functions.
- If you configure tracking for the same business through both page operations and the mock function, the mock configuration takes precedence.
- If you are not familiar with the tracking configuration format, you may use the mock function incorrectly. As a result, the default automatic log upload function may be affected.

Add mock configurations

To add mock configurations, perform the following steps:

1. On the **Configure upload switch** tab, click **Add Mock configuration**.
2. Fill the configuration information.
    - **Version**: Enter the version of your app.
    - **Platform**: Select the app platform.
    - **Value**: Enter the configurations in JSON. For more information, see Mock configuration format.
3. After you make sure that all the information is correct, click **OK** to add the mock configurations.

Mock configuration format

The mock configuration format is as follows:

```
{
  "Log header 1": {
    "Business code 1": {
      "write": "yes",
      "send": [
        "wifi"
      ],
      "maxLogCount": 50,
      "level": 3,
      "uploadRate": 1000,
      "event": [
        "gotoBackground"
      ]
    },
    "Business code 2": {
        ...
    },
    ...
  },
  "Log header 2": {
    ...
  },
  ...
}
```

- Log header: Correspond to the log header in the tracking configuration.
- Business code: Correspond to business code in the tracking configuration.
- write: Correspond to the upload switch in the tracking configuration. To turn on the switch, set this parameter to `"yes"` . Otherwise, set this parameter to `"no"` .
- send: Correspond to network in the tracking configuration. To upload logs on all networks, set this parameter to `["2g", "3g", "4g", "wifi"]` . To upload logs only on Wi-Fi, set this parameter to `["wifi"]` .
- maxLogCount: Correspond to the uploaded logs in the tracking configuration.
- level: Correspond to the minimum upload level in the tracking configuration.
- uploadRate: Correspond to the log upload rate in the tracking configuration.
- event: Correspond to the strategy in the tracking configuration. To use the "Upload in background" strategy, you can set this parameter as `["gotoBackground"]` . Otherwise, set this parameter to [].

9.4. Client diganosis

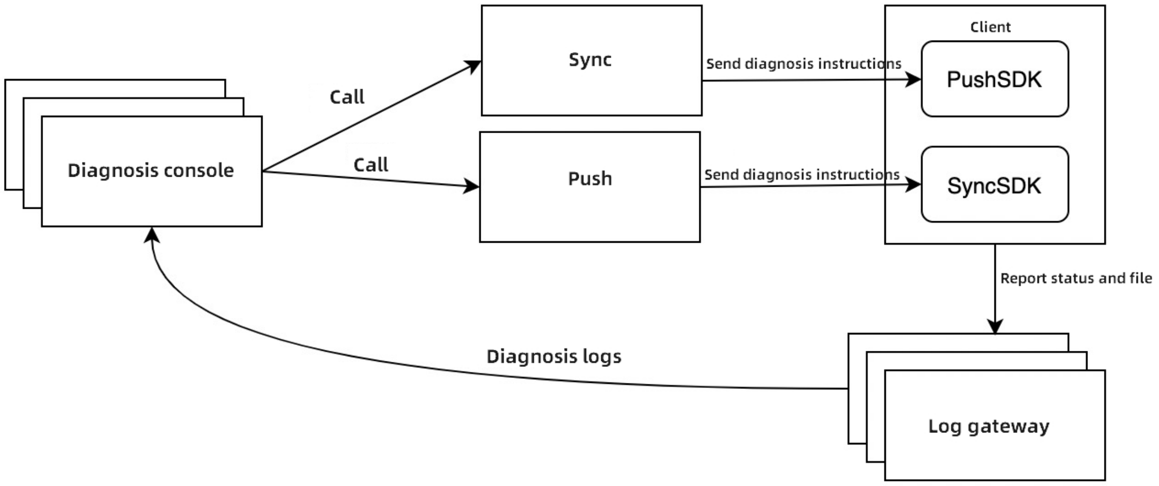
9.4.1. About client diagnosis

Client diagnosis refers to obtaining the diagnosis logs written by client from server, and then diagnosing client problems based on the diagnosis logs.

You can write diagnostic logs to key client links. When online problems occur, you can issue diagnostic instructions to users on the console to pull diagnostic logs to troubleshoot the problem. The client diagnostic function supports pulling diagnostic logs through the channel: About Mobile Sync Service or About Message Push Service.

Principles framwork

The console sends log diagnostic instructions according to different diagnostic channels (Message Push Service or Message Push Service). After receiving the diagnostic instructions, the client reports the diagnostic log to the log gateway. When the diagnostic file is successfully reported, the diagnostic file can be downloaded in the console. The basic framework is as follows:



Integrate and use

For how to integrate and use the client diagnosis function on Android clients, see Android client diagnosis; for how to integrate and use the client diagnosis function on iOS clients, see Add iOS SDK and Use iOS SDK.

9.4.2. Android client diagnosis

This topic describes how to use the client diagnosis function on Android clients. In the console, you can send diagnosis tasks through two channels: Message Push Service (Push) and Mobile Sync Service (Sync).

To implement Android client diagnosis, perform the following steps:

1. Initialize diagnosis service
2. Write diagnosis logs to client
3. Pull diagnosis logs in the console

Prerequisites

- Your App has the MAS component integrated, and log reporting function works correctly. For more information about the accessing steps, see Access Mobile Analysis Service.
- Your App has the Message Push Service or Mobile Sync Service integrated.
  - To use client diagnosis through Message Push Service channel, see Quick start Integrate Message Push Service into Android to learn the steps .
  - To use client diagnosis through Mobile Sync Service channel, see Integrate with Android Integrate Mobile Sync Service into Android to learn the steps.

Initialize diagnosis service

The client diagnosis function allows you to pull diagnosis logs through the Message Push Service and Mobile Sync Service channels. The methods for initializing different channels are different.

Use Message Push Service

If you use the Message Push Service channel, you must perform the following steps after the App starts to complete initialization:

- Initialization. Initialize the Message Push Service service and establish a persistent connection between the client and the mobile Message Push Service gateway. This persistent connection is maintained by the Message Push Service SDK, i.e., a self-built channel. Initialize the Message Push Service SDK
- Bind user ID. This user ID is customized by the developer and can be either the user ID of the real user system or other parameters that can be mapped to each user, such as account number, mobile phone number, etc.Report User ID

Use Mobile Sync Service

If you use the Mobile Sync Service channel, you must call the following methods after the App starts to complete initialization.

```
// Set the userId.
MPLogger.setUserId(String userId);
// To initialize the Sync channel, you must set the userId first. Otherwise, initializa
tion will fail.
MPDiagnose.initSyncChannel(Context context);
```

Write diagnosis logs to client

After the client receives the diagnosis task, the client uploads only the logs written by using MPLogger, the logging tool of mPaaS. Therefore, we recommend that you use MPLogger instead of `android.util.Log` to write logs. MPLogger provides a logging method that is similar to native logging, as shown in the following code:

```
void verbose(String tag, String msg);
void debug(String tag, String msg);
void info(String tag, String msg);
void warn(String tag, String msg);
void warn(String tag, Throwable t);
void warn(String tag, String msg, Throwable tr);
void error(String tag, String msg);
void error(String tag, Throwable t);
void error(String tag, String msg, Throwable t);
void print(String tag, String msg);
void print(String tag, Throwable t);
```

As for the logs written by MPLogger, debug packages are displayed in `Logcat` while release packages not. The diagnosis logs are stored in the following directories on the device:

- Debug packages: `/sdcard/[PackageName]/applog` , if the logs failed to be written to this directory (for example, the App targetSdkVersion is higher than 30), they will be written to the storage directory for release packages.

> **Important**

> When **targetSdkVersion ≥ 30** and **phone system version ≥ 11**, the storage directory is: `/storage/emulated/0/Android/data/com.mpaas.demo/cache/[PackageName]/applog/` .

- Release packages: `/data/data/[PackageName]/files/applog`

Pull diagnosis logs in the console

In the console, you can pull logs printed by the logging tool of mPaaS to quickly analyze App crashes and exceptions that specified devices or users encounter.

Step 1: Create a log pull task

1. Log on to the mPaaS console and then select an App.
2. In the left-side navigation pane, choose **Mobile Analysis Service** > **Log Management**.
3. On the **Pull real-time logs** tab, click **Add**.
4. Enter the information about the task. **User ID** uniquely identifies a user who logs on to your App. You can set the user ID by calling `MPLogger.setUserId(String userId)` or the Message Push Service's user ID reporting method.
5. Click **OK** to complete creating the log pull task.

Step 2: Trigger the log pull task

1. In the log pull task list, find the newly created task, and then select **Trigger channel**. Click **Trigger** in the **Operations** column.
2. Wait until the task status is updated.
    - **Task processed**: In this state, you can click **View** to download the diagnosis logs.
    - **Push/Sync service successfully called**: This status indicates that the instruction for uploading a diagnosis log has been delivered, but the client has not received the instruction or uploaded the diagnosis log.

      In this case, check whether your App process still exists in the system. If your App process does not exist in the system, restart your App. If the task status does not change after you restart your App, troubleshoot as instructed in the following sections.

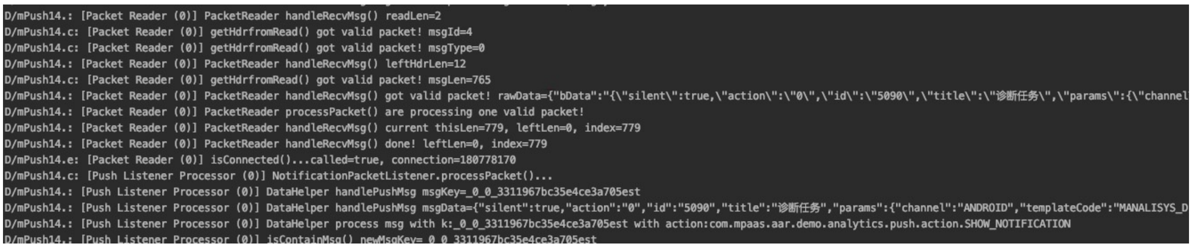
Troubleshooting

If you fail to pull logs, you can troubleshoot through the following steps. The troubleshooting methods vary with the diagnosis task delivery channels.

For Push channel

To troubleshoot log pull failures, perform the following steps:

1. In the mPaaS console, go to the **Message Push Service** page, and push a normal message to your App by userId to Check if the Push channel works correctly.
2. If the Push channel works correctly, clear logs in `Logcat` , and then switch to the push process. In the mPaaS console, find the diagnosis task and then click **Trigger** in the **Operations** column. View the log in `Logcat` .
3. Search `mPush14` to view whether any diagnosis message is received. If no diagnosis messages are received, check whether the Push channel is disconnected.

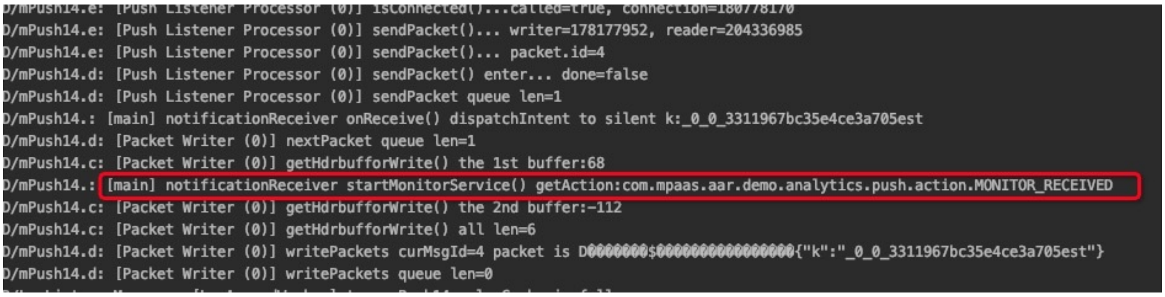


```
D/mPush14.: [Packet Reader (0)] PacketReader handleRecvMsg() readLen=2
D/mPush14.c: [Packet Reader (0)] getHdrfromRead() got valid packet! msgId=4
D/mPush14.c: [Packet Reader (0)] getHdrfromRead() got valid packet! msgType=0
D/mPush14.: [Packet Reader (0)] PacketReader handleRecvMsg() leftHdrLen=12
D/mPush14.c: [Packet Reader (0)] getHdrfromRead() got valid packet! msgLen=765
D/mPush14.: [Packet Reader (0)] PacketReader handleRecvMsg() got valid packet! rawData={"bData":"{\"silent\":true,\"action\":\"0\",\"id\":\"5090\",\"title\":\"诊断任务\",\"params\":{\"channel
D/mPush14.: [Packet Reader (0)] PacketReader processPacket() are processing one valid packet!
D/mPush14.: [Packet Reader (0)] PacketReader handleRecvMsg() current thisLen=779, leftLen=0, index=779
D/mPush14.: [Packet Reader (0)] PacketReader handleRecvMsg() done! leftLen=0, index=779
D/mPush14.e: [Packet Reader (0)] isConnected()...called=true, connection=180778170
D/mPush14.c: [Push Listener Processor (0)] NotificationPacketListener.processPacket()...
D/mPush14.: [Push Listener Processor (0)] DataHelper handlePushMsg msgKey=_0_0_3311967bc35e4ce3a705est
D/mPush14.: [Push Listener Processor (0)] DataHelper handlePushMsg msgData={"silent":true,"action":"0","id":"5090","title":"诊断任务","params":{"channel":"ANDROID","templateCode":"MANALISYS_D
D/mPush14.: [Push Listener Processor (0)] DataHelper process msg with k:_0_0_3311967bc35e4ce3a705est with action:com.mpaas.aar.demo.analytics.push.action.SHOW_NOTIFICATION
D/mPush14.: [Push Listener Processor (0)] isContainMsg() newMsgKey=_0_0_3311967bc35e4ce3a705est
```

4. After a diagnosis message is received, check whether the following log exists, which indicates whether the diagnosis message is forwarded to `MonitorService`.

