Ant Technology

Mobile Delivery Service
User Guide

Document Version: 20231226

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.Service announcement

To provide a highly-efficient real-time delivery service, starting from 21:00 on July 14, 2022 in UTC+8, the domain name used by the Mobile Delivery Service (MDS) for delivery was changed from mcube-prod.cn-hangzhou.oss.aliyuncs.com to mcube-prod.mpaascloud.com. Use the new domain name to obtain released packages.

After this change, to use the acceleration capability of the MDS, you must modify the second-level directory name to mcube-prod.mpaascloud.com.

2.About Mobile Delivery Service

Mobile Delivery Service (MDS for short) is one of the core basic service components of the mPaaS platform. It provides management and release services for version upgrade packages, hotpatch packages, and H5 offline packages, and supports configurations management, whitelists, release rules and other management functions.

Mobile Delivery Service serves to manage and release the version upgrade and hotpatch packages. After you integrate Mobile Delivery Service on your client, you can generate a new package in mPaaS plugin and release the new package on the Mobile Delivery Service console, then the client receives the package and starts upgrade. Mobile Delivery Service also supports implementing gray release by means of whitelist. You can use advanced filtering rule to make the gray release more accurate, e.g. specifying a device type.

Functions

Function	Description
Gray release	Before formal release, launch a small-scale release (for example: only released to internal staff) by means of whitelist to validate if the new package's functions meet the expectation; or implement time-window gray release to release the package to a specified number of users in a specified time period. If it all works, the package can be pushed network-wide.
Advanced filter	When you perform gray release, you can utilize the advanced filtering rule to define a more accurate whitelist. For example, only send the package to the users of MI phones. Multiple filtering rules can be superposed, that means only when all filtering rules are matched can the package be pushed.
Realtime rollback	Only available in Hotpatch. During formal release, problems may arise despite of gray release. At that moment, execute real-time rollback to revert the package to previous version.
Custom signature verification	To ensure security, hotpatch has a custom signature verification flow to ensure the correctness of script source. mPaaS plugin provides the function of generating hotpatch resource packages and signing the packages.

Advantages

- **Delivery management of multiple products, tasks, and dimensions**

  Multiple apps are supported for delivery management. It further provides features such as official release, hotpatch, offline packages, and real-time push.

  For more information about using the hotpatch feature,
  please search for the group number 41708565 with DingTalk to join DingTalk group for further communication.

- **Intelligent grey release, and three upgrade modes available**

  Internal grey release, external grey release, release targeted at specified population, regions, phone types, and networks are available.

- **Push upgraded capabilities of offline packages only**

  It helps reduce data redundancy and bandwidth occupied by devices. It plays a pivotal role when the network on the mobile devices is unstable.

- **High sensitivity and high availability**

  The client-side RPC interface capability is upgraded, with an availability rate up to 99.95%. The target devices can be reached within several minutes.

- **High performance system**

  The reach rate is up to 99.95%, and the daily UV exceeding 0.2 billion can be supported.

3.Process of Mobile Delivery Service

You can conveniently integrate the function of Mobile Delivery Service to your client by installing the client SDK provided by the Mobile Delivery Service platform.
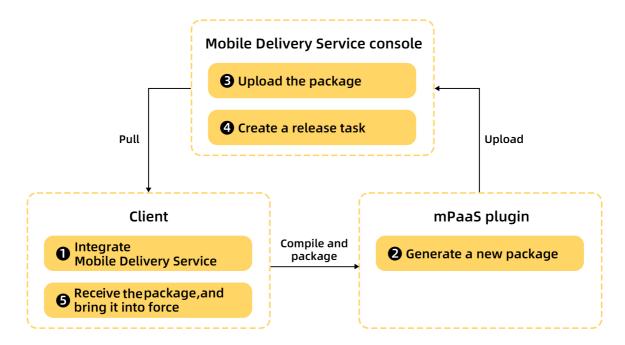
The process of Mobile Delivery Service is as follows:

1. Add the corresponding SDK on the client, the capability of integrating Mobile Delivery Service upgrade or H5 offline packages.

2. Package and generate version upgrade packages, offline packages, etc. in the mPaaS plug-in, and upload them to the release console.

3. Create a release task on the console for gray release, official release, etc.

4. The client then pulls the new release package for upgrade and offline release.

In addition, you can use the switch configuration service to modify the client-side code processing logic. By adding the required switch configuration items on the console, targeted distribution can be achieved.

Operation flow

The following diagram shows the process of Mobile Delivery Service release of version upgrade packages and offline packages:

Operations on console

You can perform the following operations on the Mobile Delivery Service console:

- Version upgrade packages > Manage releases: Manage and release the configuration of new client version.

- Offline packages > Manage offline packages: Package different businesses into different offline packages, and deliver the offline packages through the Mobile Delivery Service platform to update the client-side resources.

- Switch configuration > Manage configurations: Configure, modify or push various switches. You can deliver the packages by platform, whitelist, percentage or other condition.

- Manage whitelists: Manage whitelists so that you can easily create hundreds of thousands of whitelist data for the use in Mobile Delivery Service.

- Manage release rules: Pedefine various configuration data required for Mobile Delivery Service so that you don't have to manually input the data every time, with work efficiency improved and error rate decreased.

4.Manage version upgrade
4.1. Android

4.1.1. Quick start

This topic describes how to add the **Upgrade** SDK related to the release management. After adding the SDK and complete the necessary configurations, you can release a new version of an App is released in the mPaaS console, and the client can detect the new version through the upgrade API and remind users to download and upgrade.

Currently, **Upgrade** SDK supports access through **Native AAR** and **Portal & Bundle**.

The complete access process mainly includes the following 4 steps:

1. Add SDK
2. Configure project
3. Initialize mPaaS(only required for Native AAR)
4. Check for update

Prerequisites

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

Add SDK

Native AAR mode

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

Componentized access mode

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

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

Configure project

Configure AndroidManifest

1. Add the following permissions in the `AndroidManifest.xml` file.

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

2. Add the following configuration in the `AndroidManifest.xml` file:

   ```
   <provider
   android:name="android.support.v4.content.FileProvider"
   android:authorities="${applicationId}.fileprovider"
   android:exported="false"
   android:grantUriPermissions="true">
   <meta-data
    android:name="android.support.FILE_PROVIDER_PATHS"
    android:resource="@xml/file_paths" />
   </provider>
   ```

   > **Note**
   >
   > For more information about the configuration of `AndroidManifest.xml`, please see App Manifest Overview.

3. Create the `file_paths.xml` file in the `src/main/res/xml` directory in the main module of the Portal project with the following content:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
<paths>
 <external-files-path
      name="download"
      path="com.alipay.android.phone.aliupgrade/downloads" />
 <external-path
      name="download_sdcard"
      path="ExtDataTunnel/files/com.alipay.android.phone.aliupgrade/downloads" />
</paths>
</resources>
```

Add resources

> ⑦ **Note**
>
> If you access the UPGRADE SDK through Native AAR, you need to add the following resources to your App, otherwise, the upgrade component will not work. Click here to get the resource file.

Merge the content of `strings.xml` , `styles.xml` , and `colors.xml` under the `values` directory.

Initialize mPaaS

If you access the UPGRADE SDK through Native AAR you must initialize mPaaS.

Add the following codes in the object `Application` :

```java
public class MyApplication extends Application {

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

For more details, see Initialize mPaaS.