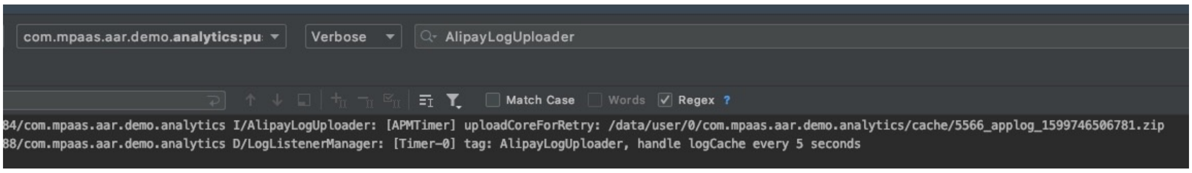
   If `ClientMonitorService` is not found, use the mPaaS plug-in to update the SDK:

   - If the baseline version is 10.1.32, upgrade the baseline to 10.1.68, and then upgrade the component to the latest version.
   - If the baseline version is 10.1.60 or 10.1.68, upgrade the component to the latest version.



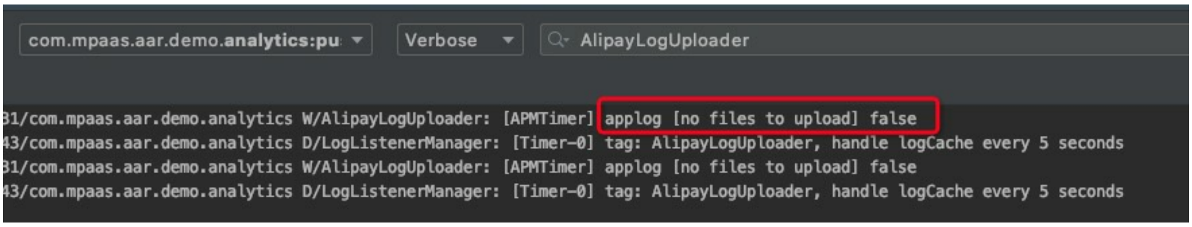
```
D/mPush14.e: [Push Listener Processor (0)] isConnected()...called=true, connection=180778170
D/mPush14.e: [Push Listener Processor (0)] sendPacket()... writer=178177952, reader=204336985
D/mPush14.e: [Push Listener Processor (0)] sendPacket()... packet.id=4
D/mPush14.d: [Push Listener Processor (0)] sendPacket() enter... done=false
D/mPush14.d: [Push Listener Processor (0)] sendPacket queue len=1
D/mPush14.: [main] notificationReceiver onReceive() dispatchIntent to silent k:_0_0_3311967bc35e4ce3a705est
D/mPush14.d: [Packet Writer (0)] nextPacket queue len=1
D/mPush14.c: [Packet Writer (0)] getHdrbufforWrite() the 1st buffer:68
D/mPush14.: [main] notificationReceiver startMonitorService() getAction:com.mpaas.aar.demo.analytics.push.action.MONITOR_RECEIVED
D/mPush14.c: [Packet Writer (0)] getHdrbufforWrite() the 2nd buffer:-112
D/mPush14.c: [Packet Writer (0)] getHdrbufforWrite() all len=6
D/mPush14.d: [Packet Writer (0)] writePackets curMsgId=4 packet is D�������$����������������{"k":"_0_0_3311967bc35e4ce3a705est"}
D/mPush14.d: [Packet Writer (0)] writePackets queue len=0
```

5. After `MonitorService` starts, search `AlipayLogUploader` to check whether any local diagnosis log exists. If the following log appears, a local diagnosis log is available for uploading.



```
84/com.mpaas.aar.demo.analytics I/AlipayLogUploader: [APMTimer] uploadCoreForRetry: /data/user/0/com.mpaas.aar.demo.analytics/cache/5566_applog_1599746506781.zip
88/com.mpaas.aar.demo.analytics D/LogListenerManager: [Timer-0] tag: AlipayLogUploader, handle logCache every 5 seconds
```

   If the following log appears, no local diagnosis logs exist.



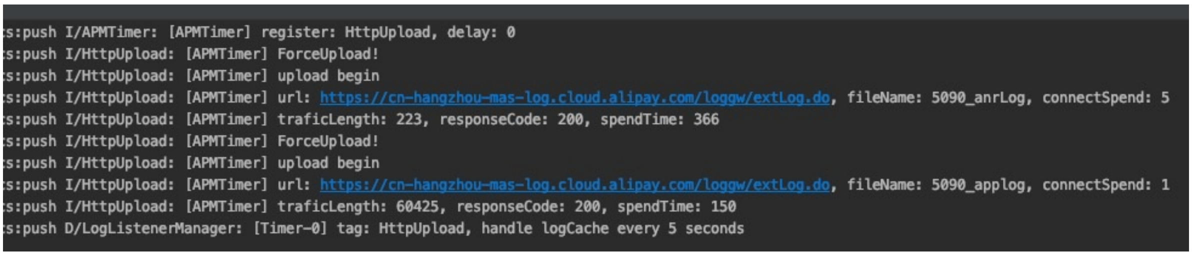
```
31/com.mpaas.aar.demo.analytics W/AlipayLogUploader: [APMTimer] applog [no files to upload] false
43/com.mpaas.aar.demo.analytics D/LogListenerManager: [Timer-0] tag: AlipayLogUploader, handle logCache every 5 seconds
31/com.mpaas.aar.demo.analytics W/AlipayLogUploader: [APMTimer] applog [no files to upload] false
43/com.mpaas.aar.demo.analytics D/LogListenerManager: [Timer-0] tag: AlipayLogUploader, handle logCache every 5 seconds
```

   Check the following items:

   - For real-time log pull tasks created in the console, select a time range longer than 1 hour. Ensure that the App runs and a diagnosis log is generated for the App during the selected time range.
   - Whether the following permissions are declared and dynamically applied for the App:

   ```
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   ```

6. After you confirm that a local diagnosis log exists, search `HttpUpload` to check whether the log is uploaded. If `responseCode` is 200, the log is uploaded.

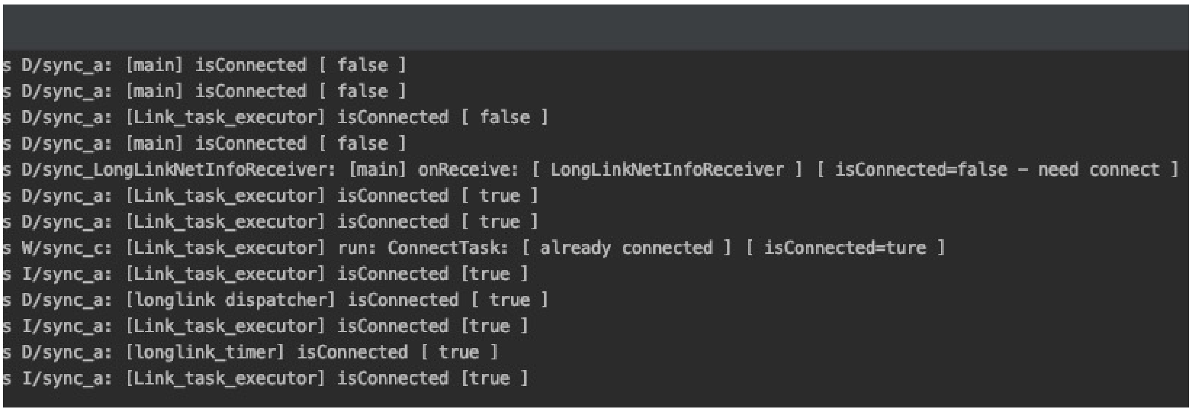


```
s:push I/APMTimer: [APMTimer] register: HttpUpload, delay: 0
s:push I/HttpUpload: [APMTimer] ForceUpload!
s:push I/HttpUpload: [APMTimer] upload begin
s:push I/HttpUpload: [APMTimer] url: https://cn-hangzhou-mas-log.cloud.alipay.com/loggw/extLog.do, fileName: 5090_anrLog, connectSpend: 5
s:push I/HttpUpload: [APMTimer] traficLength: 223, responseCode: 200, spendTime: 366
s:push I/HttpUpload: [APMTimer] ForceUpload!
s:push I/HttpUpload: [APMTimer] upload begin
s:push I/HttpUpload: [APMTimer] url: https://cn-hangzhou-mas-log.cloud.alipay.com/loggw/extLog.do, fileName: 5090_applog, connectSpend: 1
s:push I/HttpUpload: [APMTimer] traficLength: 60425, responseCode: 200, spendTime: 150
s:push D/LogListenerManager: [Timer-0] tag: HttpUpload, handle logCache every 5 seconds
```

For Sync channel

If you fail to pull logs, you can troubleshoot through the following steps.

1. Clear logs in `Logcat`, and then switch to the main process. Initialize the Sync channel and then search `isConnected` to check whether the Sync channel is connected to the client.

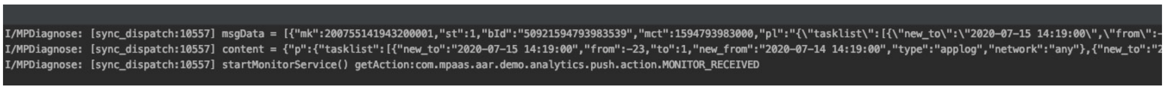


```
s D/sync_a: [main] isConnected [ false ]
s D/sync_a: [main] isConnected [ false ]
s D/sync_a: [Link_task_executor] isConnected [ false ]
s D/sync_a: [main] isConnected [ false ]
s D/sync_LongLinkNetInfoReceiver: [main] onReceive: [ LongLinkNetInfoReceiver ] [ isConnected=false - need connect ]
s D/sync_a: [Link_task_executor] isConnected [ true ]
s D/sync_a: [Link_task_executor] isConnected [ true ]
s W/sync_c: [Link_task_executor] run: ConnectTask: [ already connected ] [ isConnected=ture ]
s I/sync_a: [Link_task_executor] isConnected [true ]
s D/sync_a: [longlink dispatcher] isConnected [ true ]
s I/sync_a: [Link_task_executor] isConnected [true ]
s D/sync_a: [longlink_timer] isConnected [ true ]
s I/sync_a: [Link_task_executor] isConnected [true ]
```

2. In the mPaaS console, find the diagnosis task, and then click **Trigger** in the **Operations** column. Select `MPDiagnose` to check whether any diagnosis messages are received and whether `MonitorService` is enabled.

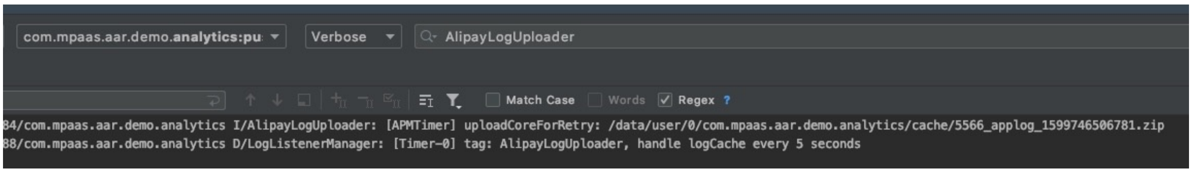
   If no diagnosis messages are received, check whether the specified userId is the same as the userId in the diagnosis task. If no `ClientMonitorService` is found, use the mPaaS plug-in to update the SDK.

   - If the baseline version is 10.1.32, upgrade the baseline to 10.1.68, and then upgrade the component to the latest version.
   - If the baseline version is 10.1.60 or 10.1.68, upgrade the component to the latest version.



```
I/MPDiagnose: [sync_dispatch:10557] msgData = [{"mk":200755141943200001,"st":1,"bId":"50921594793983539","mct":1594793983000,"pl":"{\"tasklist\":[{\"new_to\":\"2020-07-15 14:19:00\",\"from\":-
I/MPDiagnose: [sync_dispatch:10557] content = {"p":{"tasklist":[{"new_to":"2020-07-15 14:19:00","from":-23,"to":1,"new_from":"2020-07-14 14:19:00","type":"applog","network":"any"},{"new_to":"2
I/MPDiagnose: [sync_dispatch:10557] startMonitorService() getAction:com.mpaas.aar.demo.analytics.push.action.MONITOR_RECEIVED
```

3. After `MonitorService` starts, switch to the push process. Select `AlipayLogUploader` to check whether any local diagnosis log exists. If the following log appears, a local diagnosis log is available for uploading.



com.mpaas.aar.demo.analytics:pu

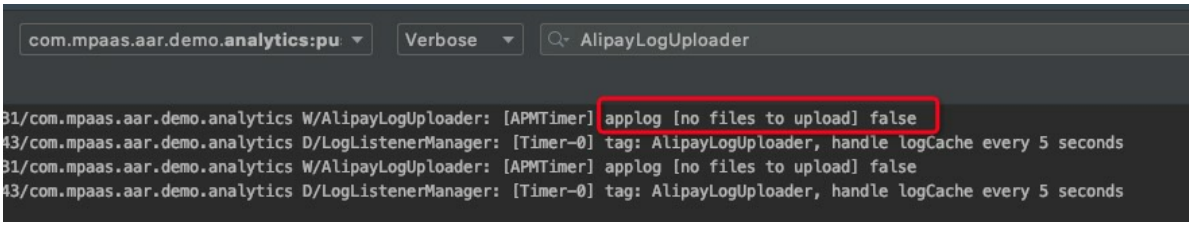
Verbose

AlipayLogUploader

☐ Match Case ☐ Words ☑ Regex

```
84/com.mpaas.aar.demo.analytics I/AlipayLogUploader: [APMTimer] uploadCoreForRetry: /data/user/0/com.mpaas.aar.demo.analytics/cache/5566_applog_1599746506781.zip
88/com.mpaas.aar.demo.analytics D/LogListenerManager: [Timer-0] tag: AlipayLogUploader, handle logCache every 5 seconds
```

   If the following log appears, no local diagnosis logs exist.



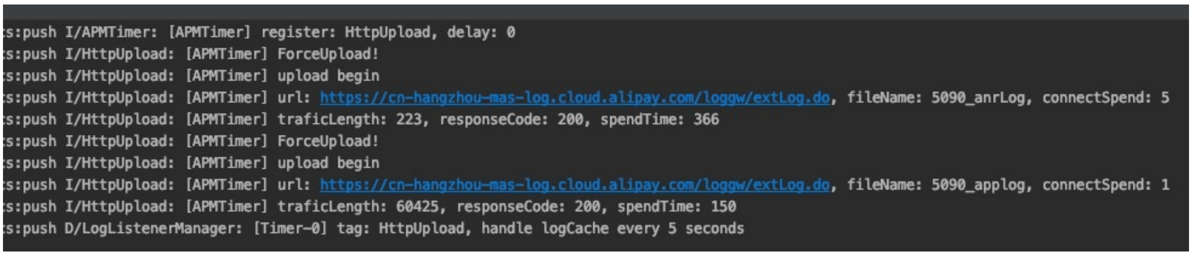
```
31/com.mpaas.aar.demo.analytics W/AlipayLogUploader: [APMTimer] applog [no files to upload] false
43/com.mpaas.aar.demo.analytics D/LogListenerManager: [Timer-0] tag: AlipayLogUploader, handle logCache every 5 seconds
31/com.mpaas.aar.demo.analytics W/AlipayLogUploader: [APMTimer] applog [no files to upload] false
43/com.mpaas.aar.demo.analytics D/LogListenerManager: [Timer-0] tag: AlipayLogUploader, handle logCache every 5 seconds
```

Check the following items:

- For real-time log pull tasks created in the console, select a time range longer than 1 hour. Ensure that the App runs and a diagnosis log is generated for the App during the selected time range.
- Whether the following permissions are declared and dynamically applied for the App:

  ```
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  ```

4. After you confirm that a local diagnosis log exists, search `HttpUpload` to check whether the log is uploaded. If `responseCode` is 200, the log is uploaded.



```
s:push I/APMTimer: [APMTimer] register: HttpUpload, delay: 0
s:push I/HttpUpload: [APMTimer] ForceUpload!
s:push I/HttpUpload: [APMTimer] upload begin
s:push I/HttpUpload: [APMTimer] url: https://cn-hangzhou-mas-log.cloud.alipay.com/loggw/extLog.do, fileName: 5090_anrLog, connectSpend: 5
s:push I/HttpUpload: [APMTimer] traficLength: 223, responseCode: 200, spendTime: 366
s:push I/HttpUpload: [APMTimer] ForceUpload!
s:push I/HttpUpload: [APMTimer] upload begin
s:push I/HttpUpload: [APMTimer] url: https://cn-hangzhou-mas-log.cloud.alipay.com/loggw/extLog.do, fileName: 5090_applog, connectSpend: 1
s:push I/HttpUpload: [APMTimer] traficLength: 60425, responseCode: 200, spendTime: 150
s:push D/LogListenerManager: [Timer-0] tag: HttpUpload, handle logCache every 5 seconds
```

9.4.3. iOS client diagnosis

9.4.3.1. Add SDK

This topic describes how to add the **Diagnosis** or **Push** SDK based on native project with CocoaPods.

The client diagnosis function allows you to pull logs through **Sync** or **Push** channel. You can select any one of the channels as needed and select the **Diagnosis** or **PUSH** module during access. Sync channel, which depends on the Mobile Sync Service, has a higher success rate on diagnosis instruction delivery. Push channel, which depends on the Message Push Service, requires that the users are online during diagnosis instruction delivery.

Prerequisites

You have completed the general access steps with reference to Integrate by using CocoaPods based on the existing project.

Add an SDK

The following content describes how to access the diagnosis feature based on different log pull methods.

Use Mobile Sync Service (Sync)

Configure `mPaaS_pod "mPaaS_Diagnosis"` in `Podfile`. For how to configure the `Podfile` file, see Connect to mPaaS based on native framework and use CocoaPods.

Use Message Push Service (Push)

Configure `mPaaS_pod "mPaaS_Push"` in `Podfile`. For how to configure the `Podfile` file, see Connect to mPaaS based on native framework and use CocoaPods.

9.4.3.2. Use SDK

Perform the following steps to diagnose the iOS client problems:

1. Initialize diagnosis service
2. Set user ID
3. Write diagnosis logs
4. View local diagnosis logs
5. Obtain online user diagnosis logs

Initialize diagnosis service

Before using the diagnosis function, initialize it first.

```
#import <MPDiagnosis/MPDiagnoseAdapter.h>
[MPDiagnoseAdapter initDiagnose];
```

The diagnosis service supports pulling logs through **Sync** or **Push** method. Codes for initializing the diagnosis service vary with the log pulling method.

Use Mobile Sync Service (Sync)

If you use the Sync method to pull diagnostic logs, you need to initialize the service before using the diagnostic function.

```
#import <MPDiagnosis/MPDiagnoseAdapter.h>
[MPDiagnoseAdapter initDiagnose];
```

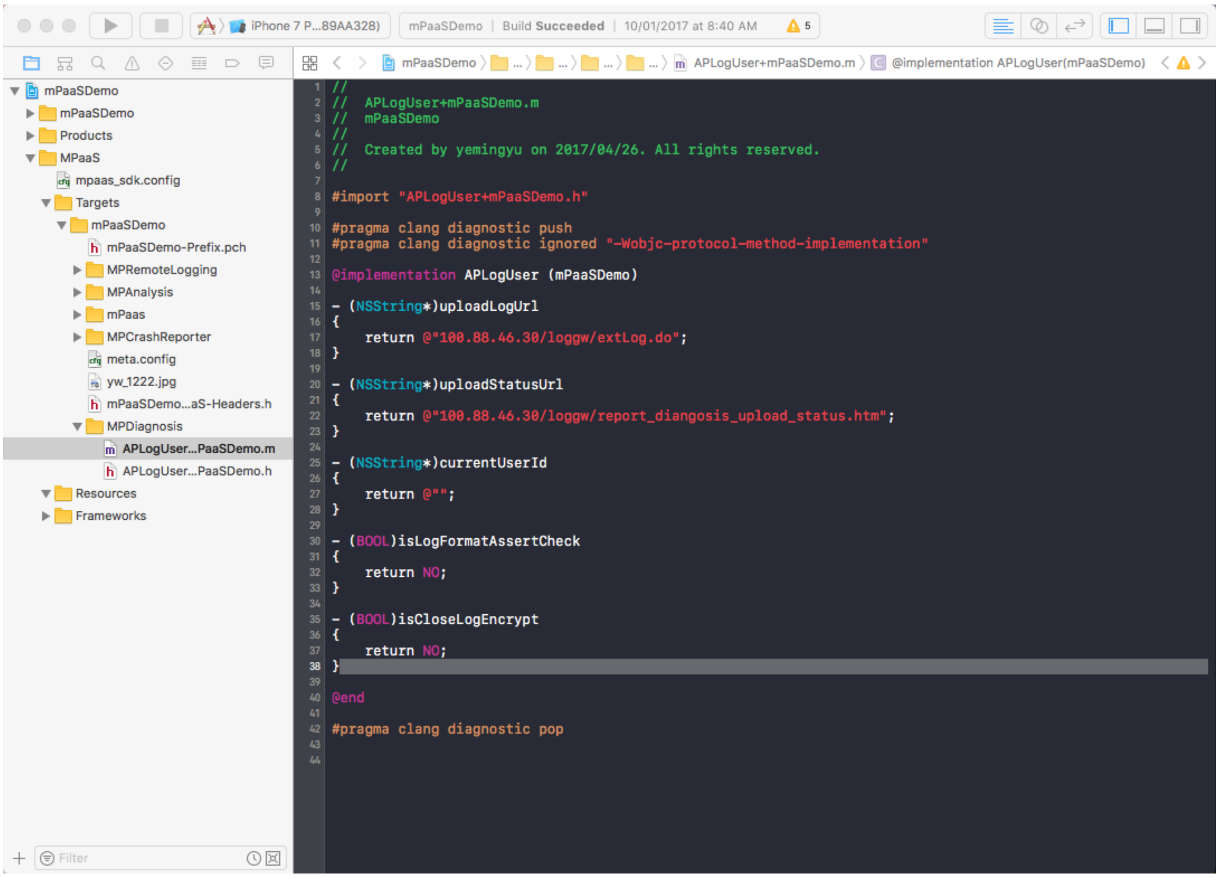
Use Message Push Service (Push)

You need to integrate the Message Push Service, and complete iOS first. After receiving the message pushed, call the following method:

```
#import <APLog/APLogMgr.h>
[[APLogMgr sharedInstance] handlePushDiagnosisCmd:[notification.userInfo
objectForKey:@"content"]];
```

Upgrade precautions

The `Category` file of the `APLogUser` class does not need to be added since version 10.1.32. The middle tier implements packaging. After upgrade, check for configurations of the earlier version in the project and delete them if any. The following shows the configurations that must be removed from the new version.

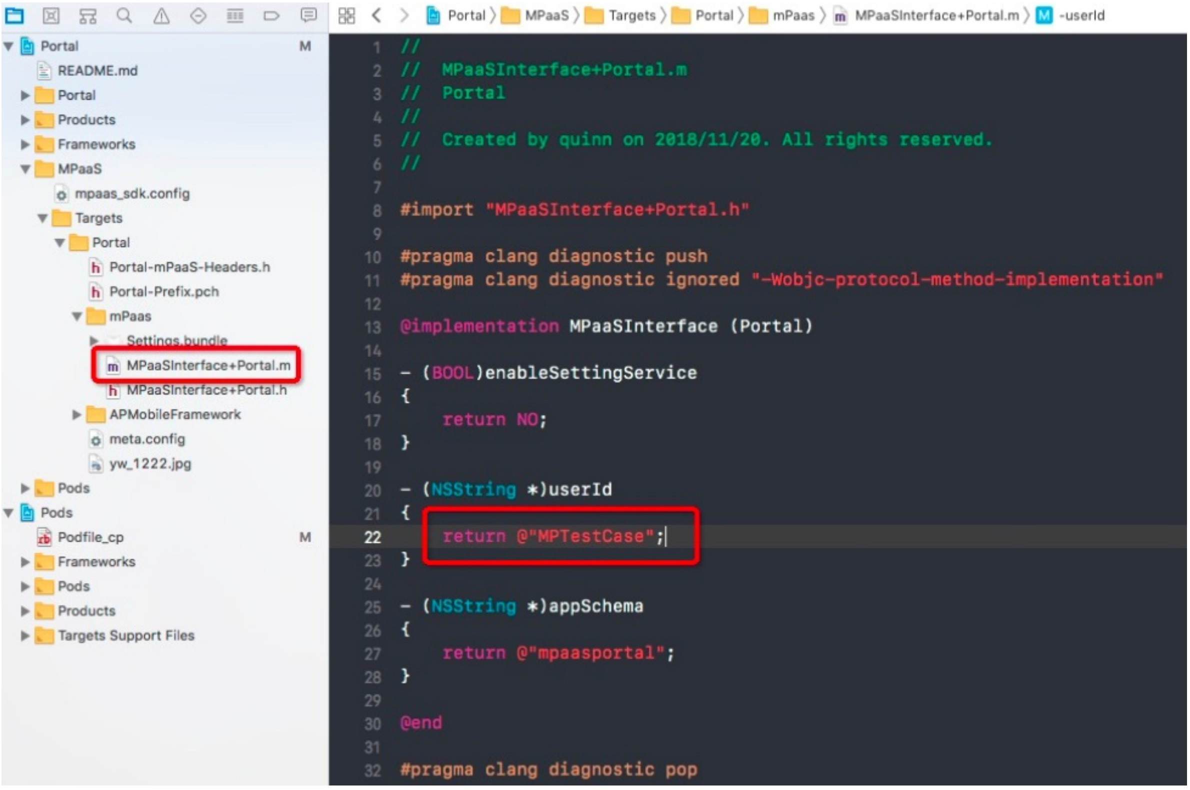


```
//
//  APLogUser+mPaaSDemo.m
//  mPaaSDemo
//
//  Created by yemingyu on 2017/04/26. All rights reserved.
//

#import "APLogUser+mPaaSDemo.h"

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wobjc-protocol-method-implementation"

@implementation APLogUser (mPaaSDemo)

- (NSString*)uploadLogUrl
{
    return @"100.88.46.30/loggw/extLog.do";
}

- (NSString*)uploadStatusUrl
{
    return @"100.88.46.30/loggw/report_diangosis_upload_status.htm";
}

- (NSString*)currentUserId
{
    return @"";
}

- (BOOL)isLogFormatAssertCheck
{
    return NO;
}

- (BOOL)isCloseLogEncrypt
{
    return NO;
}

@end

#pragma clang diagnostic pop
```

Set user ID

The diagnosis service extracts logs based on **user ID**.

1. Set user ID by using the `userId` function during the implementation of `MPaaSInterface`.



```
//
//  MPaaSInterface+Portal.m
//  Portal
//
//  Created by quinn on 2018/11/20. All rights reserved.
//

#import "MPaaSInterface+Portal.h"

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wobjc-protocol-method-implementation"

@implementation MPaaSInterface (Portal)

- (BOOL)enableSettingService
{
    return NO;
}

- (NSString *)userId
{
    return @"MPTestCase";
}

- (NSString *)appSchema
{
    return @"mpaasportal";
}

@end

#pragma clang diagnostic pop
```

2. During user switching, that is, when the value configured in the `userId` function of `MPaaSInterface` changes, call the following function:

   ```
   [MPDiagnoseAdapter userChange];
   ```

   For more information, see the `MPDiagnoseAdapter.h` file under `MPDiagnosis`.