Check for update

Quickly check for an update, and only the checking result is returned.

```java
MPUpgrade mMPUpgrade = new MPUpgrade();
//  The synchronization method, which is called in a subthread.
int result = mMPUpgrade.fastCheckHasNewVersion();
if (result == UpgradeConstants.HAS_NEW_VERSION) {
  // New version available
} else if (result == UpgradeConstants.HAS_NO_NEW_VERSION) {
  // No new version available
} else if (result == UpgradeConstants.HAS_SOME_ERROR) {
  // Error
}
```

4.1.2. Advanced operations

After integrating the SDK, you can set whitelist, use the SDK to detect upgrades and notify users of any upgrades based on business requirements.

Set whitelist

Set whitelist user ID:

```java
MPLogger.setUserId("your whitelist ID");
```

Detect new version

- Detect new versions quickly and remind users with a pop-up:

> ⑦ **Note**
>
> Only the upgrade popup is displayed quickly, and the forced upgrade logic is not included. If you need to force an upgrade, please use a custom upgrade to implement it.

```
MPUpgrade mMPUpgrade = new MPUpgrade();
mMPUpgrade.fastCheckNewVersion(activity, drawable);
```

- Detect new versions quickly and return the detection result only:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// The synchronous method, which is called in the subthread.
int result = mMPUpgrade.fastCheckHasNewVersion();
if (result == UpgradeConstants.HAS_NEW_VERSION) {
    // A new version is available.
} else if (result == UpgradeConstants.HAS_NO_NEW_VERSION) {
    // No new version is available.
} else if (result == UpgradeConstants.HAS_SOME_ERROR) {
    // An error occurs.
}
```

Obtain upgrade details

Call the `fastGetClientUpgradeRes` method to obtain more information.

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// The synchronization method, which is called in a subthread.
ClientUpgradeRes clientUpgradeRes = mMPUpgrade.fastGetClientUpgradeRes();
```

The returned example displays information such as the new version number, download address, etc. Some of the parameters have the following meanings:

- `downloadURL` : Download address
- `guideMemo` : Upgrade information
- `newestVersion` : Latest version number
- `resultStatus` : Upgrade mode
  - 202: Single reminder
  - 204: Multiple reminders
  - 203/206: Forced update

Other custom detections

For more information about custom detections, see the following example:

- Implement the `MPaaSCheckCallBack` interface to respond to requests created by the upgrade SDK, such as displaying a pop-up:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
mMPUpgrade.setUpgradeCallback(new MPaaSCheckVersionService.MPaaSCheckCallBack() {
    .........
});
```

- Call the `MPUpgrade.checkNewVersion` method to detect upgrades.

> ⑦ **Note**
>
> `MPUpgrade` encapsulates the call of `MPaaSCheckVersionService` . You can also customize the implementation method. For information about `MPaaSCheckVersionService` and `MPaaSCheckCallBack` , see API reference.

Customize the download directory of installation package (for 10.1.60 and later versions)

The configuration is as follows:

```
File dir = getApplicationContext().getExternalFilesDir("Custom directory");
MPUpgrade mpUpgrade = new MPUpgrade();
mpUpgrade.setDownloadPath(dir.getAbsolutePath());
```

Add the following configurations in the `file_path.xml` file:

```
// external-files-path corresponds to the directory of getExternalFilesDir
// Use the element corresponding to your custom directory. If you are not sure about how to select an element, search for "Adapt FileProvider" on the Internet
<external-files-path
    name="download"
    path="custom directory" />
```

Handle the SDK package parsing failure upon forced upgrade

Some ROMs may fail to parse the SDK package after forced upgrade. This is because the ROMs need to access the corresponding App process when installing the package. However, the App process will be forcibly stopped during forced upgrade. As a result, the package fails to be parsed. Although such custom ROM behavior is incompliant with the native Android platforms, you can still solve the problem by implementing `UpgradeForceExitCallback` to return `false` in `needForceExit` .

1. Implement a callback.

```
public class UpgradeForceExitCallbackImpl implements UpgradeForceExitCallback {
    @Override
    public boolean needForceExit(boolean forceExitApp, MicroApplicationContext context) {
        // If false is returned, the app process will not be forcibly stopped, and the installation package will be parsed successfully. If true is returned, you need to call the doForceExit method below to stop the process.
        return false;
    }
    @Override
    public void doForceExit(boolean forceExitApp, MicroApplicationContext context) {
        // If you need to stop the process, ensure that the needForceExit method above returns true, and then stop the process in this method.
    }
}
```

2. Set the callback.

```
MPUpgrade mpUpgrade = new MPUpgrade();
mpUpgrade.setForceExitCallback(new UpgradeForceExitCallbackImpl());
```

> ⓘ **Important**
>
> Use the same `MPUpgrade` instance to set a callback or request an upgrade.
>
> After setting the callback, you can avoid the failure of parsing the package, but the upgrade component will no longer automatically kill the process for you. Therefore, when the user does not click Install but returns to the application, please set an irrevocable pop-up cover layer by yourself to prevent the user from bypassing the forced upgrade.

4.1.3. Default storage path

Since Baselines V10.2.3.3 and V10.1.68.53, the default path to download a component APK of mPaaS is changed from an external storage path to an internal storage path.

If targetSdkVersion is 24 or later, you must add the following code to the `file_paths.xml` file:

```
<files-path
  name="files-path"
  path="." />
```

If you want to retain the original download path unchanged, add the following metadata to the manifest file:

```
<meta-data android:name="use_external" android:value="yes" />
```

4.2. iOS

4.2.1. Add SDK

This topic describes how to add the **Release upgrade** SDK related to release management. After adding the SDK and complete necessary configurations (refer to Use SDK for details), you can release new versions of an App in the mPaaS console.

- When releasing a version in the mPaaS console, you can customize release settings such as **update reminder** and **release type**.
- After a new app version is released in the mPaaS console, the client can detect the new version through the upgrade API and remind users to download the update.

Note : App Store does not allow online apps to contain the built-in upgrade detection function. In this case, do not release new versions in the mPaaS console when the App is under review.

Prerequisite

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

Add SDK

1. In the **Podfile** file, use `mPaaS_pod "mPaaS_Upgrade"` to add dependency.

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

What to do next

Use SDK

4.2.2. Use SDK

After adding the SDK, perform the following steps to connect the delivery service to the iOS client:

1. Detect new version: Call the SDK API method in the code to check whether new versions are available.

2. Configure whitelist gray release: Configure the functions such as the update reminder and gray release.

3. Online release: Generate an .ipa file in the mPaaS console, and release a new version.

Procedure

Detect new version

The upgrade detection SDK provides an API file to check available App upgrades. The method header file is in the `AliUpgradeCheckService.framework > Headers > MPCheckUpgradeInterface.h` file.

```
typedef NS_ENUM(NSUInteger, AliUpdateType) {
    AliUpgradeNewVersion = 201,            /*The latest version is currently in use.*/
    AliUpgradeOneTime,                     /*A new version of the client is available, and a single remin
der is sent.*/
    AliUpgradeForceUpdate,                 /*A new version of the client is available, and forced upgrade
is implemented (obsoleted).*/
    AliUpgradeEveryTime,                   /*A new version of the client is available, and multiple remin
ders are sent.*/
    AliUpgradeRejectLogin,                 /*Restricted login (obsoleted)*/
    AliUpgradeForceUpdateWithLogin         /*A new version of the client is available, and forced upgrade
is implemented.*/
};


/**
 The successful callback of upgrade detection during UI customization

 @param upgradeInfos The upgrade information
 @{upgradeType:202,
 downloadURL:@"itunes://downLoader.xxxcom/xxx",
 message:@"New version available, please upgrade",
 upgradeShortVersion:@"9.9.0",
 upgradeFullVersion:@"9.9.0.0000001"
 needClientNetType:@"4G,WIFI",
 userId:@"admin"
 }
 */
typedef void(^AliCheckUpgradeComplete)(NSDictionary *upgradeInfos);
typedef void(^AliCheckUpgradeFailure)(NSException *exception);



@interface MPCheckUpgradeInterface : NSObject

/**
 The interval between two single reminders, in days. Default value: 3.
 */
@property(nonatomic, assign) NSTimeInterval defaultUpdateInterval;

/**
 Modify the UI proxy of the default pop-up window.
 */
@property (nonatomic, weak) id<AliUpgradeViewDelegate> viewDelegate;

/**
 * Initialize the instance.
 */
+ (instancetype)sharedService;

/**
 Detect updates proactively. If there is an update, the default prompt of mPaaS is displayed in the pop-
up.
 *
 */
- (void)checkNewVersion;


/**
 Detect updates proactively. No pop-up is displayed. This method is usually used for UI customization, u
pdate detection, and red dot reminder.