Write diagnosis logs

Call the following method to write diagnosis logs on the critical link of the App:

```
/**
 * Log a message with kAPLogLevelInfo level.
 *
 * @param message   An NSString object that contains a printf-style string containing a
log message and placeholders for the arguments.
 * @param ...      The arguments displayed in the format string.
 */
#define APLogInfo(tag,fmt, ...) \
APLogToFile(tag, kAPLogLevelInfo, fmt, ##__VA_ARGS__)
```

For more information, see the `APLog.h` file under `APLog`.

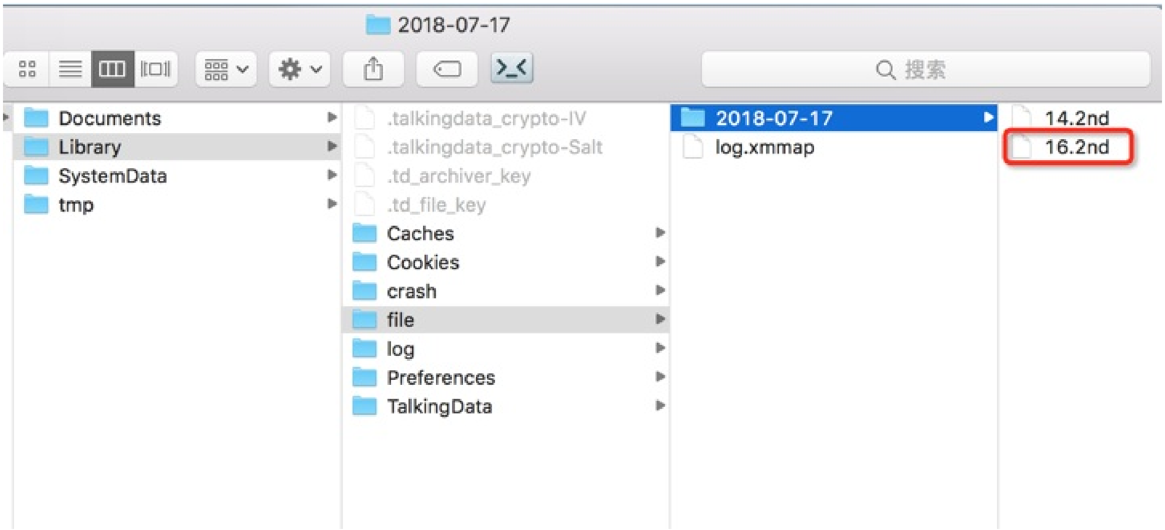
For example, call the following statement to write diagnosis logs after startup:

```
APLogInfo(@"mPaaS", @"Start Cost %d", time);
```

> **Note**
>
> - Logs written by APLogInfo are not printed in the Xcode console. To print related logs in the console during the development phase, add the ConsoleLog file to the project.
> - Out of security concerns, if you do not want your App to print any logs (including logs printed by NSLog and APLogInfo) after going online, you can add the RemoveNSLog file to the project when the app’s release package is generated.

View local diagnosis logs

You can find diagnosis logs in the sandbox directory, as shown in the following figure. This type of logs are not reported by default. The console delivers a command to extract the logs when required.



The diagnosis logs saved locally on the client are limited by the **retention period** and **file size**. The diagnosis SDK on the client follows the following policies when the App runs in the backend or when the process is ended:

- Retention period: 6 days by default. If the SDK detects that the file of logs of the previous 3 days exceeds 30 MB, the logs of the previous 3 days are all deleted.
- File size: not more than 100 MB by default When the file size exceeds 100 MB, earlier logs of a half size are deleted. For example, if the file size is 120 MB, 60 MB logs are deleted.

Obtain online user diagnosis logs

After your App is released and goes online, if you need to obtain the local diagnosis logs on the client for troubleshooting, you can deliver a command in the mPaaS console to obtain related diagnosis logs. For detailed operations, see Mobile Analysis > Extract logs.

9.5. Tracking log model

9.5.1. Log tracking

This topic introduces the log tracking involved in Mobile Analysis Service (MAS).

Data collection instructions

MAS functions rely on the tracking logs reported by the client. To provide more accurate analysis capabilities, tracking logs collect the following device information: public IP address, IMEI, IMSI, device model, operating system version, network type (such as Wi-Fi, 3G, or 4G), operating system language, number of CPU cores, CPU rotation speed, memory size, screen resolution, client channel ID, and client version number.

Tracking log model

The tracking log model varies with the log type. A log is a string of characters separated by commas. Each field in the string has a specific meaning and the server splits a log by field.

Common tracking types are as follows:

- Android or iOS tracking
  - Custom event tracking: records actions such as button clicks or link clicks. You can create tracking for any trigger time of an action in your App. Then, the tracking logs can be used in the functions such as **custom event analysis** and **funnel analysis**.

  - Behavior tracking
    - **Active device/user tracking**: records a start of your App, either cold start or App switches to the foreground. The tracking logs can be used to analyze key metrics such as **Startup count**, **New users**, **Active users**, and **Active accounts**.
      - In Android client, by default, one active device/user is reported every time the App switches to foreground after staying at background for over 30 minutes.
      - In iOS client, by default, one active device/user is reported every time the App switches from background to foreground. If you want to adjust the reporting interval as 30 minutes, you can set the return value of `[[DTFrameworkInterface sharedInstance] logReportActiveMinInterval];` as 1800.
    - **Page automation tracking**: records information such as page views, page traffic, and dwell duration. The tracking logs can be used to analyze metrics such as **PV**, **UV**, and **Page traffic**.
    - **Switch-to-background tracking**: records information about the switch between the foreground and background of your App. The tracking logs can be used to analyze metrics such as **App usage time** and **Active duration**.
  - Performance tracking
    - **Startup speed tracking**: records the **startup speed** of your App, including the first startup (start an App for the first time after the App is installed) and non-first startup (App startup other than the first startup after App installation).
    - **Stuck tracking**: records App stuck and related error logs. The stuck includes:
      - Android startup stuck: Failure to leave the welcome page and enter the home page within 30 seconds after the App is launched.
      - Android ANR stuck: The App does not respond. For more information, see the ANR description on the Android website.
      - iOS startup stuck: The main thread does not finish executing a method within **5s** when the App starts.
      - iOS ANR stuck: The main thread does not finish executing a method within **5s** when the App is running.
    - **Lag tracking**: records App lags and related error logs. A lag occurs when the main thread does not finish executing a method within the specified period (2.25 seconds for Android and 2 seconds for iOS).
    - **Crash tracking**: records App crashes and error stacks.
- H5 and PC tracking
  - Page tracking: automatically records information such as page views and traffic flow. The information is used to measure indicators such as **PV**, **UV**, and **Traffic flow direction**.
  - Tap tracking: records the click on a specific button or link on a Web page.
  - Exposure tracking: records exposure information about certain content on a Web page.

9.5.2. Android/iOS custom event tracking

This topic introduces the client-side and server-side log models for custom event tracking respectively.

Note

If the description of a field is “-“, it means the field is not used and you can ignore it.

Client-side log model

Note

The client-side and backend share a log model.

No.	Example	Field description
00	D-VM	The log header, which is fixed to D-VM.
01	2018-12-19 10:35:47.196	The client log time.
02	A1111111****1_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	V1.0.1	The client version, namely, the App version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	Format: IMSI|IMEI
06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
07	user****	The ID generated after a user gets registered with client C, namely, the user ID.
08	event	Fixed to event
09	-	-
10	first	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is - or first.
11	-	-
12	-	-

13	-	-
14	-	-
15	PayResults	Custom event ID.
16	-	-
17	Pay	The custom business code that is used to manage log upload policies.
18	-	-
19	-	-
20	-	-
21	-	-
22	time=2018-07-27^amount=10.05	Used to store custom attributes and attribute values in the key=value^key=value... format.
23	-	-
24	-	-
25	Wn1111111111111111111****	The UTDID, namely, the device ID.
26	-	-
27	-	-
28	-	-
29	-	-
30	-	-

31	-	-
32	-	-
33	vivo Xplay3S	Device model.
34	4.4.2	OS version.
35	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
36	-	-
37	alipay	Channel number.
38	en-US	Operating system language.
39	0	Version number of the hotpatch package.
40	4	Number of CPU cores.
41	2265	The maximum CPU speed, unit: MHz.
42	2853	The memory size, unit: MB.
43	-	-
44	-	-
45	2560 × 1440	Screen resolution.
46	-	-
47	-	-

Server-side log model

No.	Example	Field description

01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId.
05	V1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****|81111111111****	Format: IMSI|IMEI.
08	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID.
09	user****	The ID generated after a user gets registered with client C, namely, the user ID.
10	event	Fixed to event.
11	-	-
12	first	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is - or first.
13	-	-
14	-	-
15	-	-
16	-	-
17	PayResults	Custom event ID.

18	-	-
19	Pay	The custom business code that is used to manage log upload policies.
20	-	-
21	-	-
22	-	-
23	-	-
24	time=2018-07-27^amount=10.05	Used to store custom attributes and attribute values in the key=value^key=value... format.
25	-	-
26	-	-
27	Wn111111111111111111****	The UTDID, namely, the device ID.
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	-	-
34	-	-
35	vivo Xplay3S	Device model.

36	4.4.2	OS version.
37	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
38	-	-
39	alipay	Channel number.
40	en-US	Operating system language.
41	0	Version number of the hotpatch package.
42	4	Number of CPU cores.
43	2265	The maximum CPU speed, unit: MHz.
44	2853	The memory size, unit: MB.
45	-	-
46	-	-
47	2560 × 1440	Screen resolution.
48	-	-
49	-	-

9.5.3. Android/iOS behavior tracking

This topic introduces the client-side and server-side log models for behavior tracking respectively.

Behavior tracking includes:

- Active device/user tracking
- Page automation tracking
- Switch-to-background tracking

> **Note**
>
> If the description of a field is "-", it means the field is not used and you can ignore it.

Client-side log model

> **Note**
>
> The client-side and backend share a log model.

Active device/user tracking

No.	Example	Field description
00	D-VM	The log header, which is fixed to D-VM.
01	2018-12-19 10:35:47.196	The client log time.
02	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	1.0.1	The client version, namely, the App version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	In the format of IMSI|IMEI, only available in Android devices. **Note** • In Android 10 and later versions, the system doesn't acquire IMSI and IMEI. • In Android 9 and earlier versions, IMSI and IMEI will be obtained on the premise that the system has been authorized with READ_PHONE_STATE permission and the user agrees to the privacy agreement. • In case of not acquiring IMSI and IMEI, the timestamp generated upon the first launch after installation is used instead.
06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID

07	user****	The ID generated after a user gets registered with App, namely, the user ID.
08	event	Used to identify the behavior type, which is fixed to event.
09	-	-
10	first	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is "-" or "first”.
11	-	-
12	-	-
13	-	-
14	-	-
15	reportActive	Fixed to reportActive.
16	-	-
17	-	-
18	-	-
19	-	-
20	-	-
21	-	-
22	-	-
23	-	-
24	-	-

25	Wn1111111111111****	UTDID, namely, the device ID.
26	-	-
27	-	-
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	vivo Xplay3S	Device model.
34	4.4.2	OS version.
35	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
36	-	-
37	alipay	Channel number.
38	en-US	Operating system language.
39	0	Version number of the hotpatch package.
40	4	Number of CPU cores.
41	2265	The maximum CPU frequency, unit: MHz.
42	2853	The memory size, unit: MB.
43	-	-

44	-	-
45	2560 × 1440	Screen resolution.
46	-	-
47	-	-

Page automation tracking

No.	Example	Field description
00	D-VM	The log header, which is fixed to D-VM.
01	2018-12-19 10:35:47.196	The client log time.
02	A111111****1_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	V1.0.1	The client version, namely, the App version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	Format: IMSI|IMEI
06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
07	user****	The ID generated after a user gets registered with App, namely, the user ID.
08	auto_openPage	Used to identify the type of behavior, which is fixed to auto_openPage.
09	-	-
10	Referer_Page_ID|Referer_Page_Position	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is "-" or "first”.

11	-	-
12	-	-
13	-	-
14	-	-
15	Current_Page_ID	Current page ID.
16	-	-
17	-	-
18	-	-
19	-	-
20	-	-
21	-	-
22	-	-
23	-	-
24	9180	The time of staying on a page, unit: ms.
25	Wn11111111111111****	UTDID, namely, the device ID.
26	-	-
27	Current_Page_Position	Current page location ID.
28	-	-
29	-	-

30	-	-
31	-	-
32	-	-
33	vivo Xplay3S	Device model.
34	4.4.2	OS version.
35	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
36	-	-
37	alipay	Channel number.
38	en-US	Operating system language.
39	0	Version number of the hotpatch package.
40	4	Number of CPU cores.
41	2265	The maximum CPU frequency, unit: MHz.
42	2853	The memory size, unit: MB.
43	-	-
44	-	-
45	2560 × 1440	Screen resolution
46	-	-
47	-	-

Switch-to-background tracking

No.	Example	Field description
00	D-VM	The log header, which is fixed to D-VM.
01	2018-12-19 10:35:47.196	The client log time.
02	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	1.0.1	The client version, namely, the App version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	Format: IMSI|IMEI.
06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID.
07	user****	The ID generated after a user gets registered with App, namely, the user ID.
08	auto_event	Used to identify the behavior type, which is fixed to auto_event.
09	-	-
10	first	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is "-" or "first”.
11	-	-
12	-	-
13	-	-
14	-	-

15	-	-
16	-	-
17	-	-
18	-	-
19	-	-
20	-	-
21	leavehint	Fixed to leavehint
22	uid=1****^pid=1****^stayTime=266083	Format: key = value ^ key = value... stayTime indicates the time of staying on a page, in ms.
23	-	-
24	-	-
25	Wn1111111111****1111QxL	UTDID, namely, the device ID.
26	-	-
27	-	-
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-

33	vivo Xplay3S	Device model.
34	4.4.2	OS version.
35	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
36	-	-
37	alipay	Channel number.
38	en-US	Operating system language.
39	0	Version number of the hotpatch package.
40	4	Number of CPU cores.
41	2265	The maximum CPU frequency, unit: MHz.
42	2853	The memory size, unit: MB.
43	-	-
44	-	-
45	2560 × 1440	Screen resolution.
46	-	-
47	-	-

Server-side log model

Active device/user tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.