 @param complete Callback succeeds, and the upgrade information dictionary is returned.
 @param failure Callback fails.
```

```
 */
- (void)checkUpgradeWith:(AliCheckUpgradeComplete)complete
                 failure:(AliCheckUpgradeFailure)failure;
@end
```

Developers can call the corresponding API to detect updates after the app is started. We recommend that you call the API after the homepage is displayed so that the startup time of the app is not affected. The following three calling methods are provided for different UI requirements for the display of upgrade prompt information:

- Use the default pop-up of mPaaS to display the upgrade prompt information.

```
- (void)checkUpgradeDefault {
    [[MPCheckUpgradeInterface sharedService] checkNewVersion];
}
```

- Customize the pop-up icon, network prompt toast, or progress bar of the network request group based on the default pop-up of mPaaS.

```
- (void)checkUpgradeWithHeaderImage {
    MPCheckUpgradeInterface *upgradeInterface = [MPCheckUpgradeInterface sharedService];
    upgradeInterface.viewDelegate = self;
    [upgradeInterface checkNewVersion];
}

- (UIImage *)upgradeImageViewHeader{
    return APCommonUILoadImage(@"ilustration_ap_expection_alert");
}

- (void)showToastViewWith:(NSString *)message duration:(NSTimeInterval)timeInterval {
    [self showAlert:message];
}

- (void)showAlert:(NSString*)message {
    AUNoticeDialog* alertView = [[AUNoticeDialog alloc] initWithTitle:@"Information" message:message
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
    [alertView show];
}
```

- Call the following API to obtain the upgrade information and customize the UI if the pop-up styles provided by mPaaS do not meet your requirements.

```
- (void)checkUpgradeWIthCustomUI {
    [[MPCheckUpgradeInterface sharedService] checkUpgradeWith:^(NSDictionary *upgradeInfos) {
        [self showAlert:[upgradeInfos JSONString]];
    } failure:^(NSException *exception) {
    }];
}
```

Configure whitelist gray release

To use the whitelist gray release function in release management, ensure that you have obtained the unique identity of the client on the server. Configure the unique user identity in category of **MPaaSInterface** on the client, and return a unique identity of the app in the `userId` method, for example, the user name, mobile phone number, email address, etc.
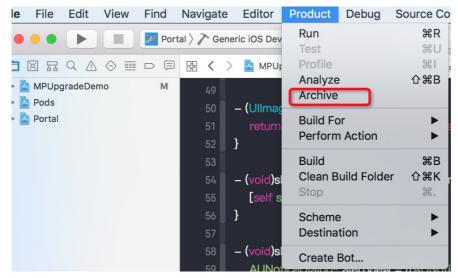
```
@implementation MPaaSInterface (Portal)

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

@end
```

For details on how to configure the whitelist in the mPaaS console, see Delivery Service > Manage whitelist.

Online release

Generate an .ipa file

- Use Xcode to generate an .ipa installation package.

- Alternatively, generate an .ipa installation package with the packaging function provided by the mPaaS plug-in. The generated package will be stored in the `product` directory under your current project.

  - **Bundle Identifier**: It must be consistent with `bundle Id` in the cloud configuration file.

  - **Bundle Version**: It must be consistent with `info.plist` in the `Production Version` file for the project.

  - **Provisioning Profile**: Signature configuration file. It must be consistent with `bundle Id`, or package generation will fail.

  - **Debug**: Specifies whether to generate the debug installation package.

  - **App Store**: Specifies whether to generate a package for release in the App Store.

Release a new version

Use the release management function of the release platform to release a new version. For details about the release process, see Release management.

Upgrade mode:

When creating a release task in the mPaaS console, choose one of the following upgrade modes:

- **Single reminder**: After a new version is released in the mPaaS console, the client calls the version upgrade API once and displays the pop-up only once in the silence period to avoid disturbing users.

  - This upgrade mode is suitable for instructing users to perform an upgrade upon the release of a new version.

  - By default, the silence period is 3 days, during which a user is notified only once. To change the length of the silence period, set the following properties before calling the upgrade detection API:

```
- (void)checkUpgradeDefault {
   [MPCheckUpgradeInterface sharedService].defaultUpdateInterval = 7;
   [[MPCheckUpgradeInterface sharedService] checkNewVersion];
   }
```

- **Multiple reminders**: After a new version is released in the mPaaS console, the client displays a pop-up reminder each time it calls the version upgrade API. This upgrade mode is suitable for instructing users to upgrade the app to the new version as soon as possible after the new version has been released for a period of time.

- **Mandatory reminder**: After a new version is released in the mPaaS console, the client displays a pop-up reminder without the Cancel button each time it calls the version upgrade API. Users can no longer use the app without an upgrade. This upgrade mode is suitable for forcing users to upgrade the app to the new version and unpublishing the earlier app version.

Related link

Code sample

4.3. Manage Android releases

Release management is the configuration backend for upgrading the client to a new version, which allows you to create multi-task and multi-dimensional upgrade configuration.

About this task

Android release management provides the following functions:

- Add upgrade resources and provide the QR code of the download link.

- Create and modify the task of the new version resource package.

- Create multiple types of release tasks for added release packages, such as whitelist gray release, time-window gray release, and official release. One upgrade package can have multiple release tasks.

- Support upgrade filtering by multiple criteria, such as the city, model, device system version, network, and release package version.

Add release packages

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

1. On the left navigation bar, choose **Mobile Delivery Service** > **Release management**. The release management list is displayed.

2. Click **Add a package**, and complete the following configuration in the pop-up window:

   ◦ **Platform**: Select **Android**.

   ◦ **Package**: Upload a local `.apk` release package.

   ◦ **Version**: Enter the version number of this package, which contains digits and characters.

   ◦ **Release description**: Describe this release package. Optional.

   ◦ **Download verification**: When enabled, after the QR code is scanned, the package can be downloaded only after the verification code is verified.

3. Click **OK**. The new release package is displayed on the top of the list.

   Note: After the release package is added, a QR code is generated in the **QR code** column for users to download the `.apk` release package. You can scan the QR code to download the release package to your phone.

4. In the release management list, click the plus icon (

   +

) in front of the release package to view the release task of the upgrade package:

   ◦ If the upgrade package has never been released, the status of the package is **To be released**, and there is no release task.

   ◦ If the upgrade package has been released, the status of the package is the release status of the latest task, and there are release tasks.

Create a release task

Create a release task for the added release package. Multiple release tasks are supported for one upgrade package. Up to 10 release tasks are allowed for one upgrade package concurrently.

Rules for delivering release tasks:

- When the client request matches multiple tasks, the higher version task takes priority.

- If a release package have multiple release tasks, by task type, the priority of the tasks is: Formal release > Whitelist gray release > Time window gray release.

- For a specific version release package, if the task type of the release tasks are same, the latest released task shall prevail.

  For example, for version 5.0 upgrade package, a whitelist release task A was created on the console to perform upgrade for version 4.0 app with single reminder; then a whitelist task B was created to perform a forced upgrade for version 4.0. With the two tasks performing at the same time, when a 4.0 version client requests an upgrade, task B is delivered first, and after task B is terminated or suspended, task A takes effect.

- If a grayscale task and a formal task are performed concurrently for a release package, the release status shows "Official release" in the release task list, and when the formal task is paused or ended, the release status changes to "Gray release". If all tasks are finished, it shows "Finished".

The operation steps are as follows:

1. Locate the release package for which you want to create a release task.

2. In the **Operation** column, click **Create release task**.

3. On the **Create a release task** page, select or enter the following information:

   ○ **Release type**: You can select **Gray release** and **Official release**.

   ○ **Upgrade mode**: You can select **Single**, **Multiple**, and **Forced upgrade**.

     ▪ **Single**: After the app is started, it displays an upgrade message based on the silent strategy.

       > ⑦ **Note**
       >
       > Silent strategy means that after the upgrade reminder pops up, after the user cancelled it, the reminder will be in Silent state for a period of time, and no longer reminds the upgrade. The default silent period is 3 days, which can be customized. To customize the silent period, see setIntervalTime.

     ▪ **Multiple**: The app displays an upgrade message every time it is started. Users can close the message window.

     ▪ **Forced upgrade**: The app displays an upgrade message each time it is started, and the message window cannot be closed.

   ○ **Release model** (only when **Gray release** is selected): You can select **Whitelist gray release** and **Time-window gray release**.

     ▪ If you select **Whitelist gray release**, you can configure a whitelist below.

       Note: You can configure a whitelist on the **Whitelist management** page. For more information, see Manage whitelist.

     ▪ If you select **Time-window gray release**, you can set **End time** and **Users count**.

   ○ **Upgrade prompt message**: Specify a message to be displayed upon an upgrade. Optional.

   ○ **Release description**: Describe this release. Optional.

   ○ **Advanced rule**: For **Gray release** only. Click **Add**. In the displayed dialog box, select a rule type such as **City**, **Device model**, or **Network** in **Type**, set **Operation type** to **Include** or **Exclude**, and set **Resource value** corresponding to the type you selected.

4. After completing the configuration, click **OK**. To view the release task that you have created, click the plus icon (

   +

) on the left of the release package.

5. Repeat the steps above if you need to create multiple release tasks.

Other operations

After creating a release task for the upgrade package, you can change the task status.

1. In the release management list, click the plus icon (

   +

) in front of the release package.

2. Perform the following operation based on your needs:

   ○ Click **Pause** to suspend the release task. To continue the task, click **Continue**.

   ○ Click **End** to terminate the release task. After the task is terminated, you can no longer operate it.

4.4. Manage iOS releases

Release management is the configuration backend for upgrading the client to a new version, which allows you to create multi-task and multi-dimensional upgrade configuration.

About this task

iOS release management provides the following functions:

- Add upgrade resources and provide the QR code (for **Enterprise distribution** only) for downloading the app.

- Create and modify the task of the new version resource package.

- Create multiple types of release tasks for added release packages, such as whitelist gray release, time-window gray release, and official release. One upgrade package can have multiple release tasks.
- Support upgrade filtering by multiple criteria, such as the city, model, device system version, network, and release package version.

Add release packages

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

1. In the left navigation bar, choose **Mobile Delivery Service** > **Release management**. The release management list is displayed.

2. Click **Add a package**, and complete the following configuration in the pop-up window:

   - **Platform**: Select **iOS**.
   - **Release type**: The options include App Store, Enterprise distribution, and TestFlight.
     - **App Store**: Upgrade prompts for apps downloaded from the App Store will be displayed.
     - **Enterprise distribution**: Upgrade prompts for apps distributed within enterprises will be displayed.
     - **TestFlight**: Gray release verification will be performed on a new version before its release in the App Store.

3. Click **OK**. The new release package is displayed on the top of the list.

4. In the release management list, click the plus icon (

   +

) in front of the release package to view the release task of the upgrade package:

   - If the upgrade package has never been released, the status of the package is **To be released**, and there is no release task.
   - If the upgrade package has been released, the status of the package is the release status of the latest task, and there are release tasks.

App Store

> ⑦ **Note**
>
> To use App Store release, first launch your App in the App Store.

If you select **App Store**, you should provide the following information:

- **App Store address**: Enter the address of your App in the App Store.
- **Version**: Enter the version number of current release.

  Note: The version number must keep consistent with the `Product Version` in info.plist file in the iOS project.

- **Release description**: Describe this release. Optional.

Enterprise distribution

If you select **Enterprise distribution**, you should provide the following information:

- **Upload icon(optional)**: Upload a picture in the `.jpg` or `.png` format. Optional.
- **Package**: Upload a local `.ipa` release package.
- **bundleId(optional)**: App bundleId. If left empty, the bundleId you entered in the the bundleId you entered on the code configuration page when downloading the configuration file will be used.
- **Version**: Enter the version number of this release.

  Note: The version number must keep consistent with the `Product Version` in info.plist file in the iOS project.

- **Release description(optional)**: Describe this release. Optional.
- **Download verification**: When enabled, after the QR code is scanned, the package can be downloaded only after the verification code is verified.

> **?** **Note**
>
> After the **Enterprise distribution** release package is added, a QR code is generated in the **QR code** column in the release management list for users to download the `.ipa` release package.

TestFlight

> **?** **Note**
>
> To use TestFlight testing features, you must have created and enabled public links in App Store Connect.
>
> TestFlight is only available in clients with version ≥ 10.1.32.
>
> The **Package Expiration Time** and **Tester Limit** you enter must match what you set in App Store Connect.

If you select **TestFlight**, provide the following information:

- **App Store Connect address**: Enter the public link address that you have created on App Store Connect. Ensure that the link is enabled.
- **Valid days**: Enter the number of valid days of the TestFlight package. It must be consistent with that you have set on App Store Connect.
- **Tester limit**: Enter the maximum number of persons involved in the test. It must be consistent with that you have set on App Store Connect.
- **Version**: Enter the version number of this release.

   Note: The version number must keep consistent with the `Product Version` in info.plist file in the iOS project.

- **Release description**: Describe this release. Optional.

Create a release task

Create a release task for the added release package. Multiple release tasks are supported for one upgrade package. Up to 10 release tasks are allowed for one upgrade package concurrently.

Rules for delivering release tasks:

- When the client request matches multiple tasks, the higher version task takes priority.