02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId.
05	1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****|81111111111****	In the format of IMSI|IMEI, only available in Android devices. **Note** • In Android 10 and later versions, the system doesn't acquire IMSI and IMEI. • In Android 9 and earlier versions, IMSI and IMEI will be obtained on the premise that the system has been authorized with READ_PHONE_STATE permission and the user agrees to the privacy agreement. • In case of not acquiring IMSI and IMEI, the timestamp generated upon the first launch after installation is used instead.
08	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID.
09	user****	The ID generated after a user gets registered with App, namely, the user ID.
10	event	Used to identify the behavior type, which is fixed to event.
11	-	-

12	first	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is "-" or "first”.
13	-	-
14	-	-
15	-	-
16	-	-
17	reportActive	Fixed to reportActive.
18	-	-
19	-	-
20	-	-
21	-	-
22	-	-
23	-	-
24	-	-
25	-	-
26	-	-
27	Wn11111111111****	UTDID, namely, the device ID.
28	-	-
29	-	-

30	-	-
31	-	-
32	-	-
33	-	-
34	-	-
35	vivo Xplay3S	Device model.
36	4.4.2	OS version.
37	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
38	-	-
39	alipay	Channel number.
40	en-US	Operating system language.
41	0	Version number of the hotpatch package.
42	4	Number of CPU cores.
43	2265	The maximum CPU frequency, unit: MHz.
44	2853	The memory size, unit: MB.
45	-	-
46	-	-
47	2560 × 1440	Screen resolution.
48	-	-

49	-	-

Page automation tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId.
05	V1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****|81111111111****	Format: IMSI|IMEI.
08	d5557b75-ff80-4aab-86a6-9b1a522bbb****	Session ID.
09	user11	The ID generated after a user gets registered with App, namely, the user ID.
10	auto_openPage	Used to identify the type of behavior, which is fixed to auto_openPage.
11	-	-
12	Referer_Page_ID|Referer_Page_Position	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is "-" or "first”.

13	-	-
14	-	-
15	-	-
16	-	-
17	Current_Page_ID	Current page ID.
18	-	-
19	-	-
20	-	-
21	-	-
22	-	-
23	-	-
24	-	-
25	-	-
26	9180	The time of staying on a page, unit: ms.
27	Wn111111111111****	UTDID, namely, the device ID.
28	-	-
29	Current_Page_Position	Current page location ID.
30	-	-
31	-	-

32	-	-
33	-	-
34	-	-
35	vivo Xplay3S	Device model.
36	4.4.2	OS version.
37	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
38	-	-
39	alipay	Channel number.
40	en-US	Operating system language.
41	0	Version number of the hotpatch package.
42	4	Number of CPU cores.
43	2265	The maximum CPU frequency, unit: MHz.
44	2853	The memory size, unit: MB.
45	-	-
46	-	-
47	2560 × 1440	Screen resolution.
48	-	-
49	-	-

Switch-to-background tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
05	1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****1|81111111111****	Format: IMSI|IMEI.
08	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID.
09	user****	The ID generated after a user gets registered with App, namely, the user ID.
10	auto_event	Used to identify the behavior type, which is fixed to auto_event.
11	-	-
12	first	Format: Upper-level page ID|Upper-level page location ID. If the current page is the home page, this field is "-" or "first”.
13	-	-
14	-	-

15	-	-
16	-	-
17	-	-
18	-	-
19	-	-
20	-	-
21	-	-
22	-	-
23	leavehint	Fixed to leavehint.
24	uid=10206^pid=16992^stayTime=266083	Format: key = value ^ key = value... stayTime indicates the time of staying on a page, in ms.
25	-	-
26	-	-
27	Wn1111111111****	UTDID, namely, the device ID.
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-

33	-	-
34	-	-
35	vivo Xplay3S	Device model.
36	4.4.2	OS version.
37	WIFI	Network type, such as Wi-Fi, 2G, 3G, or 4G.
38	-	-
39	alipay	Channel number.
40	en-US	Operating system language.
41	0	Version number of the hotpatch package.
42	4	Number of CPU cores.
43	2265	The maximum CPU frequency, unit: MHz.
44	2853	The memory size, unit: MB.
45	-	-
46	-	-
47	2560 × 1440	Screen resolution.
48	-	-
49	-	-

9.5.4. Android/iOS performance tracking

This topic introduces the client-side and server-side log models for performance tracking respectively.

Performance tracking includes:

- Startup tracking
- Crash tracking
- Lag tracking
- Stuck tracking

> **Note**
>
> - If the description of a field is "-", it means the field is not used and you can ignore it.
> - If the device ID is left empty or "-", the device will not be counted.

Client-side log model

> **Note**
>
> The client-side and backend share a log model.

Startup tracking

No.	Example	Field description
00	D-MM	Log header, which is fixed to D-MM.
01	2018-12-19 10:35:47.196	The client log time.
02	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	1.0.1	The client version, namely, the App version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	Format: IMSI|IMEI
06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
07	user****	The ID generated after a user gets registered with App, namely, the user ID.

08	-	-
09	-	-
10	-	-
11	performance	Fixed to performance
12	time_startup	Fixed to time_startup
13	3785	The startup time, unit: ms.
14	1	Whether it is initial startup: 0: Yes 1: No
15	-	-
16	-	-
17	android	Operating system
18	4.4.2	OS version
19	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
20	vivo Xplay3S	Device model
21	-	-
22	alipay	Channel number
23	Wn1111111111111****	UTDID, namely, the device ID.
24	en-US	Operating system language
25	4	Number of CPU cores
26	2265	The maximum CPU frequency, unit: MHz.

27	2853	The memory size, unit: MB.
28	0	Version number of the hotpatch package
29	-	-
30	-	-
31	-	-
32	2560 × 1440	Screen resolution
33	-	-
34	-	-
35	-	-
36	-	-

Crash tracking

No.	Example	Field description
00	e	Log header, which is fixed to e.
01	2018-12-19 10:35:47.196	The client log time.
02	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	1.0.1	The client version, namely, the App version.
04	-	-
05	41111111111****|81111111111****	Format: IMSI|IMEI

06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
07	user****	The ID generated after a user gets registered with App, namely, the user ID.
08	exception	Fixed to exception
09	-	-
10	-	-
11	-	-
12	-	-
13	MonitorPoint_Crash	Fixed to MonitorPoint_Crash
14	java.lang.RuntimeException:xx	Abnormal stack
15	-	-
16	alipay	Channel number
17	-	-
18	-	-
19	-	-
20	-	-
21	-	-
22	-	-
23	vivo Xplay3S	Device model

24	4.4.2	OS version
25	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
26	-	-
27	-	-
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	-	-
34	-	-
35	-	-
36	-	-
37	-	-

Lag tracking

No.	Example	Field description
00	D-EM	Log header, which is fixed to D-EM.
01	2018-12-19 10:35:47.196	The client log time.
02	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId

03	1.0.1	The client version, namely, the App version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	Format: IMSI|IMEI
06	user****	The ID generated after a user gets registered with App, namely, the user ID.
07	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
08	vivo Xplay3S	Device model
09	7.1.2	OS version
10	-	-
11	alipay	Channel number
12	-	-
13	-	-
14	performance	Fixed to performance
15	lag	Fixed to lag
16	stackFrame=xx	Specific error stack
17	-	-
18	-	-
19	-	-
20	-	-

21	-	-
22	-	-
23	Wn1111111111111****xL	UTDID, namely, the device ID.
24	-	-
25	-	-
26	-	-
27	-	-

Stuck tracking

No.	Example	Field description
00	D-MM	Log header, which is fixed to D-MM.
01	2018-12-19 10:35:47.196	The client log time.
02	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
03	V1.0.1	The client version, namely, the app version.
04	2	The version of the log tracking SDK, which is fixed to 2.
05	41111111111****|81111111111****	Format: IMSI|IMEI
06	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
07	user****	The ID generated after a user gets registered with App, namely, the user ID.

08	-	-
09	-	-
10	-	-
11	keybiztrace	Fixed to keybiztrace
12	BizCanNotUse	Fixed to BizCanNotUse
13	BIZ_APM	BIZ_FRAME indicates startup stuck and BIZ_APM indicates ANR stuck.
14	APM_ANR	FRAME_CLIENT_STARTUP_DEAD indicates startup stuck, and APM_ANR indicates ANR stuck.
15	1000	1111 indicates startup stuck, 1114 indicates iOS ANR stuck, and 1000 indicates Android ANR stuck.
16	historyStacks=xxx	Specific error stack
17	android	Operating system
18	4.4.2	OS version in the case of ANR stuck
19	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
20	vivo Xplay3S	Device model in the case of ANR stuck
21	-	-
22	alipay	Channel number
23	Wn1111111111111111****xL	UTDID, namely, the device ID.
24	en-US	Operating system language

25	4	Number of CPU cores
26	2265	The maximum CPU frequency, unit: MHz.
27	2853	The memory size, unit: MB.
28	0	Version number of the hotpatch package
29	-	-
30	-	-
31	-	-
32	2560 × 1440	Screen resolution
33	-	-
34	-	-
35	-	-
36	-	-

Server-side log model

Startup tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.

04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
05	1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****|81111111111****	Format: IMSI|IMEI
08	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
09	user****	The ID generated after a user gets registered with App, namely, the user ID.
10	-	-
11	-	-
12	-	-
13	performance	Fixed to performance
14	time_startup	Fixed to time_startup
15	3785	The startup time, unit: ms.
16	1	Whether it is initial startup: 0: Yes 1: No
17	-	-
18	-	-
19	android	Operating system
20	4.4.2	OS version

21	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
22	vivo Xplay3S	Device model
23	-	-
24	alipay	Channel number
25	Wn1111111111111111****xL	UTDID, namely, the device ID.
26	en-US	Operating system language
27	4	Number of CPU cores
28	2265	The maximum CPU frequency, unit: MHz.
29	2853	The memory size, unit: MB.
30	0	Version number of the hotpatch package
31	-	-
32	-	-
33	-	-
34	2560 × 1440	Screen resolution
35	-	-
36	-	-
37	-	-
38	-	-

Crash tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
05	1.0.1	The client version, namely, the App version.
06	-	-
07	41111111111****|81111111111****	Format: IMSI|IMEI
08	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
09	user****	The ID generated after a user gets registered with App, namely, the user ID.
10	exception	Fixed to exception
11	-	-
12	-	-
13	-	-
14	-	-
15	MonitorPoint_Crash	Fixed to MonitorPoint_Crash

16	java.lang.RuntimeException:xx	Abnormal stack
17	-	-
18	alipay	Channel number
19	-	-
20	-	-
21	-	-
22	-	-
23	-	-
24	-	-
25	vivo Xplay3S	Device model
26	4.4.2	OS version
27	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	-	-
34	-	-

35	-	-
36	-	-
37	-	-
38	-	-
39	-	-

Lag tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
05	1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****|81111111111****	Format: IMSI|IMEI
08	user****	The ID generated after a user gets registered with App, namely, the user ID.

09	WIFI	Network type, such as WiFi, 2G, 3G, or 4G
10	vivo Xplay3S	Device model
11	7.1.2	OS version
12	-	-
13	alipay	Channel number
14	-	-
15	-	-
16	performance	Fixed to performance
17	lag	Fixed to lag
18	stackFrame=xx	Specific error stack
19	-	-
20	-	-
21	-	-
22	-	-
23	-	-
24	-	-
25	-	-
26	-	-
27	Wn1111111111111111****xL	UTDID, namely, the device ID.

28	-	-
29	-	-

Stuck tracking

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Includes server request IP address, country, province, and city.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId.
05	V1.0.1	The client version, namely, the app version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	41111111111****|81111111111****	Format: IMSI|IMEI
08	d5557b75-ff80-4aab-86a6-9b1a522b****	Session ID
09	user****	The ID generated after a user gets registered with App, namely, the user ID.
10	-	-
11	-	-
12	-	-

13	keybiztrace	Fixed to keybiztrace
14	BizCanNotUse	Fixed to BizCanNotUse
15	BIZ_APM	BIZ_FRAME indicates startup stuck and BIZ_APM indicates ANR stuck.
16	APM_ANR	FRAME_CLIENT_STARTUP_DEAD indicates startup stuck, and APM_ANR indicates ANR stuck.
17	1000	1111 indicates startup stuck, 1114 indicates iOS ANR stuck, and 1000 indicates Android ANR stuck.
18	historyStacks=xxx	Specific error stack.
19	android	Operating system.
20	4.4.2	OS version in the case of ANR stuck.
21	WIFI	Network type, such as WiFi, 2G, 3G, or 4G.
22	vivo Xplay3S	Device model in the case of ANR stuck.
23	-	-
24	alipay	Channel number.
25	Wn1111111111111111****xL	UTDID, namely, the device ID.
26	en-US	Operating system language.
27	4	Number of CPU cores.
28	2265	The maximum CPU frequency, unit: MHz.
29	2853	The memory size, unit: MB.