- If a release package have multiple release tasks, by task type, the priority of the tasks is: Formal release > Whitelist gray release > Time window gray release.

- For a specific version release package, if the task type of the release tasks are same, the latest released task shall prevail.

   For example, for version 5.0 upgrade package, a whitelist release task A was created on the console to perform upgrade for version 4.0 app with single reminder; then a whitelist task B was created to perform a forced upgrade for version 4.0. With the two tasks performing at the same time, when a 4.0 version client requests an upgrade, task B is delivered first, and after task B is terminated or suspended, task A takes effect.

- If a grayscale task and a formal task are performed concurrently for a release package, the release status shows "Official release" in the release task list, and when the formal task is paused or ended, the release status changes to "Gray release". If all tasks are finished, it shows "Finished".

The operation steps are as follows:

1. Locate the release package for which you want to create a release task.

2. In the **Operation** column, click **Create release task**.

3. On the **Create a release task** page, select or enter the following information:

   ○ **Release type**: You can select **Gray release** and **Official release**.

   ■ **Gray release**: Before the official release, release the package to some of the users to test and validate the functionality of the new package.

User Guide·Manage version upgrade

- **Official release**: Release the package officially to all the users.

  > ⑦ **Note**
  >
  > **TestFlight** and **enterprise distribution** types of distributions only support **gray** release. The TestFlight release page does not display the **release type** option, and the release package of the **enterprise distribution** type is fixed to the **gray** type and cannot be selected.

- **Upgrade mode**: You can select **Single**, **Multiple**, and **Forced upgrade**.
  - **Single**: After the App is started, it displays an upgrade message based on the silent strategy.

    > ⑦ **Note**
    >
    > Silent strategy means that the reminder will be in silent state for a period of time, and no longer reminds the upgrade after the user cancelled the pop-up upgrade reminder. The default silent period is 3 days, which can be customized. To customize the silent period, see Release a new version.

  - **Multiple**: The App displays an upgrade message every time it is started.
  - **Forced upgrade**: The App displays an upgrade message each time it is started. You cannot close the message window.

    > ⑦ **Note**
    >
    > For **TestFlight** release packages, only **Single** and **Multiple** are available.

- **Release model** (only when **Gray release** is selected): You can select **Whitelist gray release** and **Time-window gray release**.
  - If you select **Whitelist gray release**, you can configure a whitelist below.

    > ⑦ **Note**
    >
    > You can configure a whitelist on the **Whitelist management** page. For more information, see Manage whitelist.

  - If you select **Time-window gray release**, you can specify **End time** and **Users count**.

    > ⑦ **Note**
    >
    > For **Enterprise distribution** release packages, only **Whitelist gray release** is available.

- **Upgrade prompt message**: Specify a message to be displayed upon an upgrade. Optional.
- **Release description**: Describe this release. Optional.
- **Advanced rule**: For **Gray release** only. Click **Add**. In the displayed dialog box, select a rule type such as **City**, **Device model**, or **Network** in **Type**, set **Operation type** to **Include** or **Exclude**, and set **Resource value** corresponding to the type you selected.

4. After completing the configuration, click **OK**. To view the release task that you have created, click the plus icon (

   ＋

) on the left of the release package.

Other operations

- **Upload symbol table**. In the release management list, upload a symbol table to an added release package.
  - One `.ipa` release package corresponds to one symbol table file.
  - Only symbol tables in the `dSYM` format are supported. Before uploading a symbol table, compress it into a `.tgz` package.

- **Change the release task of an upgrade package**. Click the plus icon (

   ＋

) in front of the release package in the release management list to view the release task of the upgrade package.

> Document Version: 20231226

21

- Click **Pause** to suspend the release task. To continue the task, click **Continue**.
- Click **End** to terminate the release task. After the task is terminated, you can no longer operate it.

5.Manage offline packages

5.1. Configure offline packages

Before creating an offline package, you must add relevant configurations of the offline package on the **Manage Configuration** page.

Procedure

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

1. From the navigation bar on the left, click **Delivery Service** > **Manage offline packages**.

2. On the offline package list page, click the **Manage configuration** tab.

3. In the **Manage domain** section, enter the virtual domain name, for example `h5app.com` .

   Notes:

4. Check **I've confirmed the above information is accurate, and submit without any further change.** , and then click **Save**.

5. In the **Manage keys** section, upload the key file.

   Notes:

   - Generate private key:

     ```
     openssl genrsa -out private_key.pem 2048
     ```

   - Generate public key:

     ```
     openssl rsa -in private_key.pem -outform PEM -pubout -out public.pem
     ```

6. Check **I've confirmed the above information is accurate, and submit without any further change.** , and then click **Upload**. Offline package configuration is completed.

What to do next

- Generate offline packages

5.2. Generate offline packages

Based on different requirements, you can encapsulate different businesses into an offline package, and then distribute the package through the release platform to update the client-side resource.

Generating an offline package mainly includes the following two steps:

1. Build a frontend .zip package

2. Generate an .amr package online

Build a frontend .zip package

The paths of the resource packages vary by package type:

- Global resource package

- Normal resource package

> ⑦ **Note**
> - Global resource package and normal resource package cannot coexist in the same H5 offline package.
> - Offline package ID (namely the top-level directory mentioned below) must be a 8-digit number.

Global resource package

You can place the common resource which is referenced by other normal resource packages in the global resource package. It is required to specify the resource path within the package with following rules:

- Top-level directory: Offline package ID, for example: 77777777.

- Second-level directory: The server's domain address that is accessible to the resource.

- For public cloud: The secondary directory should be fixed as `mcube-prod.mpaascloud` Otherwise, the acceleration capability of real-time publishing and docking will not be available.
  - For private cloud: Refer to the domain address of mdsweb server deployed in the private cloud.
- Third-level directory: In the format of `appId_workspaceId` , for example `53E5279071442_test` .

  Avoid using special characters in folder names, file names, and files for public resource files. Special characters are characters that will be converted by the `urlencode` function. What follows the third-level directory is your custom public resource.

After the resource files are organized according to the above rules, you can fast locate the paths of resource files as in the following formats.

- Resource files on public cloud: `http://domain/appID_workspace/resource file path`
- Resource files on private cloud: `http://domain/mcube/appID_workspace/resource file path`

> ⑦ **Note**
>
> For the resource files on private cloud, you need to add `/mcube` after the second-level directory (server domain name) in the file path.
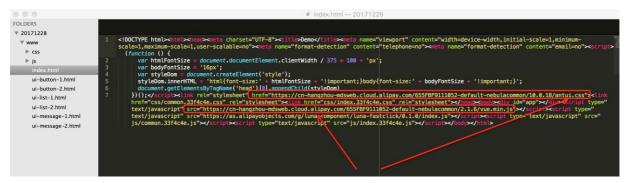
Example:

- In public cloud, the second-level directory is fixed to `mcube-prod.oss-cn-hangzhou.aliyuncs.com` , thus the path of the resource file `common.js` is `https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/53E5279071442_test/common.js` .

- In private cloud, the second-level directory is the domain name of the mdsweb server deployed in private cloud. Taking `mdsweb-outer.alipay.net` as an example, the path of the resource file `common.js` is `https://mdsweb-outer.alipay.net/mcube/53E5279071442_test/common.js` .
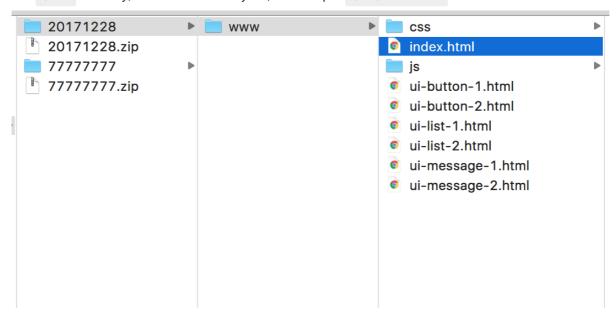

> ⑦ **Note**
>
> - The absolute length of the public resource cannot exceed 100 characters, otherwise the client might fail to load the resource and the page goes blank.
> - The server does not control the global resource package version. You can customize the version by adding a file directory structure after the third-level directory according to actual needs.
> - In private cloud, if the file storage format used by the server is hdfs or afs, you need to add a directory before the third-level directory mentioned above. The new directory name is the name of the storage space (bucket) in the mdsweb server.
> - To reference public resources, it means accessing the content of global resource packages via normal offline packages, so the access path must be absolute path, such as `https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/53E5279071442_test/common.js` .
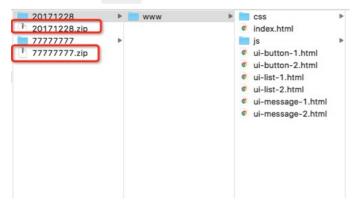
Normal resource package

You can place the relevant frontend resources such as HTML, CSS, and JavaScript into an offline package based on your business. The directory structure is as follows:

- Top-level directory: Offline package ID, for example: 20171228.

- Second-level and the following directories are your custom public resource. It is best to save all frontend files in a `/www` directory, and set a main entry file, for example: `/www/index.html`.

Generate the .zip package

After configuring the path of resource package, you can directly compress the whole directory where the appId is located into a `.zip` package.

Generate an .amr package on line

Log in to the mPaaS console, navigate to **Mobile Delivery Service** > **Manage offline packages**, and upload the `.zip` package generated in the previous step to the Mobile Delivery Service platform to generate a `.amr` package. For detailed steps, see Mobile Delivery Service > Create offline packages.

> **⚠ Important**
>
> - When you create an offline package, the minimum version of iOS client must be lower than the **Product Version** field in the `info.plist` file. You are recommended to set the minimum version of iOS client to **1.0.0**.
>
>   - The **Product Version** in the `info.plist` file should be consistent with the value of **Bundle versions string, short**, otherwise the offline package may not take effect.

5.3. Create offline packages

When creating an HTML5 offline package, you must complete the basic configuration of the offline package.

Prerequisites

You have configured an offline package on the **Manage configurations** page. For more information, see Configure offline packages.

About this task

You can create a single HTML5 offline package, or create multiple offline packages at a time by batch importing HTML5 offline package files.

When you upload the first offline package for an HTML5 app, you must select the offline package type. Once the offline package type is selected, it can not be changed. Each HTML5 app has only one type of offline package.

Procedure

Create a single offline package

Go to the mPaaS console, and perform the following steps:

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**.

2. On the offline package list page, click **Create an HTML5 app**. You can skip this step if an HTML5 application already exists.

3. In the **Create an HTML5 app** window, enter the **HTML5 app ID** and **HTML5 app name**, and then click **Submit**. You can skip this step if an HTML5 application already exists.

   HTML5 app ID must be a 8-digit number.

   > **⚠ Important**
   >
   > - The H5App ID is an 8-digit number.
   >
   > - 20000196, 66666692, 68687029, and 68687209 are offline package IDs built into the SDK. It is recommended not to use the H5App ID, otherwise conflicts will occur.
   >
   > - It is recommended not to use numbers starting with 666666 or 20000 for H5App ID.

4. Select the HTML5 app from the HTML5 app list, and click **Add an offline package** on the right.

5. Configure the following information in the **Basic information** section:

   - **Resource package type**: Select **Global resource package** or **Normal resource package**.

     > **❓ Note**
     >
     > If you use the global resource pack, you need to change the name of the secondary directory in the global resource pack to mcube-prod.mpaascloud.com, otherwise you will not be able to use the acceleration capability of real-time release docking.

   - **HTML5 app version**: Enter the version of the offline package, for example, 1.0.0.1.

   - **File**: Upload the offline package file in `.zip` format.

   - **Client version**: Select the type of the client and set the version range. Only the clients within the version range can receive the new offline packages.

> **Note**
> - At least one client type is required. If both Android and iOS are selected, you should ensure that the both clients adopt the same strategy on the latest version. **Specifically**, the latest versions of both clients should be either kept empty (system default) or **set as the same value**.
> - If the latest version is kept empty, all future versions will be supported. It is recommended to to set it as default, in case that the offline package becomes ineffective when the version of the upgraded client is higher than that is previously specified.
> - The version of the iOS client must be older than the value of **Product Version** field in the project's `info.plist` file.

6. In the **Configuration information** section, configure the following information:

   - **URL of main entrance**: Optional. The homepage of the offline package.

     > **Note**
     > A complete path is required, such as `/www/index.html`, where `/www` is the name of second-level directory you customized.

   - **Virtual domain**: The virtual domain name that you enter when you configure the offline package is automatically displayed.

   - **Extended information**: Optional. Enter the page loading parameters in key-value (KV) formats. Separate multiple KV pairs with commas (,).

     > **Note**
     > On the mPaaS platform, you can configure a request interval for HTML5 offline packages. You can apply the settings to a single offline package or globally.
     > - Single package configuration: Apply to the current offline package only. In the **Extended information** field, you can enter `{"asyncReqRate":"1800"}` to set the request interval, where `1800` indicates the interval. The interval is measured in seconds and ranges from 0 to 86400 seconds (0 to 24 hours). Value 0 indicates no limit on request interval.
     > - Global configuration: Apply to all offline packages. This parameter is specified in the client code. For more information, see Access Android client and Access iOS client.

   - **Network for download**: Choose the network environment for downloading the offline package. You can choose **Wi-Fi only** or **All networks**.
     - If you select **Wi-Fi only**, the offline package will be automatically downloaded in the background only with Wi-Fi connection.
     - If you select **All networks**, in non-Wi-Fi network, the offline package will still be automatically downloaded consuming user's mobile data. Thus, set it with caution.

   - **Time of installation**: Select the time to install the offline package.
     - If you select **Not preload**, the offline package will be installed only when the offline package is opened.
     - If you select **Preload**, the offline package will be automatically installed after the offline package is downloaded.

7. Check **I confirm the above information is accurate, and submit without any further change**, then click **Submit**.

Batch import offline packages

When creating multiple offline packages, you can choose to import the packages in batch to improve efficiency and avoid errors during configuration.

> **Note**
> - After importing, if the app to which the offline package belongs does not exist on mPaaS, an HTML5 app will be created automatically.
> - After importing, if the app to which the offline package belongs already exists on mPaaS, the package will be added to the HTML5 app when configuration is complete.

Go to the mPaaS console, and perform the following steps:

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**, then click **Batch import HTML5 app**.

2. In the popup window, follow the on-screen instructions to upload the ZIP offline package file and the configuration file.

> ⑦ **Note**
>
> - The file size cannot exceed 300 MB, and the number of offline packages cannot exceed 100.
> - The offline package resource file must be named after the offline package ID, which must be 8-digit number.

3. In the import result list, click **Edit** in the **Operation** column to edit the offline package. Refer to Create a single offline package for details about the configurations.

> ⑦ **Note**
>
> The default version of imported offline packages follow the rules below, and you can edit the versions based on your needs.
>
> - If the app to which the offline package belongs does not exist on mPaaS, the default version of the package is 0.0.0.1.
> - If the app to which the offline package belongs already exists on mPaaS, the default version of the package is to add 1 to the currently highest version number.

4. After completing editing all the packages, check **The information can't be modified after submission**, and click **Submit**.

   The submitted offline package information will be verified. If verification fails, error message will appear. If verification succeeds, the HTML5 offline package appears on the HTML5 offline package management page, which indicates that the offline package has been created successfully.

What to do next

Release offline packages

5.4. Release offline packages

To release the offline package that you created, you must create a release task and complete relevant configurations.

Procedure

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

1. On the left navigation bar, click **Delivery Service > Manage offline packages**.

2. On the offline package list page, select the offline package and version to be released, and then click **Create a release task**.

3. On the **Create a release task** page, complete the following configurations:

   - Release type: You can choose **Gray release** or **Official release**.

     - **Gray release**: Before the official release, release the package to some of the users to test and validate the functionality of the new package.

     - **Official release**: Release the package officially to all the users.

   - Release model: Select **Whitelist** or **Time window**. Available only when **Gray release** is selected.

     - If you select **Whitelist**, select a whitelist in **Whitelist**.

       Note: About creating whitelists, see Manage whitelists.

     - If you select **Time window**, select the **End time** and **Users count**.

   - Release description: Enter the description about the offline package release task.

   - Advanced rule: You can add one or more advanced rules for the release task. Optional field, available only when **Gray release** is selected.

     - Type: Select **City**, **Model**, **Network** or **Device OS** version.

     - Operation type: Select the operation type.

- Resource value: From the drop-down menu, select the resource value that corresponds to the selected operation type.

4. Click **OK**.

Result

On the offline package list page, you can find the status of the released package: **Gray releasing** or **Official releasing**.

Note: Due to the cache refresh mechanism of the server, the client will not receive it until about 1 minute later after you release the offline package from the console.

What to do next

Manage the released offline packages

5.5. Manage offline packages

You can manage the released HTML5 offline packages. The management operations include viewing, suspending and ending release tasks, and deleting HTML5 offline packages.

View offline package release tasks

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

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**.

2. In the HTML5 app list, select the target app, choose the offline package version that you want to view from the corresponding offline package list, and click unfold icon (+) left to the offline package version.

3. In the unfolded task list, click **View** to view the release task details.

Suspend offline package release tasks

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

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**.

2. In the HTML5 app list, select the target app, choose the offline package version that you want to suspend from the corresponding offline package list, and click unfold icon (+) left to the offline package version.

3. Click **Pause**, and then click **OK** to confirm the operation.

   To resume releasing the offline package, click **Continue**.

Terminate offline package release tasks

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

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**.

2. In the HTML5 app list, select the target app, choose the offline package version that you want to suspend from the corresponding offline package list, and click unfold icon (+) left to the offline package version.

3. Click **End**, and then click **OK** to confirm the operation.

   > ⑦ **Note**
   >
   > - After terminating a release task, to release the offline package again, you need to create a new release task.
   > - The offline package cannot be downloaded when its release task is terminated. But other versions of the package will not be affected. For example, the release of offline package V1.1 is terminated, and offline package V1.0 is still in release, then package V1.1 cannot be downloaded on the client, while package V1.0 can still be downloaded.

Export offline packages

You can download the resource file (.arm) or configuration file (.json) of a single offline package

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

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**.

2. In the HTML5 app list, select the target app, choose the offline package version from the corresponding offline package list, and click **Download AMR file** or **Download configuration file** to download the resource file or configuration file.

Delete an HTML5 app

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

1. On the left navigation bar, click **Mobile Delivery Service** > **Offline package management**.

2. In the HTML5 app list, select the target app that you want to delete, hover mouse over the app, and click the deletion icon to delete it. When the app is deleted, all the offline packages and resource files of the app will be deleted.

> ⑦ **Note**
>
> The HTML5 app cannot be restored once deleted.

5.6. OpenAPI

5.6.1. Overview and preparation

MDS provides Java SDK for offline package release, enabling developers to configure, create, release, and manage offline packages by calling the API.

Preparation

Before using the open API, you need to obtain AccessKey, App ID, Workspace ID and Tenant ID, configure Maven dependencies and configure file uploading.

Obtain AccessKey

AccessKey includes **AccessKey ID** and **AccessKey Secret**. Click here for obtaining method.

- **AccessKey ID**: used to identify users.
- **AccessKey Secret**: used for user authentication. MUST be kept safe.

Obtain App ID, Workspace ID and Tenant ID

1. Log in to mPaaS console, and enter the App.

2. In **Overview** page, click **Code configurations** (choose Android or iOS based on your needs)> **Download configuration file** > **Download now**. You can view App ID and Workspace ID in the **Code configurations** panel.

Configure Maven dependencies

Before using the API, you need to complete the following Maven dependency configurations.