30	0	Version number of the hotpatch package
31	-	-
32	-	-
33	-	-
34	2560 × 1440	Screen resolution.
35	-	-
36	-	-
37	-	-
38	-	-

9.5.5. HTML5/PC page tracking

The HTML5/PC page tracking model is as follows.

> **Note**
>
> If the description of a field is "-", it means the field is not used and you can ignore it. The log header reported by the client is "D-VM", which will be processed and then removed on server.

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time. This field is filled by the server and absent in client logs.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Include server request IP address, country, province, and city. This field is filled by the server and absent in client logs.
03	2018-12-19 10:35:47.196	The client log time.

04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
05	1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	-	-
08	9ac4990a-7d4d-45fc-9f1d-b0963d9f****	Session ID, generated based on the session. For the client without integrating mPaaS HTML5 Container, the session ID is generated based on UUID and stored in `sessionStorage` .
09	userId	User ID entered manually. See Configure common tracking.
10	auto_openPage	Fixed to auto_openPage
11	-	-
12	file:///Users/work/mtracker/mtrackerRefer.html|-	The upper-level page ID. If no page information is available, it defaults to "-".
13	-	-
14	-	-
15	-	-
16	-	-
17	file:///Users/work/mtracker/mtrackerDemo.htm	The URL of current page
18	-	-

19	UserBehaviorH5	Business code, fixed to `UserBehaviorH5`
20	-	-
21	-	-
22	-	-
23	-	-
24	userAgent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_0) AppleWebKit/537.36 (KHTML%2C like Gecko) Chrome/81.0.xx.xxSafari/537.36^fullURL=file:///Users/work/mtracker/mtrackerDemo.htm^mBizScenario=mPaaS	Format: key=value^key=value... Thereinto, • userAgent: device model or browser • mBizScenario: `bizScenario` (channel source) specified during tracking. See Configure common tracking. • fullURL: page URL
25	-	-
26	-	-
27	267aa54d-a1f3-472e-a36f-e188f4732f42-154356150****	Device ID, generated based on `localStorage`
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	-	-

34	-	-
35	Mac OS X	Device model
36	10_14_2	OS version
37	-	-
38	-	-
39	-	-
40	en-US	Operating system language
41	-	-
42	-	-
43	-	-
44	-	-
45	-	-
46	-	-
47	320 x 568	Screen resolution
48	-	-
49	-	-

9.5.6. HTML5/PC click tracking

The HTML5/PC click tracking model is as follows.

Note

If the description of a field is "-", it means the field is not used and you can ignore it. The log header reported by the client is "D-VM", which will be processed and then removed on server.

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time. This field is filled by the server and absent in client logs.
02	ip=182.11.xx.xx^country=China^province=Beijing^city=Beijing^district=Chaoyang District	The field includes server request IP address, country, province, and city. This field is filled by the server and absent in client logs.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId.
05	1.0.1	Client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	-	-
08	9ac4990a-7d4d-45fc-9f1d-b0963d9****	Session ID, generated based on the session.
09	userId	User ID entered manually. See HTML5 general tracking types.
10	clicked	Fixed to clicked.
11	-	-
12	-	-
13	-	-

14	-	-
15	-	-
16	-	-
17	h5testSeed	`seedName` for button clicks. See HTML5 common tracking types.
18	-	-
19	UserBehaviorH5	Business code, it is `UserBehaviorH5` by default.
20	-	-
21	-	-
22	-	-
23	-	-
24	userAgent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_0) AppleWebKit/537.36 (KHTML%2C like Gecko) Chrome/81.0.xx.xx Safari/537.36^fullURL=file:///Users/work/mtracker/mtrackerDemo.htm^mBizScenario=mPaaS	Format: key=value^key=value...Thereinto, • userAgent: device model or browser. • mBizScenario: `bizScenario` (channel source) specified during tracking. See Configure common tracking. • fullURL: page URL.
25	-	-
26	-	-
27	267aa54d-a1f3-472e-a36f-e188f4732f42-15435615****	Device ID, generated based on `localStorage` .

28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	-	-
34	-	-
35	Mac OS X	Device model.
36	10_14_2	OS version.
37	-	-
38	-	-
39	-	-
40	en-US	Operating system language.
41	-	-
42	-	-
43	-	-
44	-	-
45	-	-
46	-	-

47	320 x 568	Screen resolution.
48	-	-
49	-	-

9.5.7. HTML5/PC exposure tracking

The HTML5/PC exposure tracking model is as follows.

> **Note**
>
> If the description of a field is "-", it means the field is not used and you can ignore it. The log header reported by the client is "D-VM", which will be processed and then removed on server.

No.	Example	Field description
01	2018-12-19 10:35:47.996	The server log time. This field is filled by the server and absent in client logs.
02	ip=182.11.XX.XX^country=China^province=Beijing^city=Beijing^district=Chaoyang District	Include server request IP address, country, province, and city. This field is filled by the server and absent in client logs.
03	2018-12-19 10:35:47.196	The client log time.
04	A11111111****_IOS-default	Format: App ID created in the backend_Platform type-workspaceId
05	1.0.1	The client version, namely, the App version.
06	2	The version of the log tracking SDK, which is fixed to 2.
07	-	-
08	9ac4990a-7d4d-45fc-9f1d-b0963d9f****	Session ID, generated based on the session.

09	userId	User ID entered manually. See HTML5 common tracking types.
10	exposure	Fixed to exposure.
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-
16	-	-
17	h5testSeed	`seedName` for button clicks. See HTML5 common tracking types.
18	-	-
19	UserBehaviorH5	Business code, it is `UserBehaviorH5` by default.
20	-	-
21	-	-
22	-	-
23	-	-

24	userAgent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_0) AppleWebKit/537.36 (KHTML%2C like Gecko) Chrome/81.0.XX.XX Safari/537.36^fullURL=file:///Users/work/mtracker/mtrackerDemo.htm^mBizScenario=mPaaS	Format: `key=value^key=value…` Thereinto, • userAgent: device model or browser. • mBizScenario: `bizScenario` (channel source) specified during tracking. See Configure common tracking. • fullURL: page URL.
25	-	-
26	-	-
27	267aa54d-a1f3-472e-a36f-e188f4732f42-154356150****	Device ID, generated based on localStorage.
28	-	-
29	-	-
30	-	-
31	-	-
32	-	-
33	-	-
34	-	-
35	Mac OS X	Device model.
36	10_14_2	Operating system version.
37	-	-

38	-	-
39	-	-
40	en-US	Operating system language.
41	-	-
42	-	-
43	-	-
44	-	-
45	-	-
46	-	-
47	320 x 568	Screen resolution.
48	-	-
49	-	-

10.H5 common event tracking

10.1. Common tracking types

The Web common automatic tracking solution (mtracker) is a common tracking solution for PC and mobile HTML5 pages. The solution realizes the automatic reporting of logs by setting the tag attribute. This solution is used in mobile HTML5 page to report the event logs of page display, click, and exposure.

Before configuring common tracking, you need to learn about the tracking types used in different business scenarios:

- PV tracking
- Click tracking
- Exposure tracking

PV tracking

After you introduce the mtracker, the system records a PV count in the log every time the page is opened, and you don't have to do any extra operations.

Click tracking

Click tracking records the clicks on certain buttons and links on a page.

Tag-based tracking

- The mtracker monitors click events, and automatically adds log about clicks for the tags with and `data-seed` attribute.

  ```
  <div seed="seedname"></div>
  ```

- If you need `bizType` , you can add the extended attribute prefixed with `data-biztype` which will also be logged together with `seed` attribute.

  ```
  <div data-seed="seedname" data-biztype="xxx"></div>
  ```

- If you need extra click event information, you can add the extended attribute prefixed with `data-mtr-` which will also be logged together with `seed` attribute. The extended attribute actually logged will have `data-mtr-` removed.

  ```
  <div seed="seedname" data-mtr-extra1="111" data-mtr-extra2="222" >
  </div>
  ```

Trigger actively

In some business scenarios, you must manually trigger click events. For example, different clicks on the same tag need to be logged to different points based on judgment. At this time, the JS click tracking method is needed.

```
Tracker.click(eventId, options)
```

- Parameter Description

Parameter	Type	Example	Description
eventId	String	clickseedname	Event ID
options	Object		Options, including `bizType` , `ext` .
bizType	String	Pay	Business type
ext	Object	`{productId: ‘xxx’ }`	Extended parameter

- Code sample

```
Tracker.click('clickseedname', { bizType: 'Pay', ext: { productId: 'xxx' } });
```

Exposure tracking

Exposure tracking records the exposure of certain content on a page.

If the exposure event is required to be manually triggered, such as a carousel display, you need to use the JS exposure event logging method `tracking.expo()` .

```
Tracker.expo(eventId, options)
```

- Parameter Description

Parameter	Type	Example	Description
eventId	String	exposeedname	Event ID
options	Object		Options, including `bizType` , `ext` .
bizType	String	Pay	Business type
ext	Object	`{productId: ‘xxx’ }`	Extended parameter

- Code sample

```
Tracker.expo('exposeedname', { bizType: 'Pay', ext: { productId: 'xxx' } });
```

Related links

- Configure common tracking
- Analyze common tracking

10.2. Configure common tracking

You can use a unified HTML5 tracking solution in PC and mobile HTML5 pages. By configuring tracking, you can realize unified HTML5 tracking.

About this task

You can configure different HTML5 common tracking in different business scenarios. To learn the specific business scenarios and corresponding tracking types, see Common tracking types.

Procedure

1. Introduce the mtracker of CDN version directly. When the mtracker is introduced, `Tracker` object is injected in the global `window`.

   In case of any messy code in the downloaded mtracker JS file, you can still use the JS file normally because that will not affect the tracking configuration. The messy code may be caused by the difference between the file encoding format and the encoding format interpreted by the browser.

2. Initialize configuration. Inject corresponding information in the HTML5 tracking JS file according to different situations:

   - For the Apps integrated with mPaaS HTML5 container

```
<script>
window._to = {
  bizScenario: 'alipay',
  mtrDebug: true,
};
</script>
```

Parameter	Description
bizScenario	Channel source, optional. It is empty by default.
mtrDebug	Whether to enable the debug mode for mtracker. In debug mode, the reported log will be printed, it defaults to "false".

   - For the Apps not integrated with mPaaS HTML5 container or browser-side Apps

```
<script>
window._to = {
  server: 'https://cn-hangzhou-mas-log.cloud.alipay.com/loggw/webLog.do',
  appId: 'xxxxxxxxxx',
  workspaceId: 'default',
  h5version: '1.0.0',
  userId: '1234567890',
  bizScenario: 'alipay',
  mtrDebug: true,
  extendParams: { test: 111 }
};
</script>
```

Parameter	Description
server	The service address that accepts the tracking.
appId	App's unique identifier.
workspaceId	Workspace identifier.
h5version	App version or HTML5 page version, optional.
userId	User ID, optional.
bizScenario	Channel source, optional. It is empty by default.
mtrDebug	Whether to enable the debug mode for mtracker. In debug mode, the reported log will be printed, it defaults to false.
extendParams	Global extended parameters, supported in mtracker 1.2.0 and later versions. It is empty by default.

3. Initialize mtracker objects.

   By default, mtracker is automatically initialized and injected into the `window` object after the JS file is introduced. If manual initialization is required in some scenarios, you should complete the initialization the performing the steps below:

   i. Add the following code in front of the location where the JS file is introduced to prohibit automatic initialization.

   ```
   window.notInitTrackerOnStart = true;
   ```

   ii. Add initialization code.

   ```
   window.initTracker();
   ```

Note

The global extended parameters can be modified since mtracker 1.2.0. If your mtracker version is lower than 1.2.0, please upgrade it first.

Modify the global extension parameters as needed. By setting the `extendParams` in `window._to`, the extended parameters you set can be added in subsequent tracking logs. In case that duplicate attributes exist in the `ext` parameter set in the `click` or `expo` method, the value in the `click` or `expo` method shall prevail.

You can call the following code to change the `extendParams`, and the newly set object will overwrite all the previously set `extendParams` values.

```
window.changeTrackerExtendParams({ newValue: 11111 });
```

What to do next

Log in to the Mobile Analysis Service console, you can analyze the HTML5 common tracking logs reported by mtracker on the custom analysis page. See Analyze common tracking logs for more information.

10.3. Analyze common tracking logs

Analyze the logs reported by mtracker using the custom analysis function of Mobile Analysis Service.

Prerequisite

Make sure that you understand the features of custom analysis. For more information, see Custom analysis.

About this task

When you add a mtracker-related tracking event, the tracking event ID is also the custom event ID. See the parameter `eventId` in Common tracking types.

When you add an extended attribute in mtracker, see the following preset extended attribute introduction for reference.

Attribute ID	Description
userAgent	The `userAgent` of browser
fullURL	The full URL of the current page
mBizScenario	Channel source

Procedure

1. Add a mtracker-related event. For the specific steps, see Configure events.

2. Add the extended attribute in mtracker. Attributes prefixed with `data-mtr-` are custom attributes that can be customized. The prefix `data-mtr-` will be removed from the reported tracking logs. For how to add the extended attributes, see Configure attributes.
3. Analyze the data reported by mTracker by analyzing the events. For the specific steps, see Add event analysis.
4. Display data through the custom dashboard. For the specific steps, see Create custom dashboards.

11.Tutorial

11.1. Custom event analysis

11.1.1. About this tutorial

Before you follow the tutorial, you need to understand the following basic concepts:

Concept	Description
Event	An event records an action performed by a user in the App. You can set up a custom event tracking to log any action (such as button click).
Event ID	An event ID uniquely identifies an event. Events are global in the App. Therefore, event IDs must be **unique** within the same mPaaS App.
Attribute	An event contains some information, for example, user ID, App version, device model, region and language. These information are known as attributes. MAS provides some preset common attributes. You can also customize some attributes applicable to the entire App based on your business requirements, and use them when configuring specific events.
Attribute ID	An attribute ID uniquely identifies an attribute. Attributes are global in the App. Therefore, attribute IDs must be **unique** within the same mPaaS App.
Event analysis	Events and their attributes are stored as logs on the local client and reported to the server. After finishing related configuration and operations in the console, you can view event analysis reports.