```
<dependency>
   <groupId>com.aliyun</groupId>
   <artifactId>aliyun-java-sdk-mpaas</artifactId>
   <version>3.0.5</version>
</dependency>

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

Environment Variable Configuration

Configure environment variable MPAAS_AK_ENV and MPAAS_SK_ENV

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

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

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

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

Code sample

The following shows the code sample of using Client in Maven:

```java
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.mpaas.model.v20201028.QueryMcubeVhostRequest;
import com.aliyuncs.mpaas.model.v20201028.QueryMcubeVhostResponse;
import com.aliyuncs.profile.DefaultProfile;

public class MpaasApiDemo {

    /**
     * The corresponding APP ID on the mPaaS console
     */
    private static final String APP_ID = "ALIPUB40DXXXXXXX";

    /**
     * The corresponding workspace id on the mPaaS console
     */
    private static final String WORKSPACE_ID = "default";

    /**
     * The corresponding tenant id on the mPaaS console
     */
    private static final String TENANT_ID = "XVXXXXXF";

    /**
     * Region ID, the default is cn-hangzhou
     */
    private static final String REGION_ID = "cn-hangzhou";

    /**
     * Product name
     */
    private static final String PRODUCT = "mpaas";

    /**
     * The endpoint to call
     */
    private static final String END_POINT = "mpaas.cn-hangzhou.aliyuncs.com";

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

        DefaultProfile.addEndpoint(REGION_ID, PRODUCT, END_POINT);
```

```
        DefaultProfile profile = DefaultProfile.getProfile(REGION_ID, accessKeyId, accessKeySecret);
        IAcsClient iAcsClient = new DefaultAcsClient(profile);
        QueryMcubeVhostRequest queryMcubeVhostRequest = new QueryMcubeVhostRequest();
        queryMcubeVhostRequest.setAppId(APP_ID);
        queryMcubeVhostRequest.setWorkspaceId(WORKSPACE_ID);
        queryMcubeVhostRequest.setTenantId(TENANT_ID);
        QueryMcubeVhostResponse acsResponse = null;
        try {
            acsResponse = iAcsClient.getAcsResponse(queryMcubeVhostRequest);
            System.out.println(acsResponse.getResultCode());
            System.out.println(acsResponse.getQueryVhostResult());
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

Configure file uploading

Since file streaming is not allowed in all APIs, to upload a file, you need to upload it to OSS first by calling the upload tool class, and then send the returned OSS address as a parameter to the specified API.

You can download the file upload tool class OssPostObject.java.zip.

Code sample

The following shows the code sample of file uploading:

```
        GetMcubeFileTokenRequest getMcubeFileTokenRequest = new GetMcubeFileTokenRequest();
        getMcubeFileTokenRequest.setAppId(APP_ID);
        getMcubeFileTokenRequest.setOnexFlag(true);
        getMcubeFileTokenRequest.setTenantId(TENANT_ID);
        getMcubeFileTokenRequest.setWorkspaceId(WORKSPACE_ID);
        GetMcubeFileTokenResponse acsResponse = iAcsClient.getAcsResponse(getMcubeFileTokenRequest);
        System.out.println(JSON.toJSONString(acsResponse));

        GetMcubeFileTokenResponse.GetFileTokenResult.FileToken fileToken =
 acsResponse.getGetFileTokenResult().getFileToken();
        OssPostObject ossPostObject = new OssPostObject();
        ossPostObject.setKey(fileToken.getDir());
        ossPostObject.setHost(fileToken.getHost());
        ossPostObject.setOssAccessId(fileToken.getAccessid());
        ossPostObject.setPolicy(fileToken.getPolicy());
        ossPostObject.setSignature(fileToken.getSignature());
        ossPostObject.setFilePath("your/local/file/path");
        String s = ossPostObject.postObject();
```

Refer to Obtain upload file token for descriptions about `GetMcubeFileTokenRequest` .

5.6.2. API description

General parameter description

All interfaces contain three parameters: `appId` , `workspaceId` and `tenantId` . The meanings of these three parameters are as follows. The three parameters will be omitted in subsequent interface descriptions of this document.

Parameter	Type	Description
appId	String	App ID.

workspaceId	String	Workspace ID.
tenantId	String	Tenant ID.

General response description

Parameter	Type	Description
resultCode	String	Normally, the code returned is `OK`. Other values indicate that the API request is abnormal.
requestId	String	Request ID.
resultMessage	String	Returned value after query failure.
***Result	Object	The objects returned. The actual meaning varies with the value returned.

The objects returned include the following fields:

Name	Type	Description
resultMsg	String	Returned value after query failure.
success	Boolean	Whether the query is successful.

Create virtual domain

Request - CreateMcubeVhostRequest

Parameter	Type	Description
vhost	String	Virtual domain name.

Response - CreateMcubeVhostResponse