Tutorial scenario

When a user finishes a payment, you can record a **payment completed** event. The event contains the **payment time**, **user ID**, and **payment method** attributes.

Event ID and **Attribute ID** will be used during client development and console operations. Therefore, Android and iOS developers and console operators must jointly decide these IDs in advance. The tutorial assumes that the event ID and attribute ID are as follows:

- **Event ID**: `PayResults`
- **Attribute ID**: `Pay_time` , `User_id` , or `Payment_method`

The ID of a custom attribute cannot be duplicate with that of a preset attribute. Preset attributes are visible on the **Mobile Analysis Service** > **Custom analysis** > **Event and attribute configuration** > **Attribute** page in the **mPaaS console**.

This tutorial helps you with **payment completed** event analysis, including:

1. Complete client development: Android & iOS
2. Create attributes
3. Create an event

4. View event PV and UV
5. Add custom analysis
6. Add a dashboard

For any problems, see FAQ.

11.1.2. Android client development

This topic describes how to develop custom event anlaysis function on the Android client, including:

1. Integrate MAS component
2. Record event logs
3. Report logs

Integrate MAS component

Integrate MAS component by referring to Access Android - Quick start guide.

Record event logs

The section guides you how to record event logs through a code sample, and gives introduces the parameters involved in the code sample.

Code sample

In the following code example, the client will record the business ID, event ID, payment time, user ID, and payment method corresponding to the event.

```
import com.mpaas.mas.adapter.api.MPLogger;

import java.util.HashMap;
import java.util.Map;

// Specify the business ID
String bizType = "Pay";
// Event ID
String logId = "PayResults";
// Add attributes
Map<String, String> params = new HashMap<>(4);
// Attribute: payment time. Key corresponds to the attribute ID and Value corresponds t
o the attribute value.
params.put("pay_time", String.valueOf(System.currentTimeMillis()));
// Attribute: user ID
params.put("user_id", "the-userId");
// Attribute: payment mode
params.put("payment_method", "alipay");
// Print logs
MPLogger.event(logId, bizType, params);
```

Parameters description

Parameter	Description

bizType	- `bizType` specifies the **business ID** (also known as **business code** or **business type**), which is the unique identifier of a business. In the code sample, bizType is set to `Pay` , indicating payment business. - `bizType` affects the log file name on the client. The log file naming format is `timestamp_package name-process_bizType` .
logId	`logId` specifies the **event ID**, which is the unique identifier of an event. For more information, see the description of tutorial scenario in About this tutorial.
params	`params` stores attributes associated with events. In `params.put("param-key", "param-value")` : - `param-key` : corresponding to the attribute ID. For more information, see the description of tutorial scenario in About this tutorial. - `Param-value` : corresponding to the attribute value. The attribute value is stored as a character string on the client. In actual analysis, the server supports converting the attribute value into a character, integer, or float value.

Report logs

By default, when logs cached on the client reach a certain number or the program runs in the backend for a certain period of time, local logs are automatically reported to the MAS server. During development testing, you can call the following APIs to forcibly report local logs to the server immediately:

```
import com.mpaas.mas.adapter.api.MPLogger;
MPLogger.uploadAll();
```

Related content

- Integration method introduction
- mPaaS baseline release overview
- FAQ

11.1.3. iOS client development

> **Important**
>
> mPaaS will stop maintaining the 10.1.32 baseline starting June 28, 2020. Please use the 10.1.68 series baseline (discontinued maintenance) or 10.1.60 series baseline (discontinued maintenance) series baselines. You can refer to the Upgrade guide of mPaaS 10.1.68 or the mPaaS 10.1.60 upgrade guide to upgrade the baseline version.

This topic describes how to develop custom event anlaysis funciton on the iOS client, including:

1. Integrate MAS
2. Record event logs

Integrate MAS

Integrate MAS by referring to Instructions on MAS integration procedure.

Record event logs

The section guides you how to record event logs through a code sample of 10.1.68 version.

Code sample

```
#import <MPMasAdapter/MPMasAdapter.h>

// Currently actionId supports only KActionID_Event and you do not need to pay attentio
n to it.
NSString * actionId = KActionID_Event;
// Event ID
NSString * eventId = @"PayResults";
// Add attributes
NSMutableDictionary * extParam = [NSMutableDictionary dictionary];
// Attribute: payment time. Key corresponds to the attribute ID and Value corresponds t
o the attribute value.
[extParam setObject:@"2017-05-01 12:03:16" forKey:@"pay_time"];
// Attribute: user ID
[extParam setObject:@"the-userId" forKey:@"user_id"];
// Attribute: payment mode
[extParam setObject:@"alipay" forKey:@"payment_method"];

// Print logs
[MPRemoteLoggingInterface writeLogWithActionId:actionId eventId:eventId
extParam:extParam];
```

Parameters description

Parameter	Description
eventId	An event ID uniquely identifies an event. For more information, see the description of tutorial scenario in About this tutorial.
extParam	`ExtParam` of the NSDictionary type is used to store attributes associated with events. • Key: corresponding to the attribute ID. For more information, see the description of tutorial scenario in About this tutorial. • Value: corresponding to the attribute value. The attribute value is stored as a character string on the client. In actual analysis, the server supports converting the attribute value into a character, integer, or float value.

Related content

- Access methods
- Release notes

- FAQ

11.1.4. Create attributes

Before analyzing an event, you need to configure the attributes that you want to associate with the event.

Procedure

Log in to the mPaaS console, click **Mobile Analysis Service** > **Custom analysis** > **Custom configurations** > **Attribute** page, and click **Create** to add attribute.

The description of the attribute fields is as follows:

- **Attribute ID**: Unique identifier of an attribute. The attribute ID must be globally unique in the whole App.
- **Attribute name**: Name of the attribute ID, customizable.
- **Data type**: Data type of the attribute value.
- **Unit**: Unit of the current attribute.
- **Display status**: Whether to display the current attribute.

The following three properties need to be created:

Attribute ID	Attribute name	Data type	Unit	Enumerable	Display status
pay_time	Payment time	Character	-	No	On
user_id	User ID	Character	-	No	On
payment_method	Payment method	Character	-	Yes	On

What to do next

Create an event

11.1.5. Create an event

Procedure

After creating event attributes in the console, go to the **Mobile Analysis Service** > **Custom analysis** > **Custom configurations** > **Event** page, and create a **complete payment** event.

- **Event ID**: Unique identifier of the event, for example `PayResults`. For more information, see About this tutorial.
- **Event name**: Enter an event name easy to identify, for example, **complete payment**.
- **Add attribute**: On the adding attributes page, you can select created event attributes. All preset platform attributes are automatically added and no manual operation is required.

What to do next

Add a custom analysis

11.1.6. View event PV and UV

You can view **Occurrences (PV)** and **Users (UV)** on the **Mobile Analysis** > **Custom analysis** > **Event analysis** page. In this tutorial, PV indicates the number of successful payment times and UV indicates the number of users finishing payment.

If no data is presented or the data is not as expected, see FAQ for troubleshooting.

11.1.7. Add a custom analysis

In addition to **PV** and **UV**, you can further analyze the event based on attributes.

For example, the tutorial records the **payment method** attribute, which is an enumerated value. Therefore, you can analyze the event PV corresponding to each payment type and thereby analyze payment method preferences of users.

Complete the following steps:

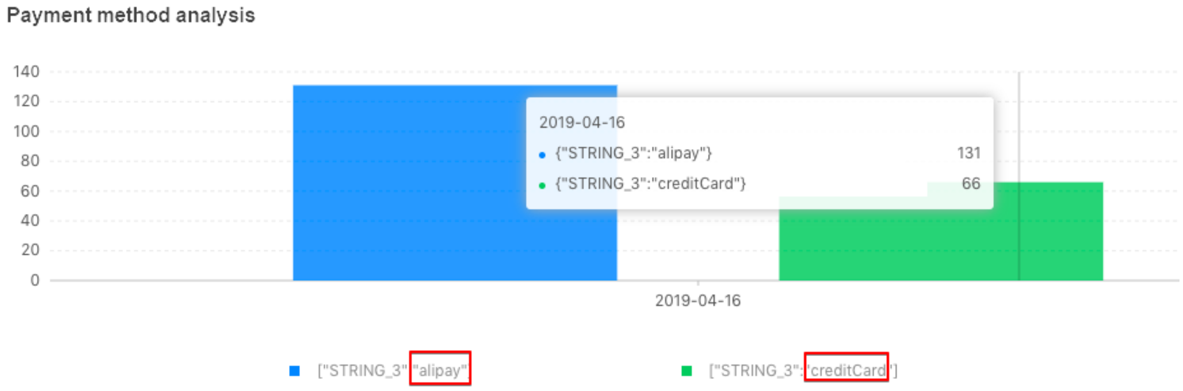
1. On the Event PV and UV page, click the event name to enter the custom analysis page of the corresponding event.

   You can view the default “Dashboard” report created by mPaaS. The report presents the daily PVs and UVs of the event in the line chart.

2. On the Custom Dashboard page, click **Add custom analysis**. Set **Analysis name**, **Displayed indicator**, **Aggregation rule**, and **Display mode**.

3. Click **Submit**. Wait for a moment, and then you can see an analysis report.

   According to the report, the users finish payment using Alipay for 131 times while using credit card for 66 times, suggesting that the users prefer Alipay.



Related content

Add event analysis

11.1.8. Add a dashboard

When there are multiple events, you can create a dashboard in the console to quickly view key business indicators.

The steps are as follows:

1. log in to the **mPaaS console**, select target application, click **Mobile Analysis Service** > **Custom analysis** > **Custom dashboard**.
2. Click **Add a dashboard**. Enter dashboard name, select a dashboard template, and click **OK**.

3. Click the added dashboard.
    - When a dashboard is created, an **Unnamed** category is added by default. Move the pointer to the category name and you can view a rename icon.
    - Click **Add a category** on the right side to add a category.
4. Click **Add custom analysis** and add key indicators report to the dashboard.
    - Add existing analysis: A custom analysis created previously can be directly used here. You can view details of the existing analysis on the **Mobile Analysis Service** > **Custom analysis** > **Card management** page.
    - Create new custom analysis: You can create a custom analysis. For more information, see Add a custom analysis and Funnel analysis.
5. After you add the custom analysis, click **Submit**, the dashboard is created successfully.

11.1.9. Related steps

Now, you have gone through the entire process of implementing a single event analysis.

In practice, a complete process may contain multiple events. For example, the shopping process roughly includes events such as searching for goods, browsing goods, adding goods to shopping cart, placing orders, and conducting payment. At this point, you can perform funnel analysis on the events in the entire business process and analyze the conversion rate of users from the first event to subsequent events. For related operations, see Funnel analysis.

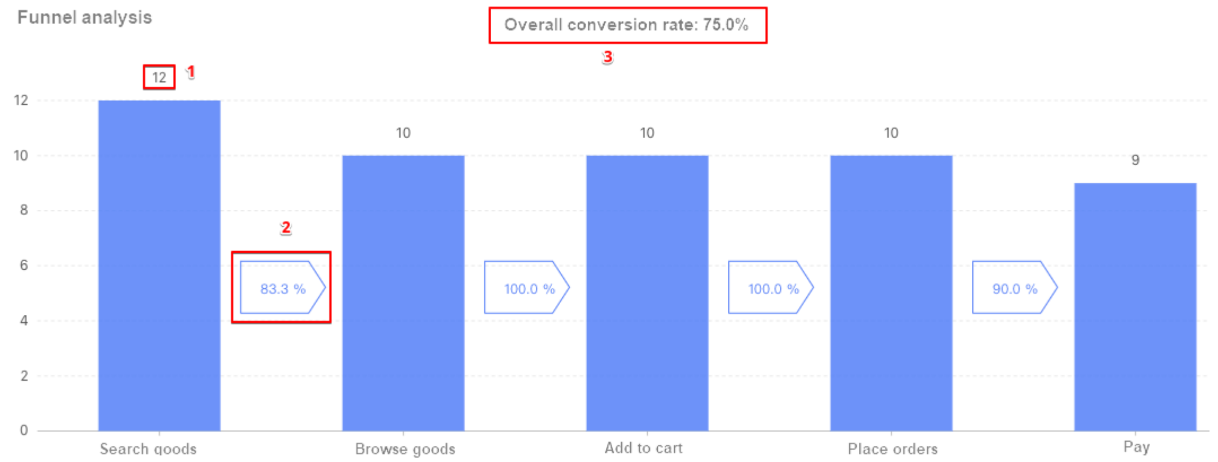
11.2. Funnel analysis

In practice, a complete process usually covers multiple steps and each step can record an event. Funnel analysis helps analyze the conversion rate from the first step to subsequent steps. In the case of a step with a low conversion rate, you can optimize the business logic of this step or perform refined operation by using the Message Push Service, About Mobile Content Delivery Platform, or other products.

To get a deeper insight into funnel analysis, you need to understand the following concepts:

- Conversion rate: Conversion rate from event A to event B under certain conditions = Number of users triggering events A and B concurrently / Number of users triggering event A.
    - The number of users can be measured by **user ID** or **device ID**. You can select either measurement dimension when creating a funnel
    - Certain conditions include: time period and custom filtering conditions.
- Overall conversion rate: Conversion rate from the first event to the last event in a complete process.

The following figure shows the funnel analysis results for a shopping process:



The shopping process covers five steps (events). The figure description is as follows:

1. 12: number of users of the corresponding event (search goods).
2. 83.3%: conversion rate of adjacent events (namely, search goods to browse goods), that is, 10/12 = 83.3%.
3. 75.0%: overall conversion rate, that is, 9/12 = 75.0%.

Prerequisite

A funnel is used to analyze the conversion rate between events. Therefore, you need to customize an event first. For more information, see Create an event.

Procedure

Step 1. Create a funnel

Perform the following steps to create a funnel:

1. In the console, choose **Mobile Analysis Service** > **Custom analysis** > **Funnel analysis**. Click **Create a funnel**.
2. On the **Create a funnel** page, complete related configurations. Fields are described as follows:
    - **Time window**: Required. The maximum time interval for completing two adjacent steps (events) in the funnel.
    - **Dimension**: Select **User ID** or **Device ID**.
3. Add steps (events) based on the actual business process.
4. Click **Submit** to finish the funnel creation.

Step 2. View the funnel

On the **Funnel analysis** page, click the funnel name to view the details of the funnel.

Funnel analysis is T+1 and data is calculated every midnight. You can query the data of yesterday and the day before yesterday.

11.3. Application startup speed analysis

One of the advantages of mPaaS is to help you build super stable, high-performance Apps. This topic guides you to analyze the app startup time.

Android development

1. Integrate Mobile Analysis Service

You can integrate MAS through Native AAR, mPaaS Inside, or Portal & Bundle. However, only Apps based on the mPaaS framework can use APIs encapsulated in the MAS SDK to collect the startup time.

Integrate Mobile Analysis Service by referring to Access Android. The operations include completing the basic configurations, adding the SDK, and configuring the project. It is recommended that you select the SDK of the latest version.

2. Record startup time logs

```
import com.mpaas.mas.adapter.api.MPLogger;

MPLogger.reportLaunchTime(Context context);
```

For more information, see Performance log > Startup time.

iOS development

1. Integrate Mobile Analysis Service

Integrate Mobile Analysis Service by referring to Access iOS.

2. Record the startup time logs

The log API varies according to the SDK version. For more information, see Add performance log.

View the analysis report

Log in to the console, choose **Mobile Analysis Service** > **Basic analysis** > **Behavior analysis**, and select a date. You can view the startup time report.

- First startup: It refers to the first startup of an App after it is installed.
- Non-first startup: It refers to the startup after the first startup.

Only cold startup is involved in the startup time statistics, and the startup time analysis is in **quasi real time**. That is, you can view the reports immediately after Apps are connected to mPaaS and logs are reported to the log server.

11.4. Crash analysis

Crash refers that an app exits unexpectedly. Crash analysis provides crash statistics function, supports statistics on crash count, crash rate and affected devices count, supports aggregation of crash reports by cause, and statistics on the number of crashes of a certain type, number of affected users, main models and other information.

Integrate crash analysis function

- **Android**

  The steps are as follows:

  i. Integrate MAS component. For more information, see Integrate MAS into Android.

  ii. Enable crash monitoring. For more information, see Crash log.

- **iOS**

  The steps are as follows:

   i. Integrate MAS component. For more information, see Integrate MAS into iOS.
   ii. Enable crash monitoring. For more information, see Add crash log.

View crash report

Log in to the mPaaS console, choose **Mobile Analysis Service** > **Performance analysis** > **Crash report** and view the crash report.

In the **Crash classification** list, click the content in the **Details** column to view detailed crash information.

- Device details:
  - **Device ID**: The format is IMSI|IMEI.
  - **Platform**: Consists of three fields and is in the `appId-Platform type ANDROID or iOS-workspaceId` format.
- Log details: Log records are separated by commas and contain complete error stacks. For more information about the specific meaning of each field, see Crash tracking.

Crash analysis is **quasi-real-time**. In other words, you can view the preceding report after the app crashes and a crash log is reported.

11.5. Stuck analysis

Stuck analysis provides statistics on stuck times, stuck rate, and affected devices.

Stuck includes the following conditions:

Stuck type	Android	iOS
Startup stuck	The App fails to leave welcome page and enter home page within 30 seconds after startup.	The `UIApplicationDidFinishLaunchingNotification` notification was not received 15s after the App started (30s on low-end devices).
ANR stuck	System ANR stuck, see ANRs on Android Official Website for definition.	The App's main thread hasn't completed executing any method within 5 seconds at runtime.

Integrate stuck analysis function

- **Android**

  The steps are as follows:

   i. Integrate MAS component. For more information, see Integrate MAS into Android.
   ii. Enable stuck monitoring, see Performance log.

- **iOS**

  The steps are as follows:

   i. Integrate MAS component. For more information, see Integrate MAS into iOS.
   ii. Enable stuck monitoring, see Add crash log.

View stuck report

Log in to the mPaaS console, choose **Mobile Analysis Service** > **Performance analysis** > **Stuck report** and view the **Stuck rate**, **Stuck count**, **Affected devices**, **Stuck report** and other informatoin.

In the **Stuck classification** list, click the content in the **Details** column to view the details of a certain type of stuck.

- Device details:
  - Device ID: The format is IMSI|IMEI.
  - Platform: Consists of 3 fields, which are `appId-Platform-workspaceId`.
- Log details: Log records are separated by comma and contain complete error stacks. For the specific meaning of each field, see Log model > Performance tracking > Stuck tracking.

Stuck analysis is in **quasi-real-time**. In other words, you can view the preceding report after the App stucks and a stuck log is reported.

11.6. Lag analysis

Lag means that the main thread does not complete executing a method within the specified period of time (2.25 seconds for Android and 2 seconds for iOS).

Integrate lag analysis function

- **Android**

  The steps are as follows:

  i. Integrate MAS component. For more information, see Integrate MAS into Android.

  ii. Enable lag monitoring, see Performance log.

- **iOS**

  The steps are as follows:

  i. Integrate MAS component. For more information, see Integrate MAS into iOS.

  ii. Enable lag monitoring, see Add performance log.

View lag report

Log in to the console, choose **Mobile Analysis Service** > **Performance analysis** > **Lag report**. You can view **Device lag times**, **Lag rate**, **Affected devices count**, **Lag report** and other information.

In the **Lag classification** list, click the information in the **Details** column to view detailed lag information.

- Device details:
  - Device ID: The format is IMSI|IMEI.
  - Platform: Consists of 3 fields, which are `appId_Platform-workspaceId`.
- Log details: Log records are separated by comma and contain complete error stacks. For the specific meaning of each field, see Log model > Performance tracking > Lag tracking.

Lag analysis is in **quasi-real-time**. In other words, you can view the lag report immediately after a lag occurs on your app on a device matching the sampling rules and a lag log is reported.

For more information about the sampling rules and the sampling ratio setting method, see the relevant documents when client integration.

11.7. Active device/user analysis

MAS supports recording active logs when the App is started to report that the current user is active.

The active state logging is as follows:

- Active device logging: The current user is uniquely identified by the **device ID**.
- Active user logging: The current user is uniquely identified by the **user ID**.

Integrate active device/user analysis function

- ### Android

  The steps are as follows:

  i. Integrate MAS component. For more information, see Integrate MAS into Android.

  ii. Record active logs, see Add active device/user log.

- ### iOS

  The steps are as follows:

  i. Integrate MAS component. For more information, see Integrate MAS into iOS.

  ii. Record active logs, see Add active device/user log.

View active device/user data

Log in to the mPaaS console, and go to the **Mobile Analysis Service** > **Data overview** page, you can view real-time or historical information such as the **number of application installations** (corresponding to the number of active users), the **number of registered users** (corresponding to the number of active accounts), etc.

On the **Mobile Analysis Service** > **Basic analysis** > **Retention analysis** page, you can view user retention information.

On the **Mobile Analysis Service** > **Basic analysis** > **Behavior analysis** page, you can view user information by region.

Related content

- Indicator calculation rules
- User ID

12.FAQ

The following are some common problems you may encounter when using MAS and how to troubleshoot them.

- How to check whether the client is integrated correctly?
- Why the data is invisible in the console after the client is integrated correctly?
- Why are the event PV and UV invisible after an event is created in the console?
- Why is the event UV always displayed as 0 after an event is created in the console?
- Why no data is displayed in the custom dashboard?
- Why the iOS crash logs in the MAS console are not parsed?

How to check whether the client is integrated correctly?

You can view local logs (Android/iOS) or Query logs in the console to check whether the client is integrated correctly.

Why the data is invisible in the console after the client is integrated correctly?

The client log will be automatically reported to the log server only when certain conditions are met (such as a certain number of local logs or the Application is pushed to the background for more than a certain period of time). During testing, in order to see the data as soon as possible, you can force the log to be reported immediately by manually reporting on the client.

For more information, see Report logs manually - Android or Report logs manually - iOS.

Why are the event PV and UV invisible after an event is created in the console?

To view the event PV and UV, ensure that:

- The client is integrated correctly. For more information, see How to check whether the client is integrated correctly?.
- Logs are reported to the log server. For more information, see Why the data is invisible in the console after the client is integrated correctly?.
- The event created in the console has been triggered.

Why is the event UV always displayed as 0 after an event is created in the console?

Ensure that a user ID is set on the client. For more information, see User ID.

Why no data is displayed in the custom dashboard?

Perform the following steps to troubleshoot the issue:

1. Check whether the log upload switch is turned on. Log in to the mPaaS console, on the left navigation pane, click **Mobile Analysis Service > Log management > Configure upload switch > Event tracking configuration** to go to the log switch list page, and check if the log upload switch is turned on for the specified tracking configuration. If not, just turn the switch on. For more information, see Switch configuration.
2. Check whether the log that corresponds to the dashboard is uploaded to the server. In the console, click **Log management** > **Log playback** to query history logs.
    - If the log that corresponds to the dashboard is found, it means the log has been uploaded. For more information, see Query history logs.

- If the log that corresponds to the dashboard is not found, check whether the App has triggered log generation. Perform the following steps:

  a. Disconnect the mobile from network, and then trigger log generation.

  b. After the App switches to background, go to the local log directory to check whether the queried log exists.

    - iOS clients: The log is stored in the sandbox directory: `Library > atrack > logs` .
    - Android clients: The log is stored in `/data/data/[PackageName]/files/mdap` or `/sdcard/Android/data/[PackageName]/files/mdap` . The logging path varies depending on the `release_type` field in `assets/channel.config` . For more information, see View local logs.

  c. After the client triggered log generation, playback the log again to check whether the log is uploaded to the server.

     After the client generated logs, automatic log uploading is triggered only when the number of locally cached logs reaches the threshold. The threshold may vary depending on the log type. To facilitate debugging, you can go to the **Mobile Analysis Service** > **Log management** > **Configure upload switch** > **Event tracking configuration** page on the mPaaS console, and temporarily change the value of **Upload quantity** (the number of logs of the current type in the local file that triggers log reporting) to 1 to trigger log reporting. After the debugging is completed, change it back to the original number. For more information about triggering of log uploading, see Upload logs from Android client or Upload logs from iOS client.

3. If you find that the logs are still not reported to the server through log playback, refer to Active device/user log or Add activity report log according to the log type to check whether there is any error in the tracking point integration process. After correctly integrating the tracking point according to the document, proceed to step 2 to ensure that the logs are reported normally.
4. If logs are uploaded to the server but no data is displayed on the dashboard, check whether the log data format is correct.

   Compare the original logs in log playback with the tracking log model to check whether the log data format is correct. If the log data format is incorrect, modify the format based on the log models for different types of tracking.
5. After you have performed the preceding steps and ensured that the logs in correct format have been uploaded to the server, but still no data is displayed on the dashboard, please search for the group number 41708565 with DingTalk to join the group for further communication.

Why the iOS crash logs in the MAS console are not parsed?

The iOS client's crash log parsing requires the packaged and generated dSYM symbol table.

The iOS crash statistics feature supports symbolicating crash logs. To use this feature in your App, you must upload the dSYM file on the **Mobile Analysis Service** > **Performance analysis** > **iOS symbol table management** page in the mPaaS console. For more information, refer to iOS symbol table management.

13.Reference

13.1. User ID

User ID is very important in many scenarios.

- Mobile Analysis Service (MAS) presets a string-type attribute named `userId`. You can log in to the console and go to the **Mobile Analysis Service** > **Custom analysis** > **Custom configuration** > **Attribute** page to view the attribute details.
- Different types of tracking logs contain a **User ID** field. For more information, see Log model > Custom event tracking.
- Event analysis related to user volume (UV) depends on the **User ID** field.
- When you use the mPaaS Delivery Service to perform App release to the specific whitelist of users in gray release mode, user IDs are required on the client.

User ID and device ID

A user can use an App before login, which means the **user ID** may be empty, but the **device ID** is usually available. In this case, the device ID can be used for user behavior analysis instead of the user ID.

For example, you can specify the **user ID** or **device ID** for calculation when using the funnel analysis function. When the user ID is used for calculation, the number of users is the number of distinct user IDs. When the device ID is used for calculation, the number of users is the number of distinct device IDs.

Set user ID

To use the user ID related analysis functions, you need to call the SDK API to set user IDs.

> **Note**
>
> - Avoid using special symbols when setting up user IDs. Numbers and letters are recommended.
> - Do not use the device ID read by UTDID as `userId`.

Android

- Case 1: Call the user activation interface `MPLogger.reportUserLogin("userId");`.

  If you want to collect the number of registered users of an App, call `MPLogger.reportUserLogin("userId");`. This API automatically sets the global user ID as the input parameter. No manual operation is required.
- Other cases: Call the `MPLogger.setUserId("userId");` to set a user ID.

For more information, see Active device/user reporting log.

iOS

Set user ID in the `Category` of `MPaaSInterface`. For more information, see Configure a project.

```
@implementation MPaaSInterface (Demo)
- (NSString *)userId
{
    return @"the-user-id";
}
@end
```