```
{
    "createVhostResult":{
        "data":"success",
        "resultMsg":"",
        "success":true
    },
    "requestId":"F9C681F2-6377-488D-865B-1144E0CE69D2",
    "resultCode":"OK"
}
```

Response description

Name	Type	Description

createVhostResult	Object	The objects returned, includes general response only. See General response description for details.

Query virtual domain

Request - QueryMcubeVhostRequest

Includes general parameters only. See General parameter description for details.

Response - QueryMcubeVhostResponse

```
{
    "queryVhostResult":{
        "data":"test.com",
        "resultMsg":"",
        "success":true
    },
    "requestId":"637D5BE0-0111-4C53-BCEE-473CFFA0DBAD",
    "resultCode":"OK"
}
```

Response description

Response name	Type	Description
queryVhostResult	Object	The objects returned. See the table below for meanings.

The objects returned include the following fields:

Name	Type	Description
data	String	The virtual domain name queried.

Query if key file exists

Request - ExistMcubeRsaKeyRequest

Includes general parameters only. See General parameter description for details.

Response - ExistMcubeRsaKeyResponse

```
{
    "checkRsaKeyResult":{
        "data":"fail",
        "resultMsg":"",
        "success":false
    },
    "requestId":"8F76783A-8070-4182-895D-14E5D66F8BA3",
    "resultCode":"OK"
}
```

Response description

Response name	Type	Description
checkRsaKeyResult	Object	The objects returned. See the table below for meanings.

The objects returned include the following fields:

Name	Type	Description
data	String	Query result: `fail` indicates the key does not exist, and `success` indicates the key exists.

Obtain upload file token

Request - GetMcubeFileTokenRequest

Parameter	Type	Description
onexFlag	Boolean	The fixed value is `true`.

Response - GetMcubeFileTokenResponse

```
{
    "getFileTokenResult":{
        "fileToken":{
            "accessid":"LTAI7z7XPfKU****",
            "dir":"mds/tempFileForOnex/ONEXE9B092D/test/PUQYHL/8b574cb7-3596-403f-a0e9-208660fc2081/",
            "expire":"1584327372",
            "host":"https://mcube-test.oss-cn-hangzhou.aliyuncs.com",
            "policy":"QwM2YtYTBlOS0yMDg2NjBmYzIwODEvIl1dfQ==",
            "signature":"kisfP5YhbPtmES8+w="
        },
        "resultMsg":"",
        "success":true
    },
    "requestId":"8BAA3288-662E-422C-9960-2EEBFC08369F",
    "resultCode":"OK"
}
```

Response description

Response name	Type	Description
getFileTokenResult	Object	The objects returned, includes general response only. See General response description for details.

Upload key file

Request - UploadMcubeRsaKeyRequest

Parameter	Type	Description
onexFlag	Boolean	The fixed value is `true`.
fileUrl	String	The save address of the key file in OSS.

Response - UploadMcubeRsaKeyResponse

```
{
    "requestId":"519E35CF-CC60-4890-8C8E-89A98CEA6BB0",
    "resultCode":"OK",
    "uploadRsaResult":{
        "data":"processed successfully",
        "resultMsg":"",
        "success":true
    }
}
```

Response description

Response name	Type	Description
uploadRsaResult	Object	The objects returned, includes general response only. See General response description for details.

Obtain offline package app list

Request - ListMcubeNebulaAppsRequest

Includes general parameters only. See General parameter description for details.

Response - ListMcubeNebulaAppsResponse

```
{
    "listMcubeNebulaAppsResult":{
        "nebulaAppInfos":[
            {
                "h5Id":"12345678",
                "h5Name":"12345678"
            },
            {
                "h5Id":"12345679",
                "h5Name":"openapiTest"
            }
        ],
        "resultMsg":"",
        "success":true
    },
    "requestId":"BE728F09-6EBD-4688-9329-896813EAD075",
    "resultCode":"OK"
}
```

Create offline package app

Request - CreateMcubeNebulaAppRequest

Parameter	Type	Description
h5Name	String	Offline package name.
h5Id	String	Offline package ID. 8 digits.

Response - CreateMcubeNebulaAppResponse

```
{
    "createNebulaAppResult":{
        "resultMsg":"",
        "success":true
    },
    "requestId":"5B588AFE-8D58-4460-B0AA-6A48A9FD0852",
    "resultCode":"OK"
}
```

Delete offline package app
Request - DeleteMcubeNebulaAppRequest

Parameter	Type	Description
h5Id	String	Offline package ID. 8 digits.

Response - DeleteMcubeNebulaAppResponse

```
{
    "deleteMcubeNebulaAppResult":{
        "resultMsg":"",
        "success":true
    },
    "requestId":"E24C760E-4849-4341-91C6-6DA97F5B6B76",
    "resultCode":"OK"
}
```

Upload offline package
Request - CreateMcubeNebulaResourceRequest

Name	Type	Description
h5Id	String	ID of the HTML5 app.
h5Name	String	Name of the HTML5 app.
h5Version	String	Version of the offline package. Must be unique in an HTML5 app.
mainUrl	String	Main URL of the offline package. Should satisfy the regular expression pattern: `^/[\w|/]+\.html$`.
vhost	String	Virtual domain name of the HTML5 app.
extendInfo	String	Extended fields in JSON format.

autoInstall	Integer	Specify the network in which downloads are allowed. 0: Wi-Fi only (Without Wi-Fi connection, download starts only when users use the app). 1: All networks (Consumes cellular data. Do not choose this mode unless in special situations.)
resourceType	Integer	Resource type. One HTML5 app can have only one resource type. 0: Global resource. 1: Private resource.
installType	Integer	Specify whether to preload the offline package before installing it. 0: Not preload (install only when the user enters the offline package page) 1: Preload (automatically install after the offline package is downloaded)
platform	String	Platform. Includes all, Android and iOS.
clientVersionMin	String	Minimum client version. Minimum version is required when platform is chosen. The format is `aaa;bbb`. `aaa` indicates iOS client version, and `bbb` indicates Android client version. The semicolon ; cannot be omitted. If Android is chosen as the platform, then the value is `;bbb`.
clientVersionMax	String	Maximum client version. Can be empty. If the value of platform is `all`, then this value must appear in pairs. That is, iOS and Android version must both be included, or neither of them is included.
fileUrl	String	The OSS URL of the offline package file. The package must be a ZIP file.
repeatNebula	Integer	Whether to reuse the global package. Required when the resource package is global resource. 0: No, 1: Yes.
onexFlag	Boolean	The fixed value is `true`.

Response - CreateMcubeNebulaResourceResponse

```
{
    "createMcubeNebulaResourceReslult":{
        "nebulaResourceId":"4154",
        "resultMsg":"",
        "success":true
    },
    "requestId":"DFCA28DF-0F97-4C41-B3D4-351D284B51E7",
    "resultCode":"OK"
}
```

`nebulaResourceId` is the ID of the offline package uploaded.

Obtain resource package list

Request - ListMcubeNebulaResourcesRequest

Parameter	Type	Description
h5Id	String	HTML5 app ID.

Response - ListMcubeNebulaResourcesResponse

Response description

Response name	Type	Description
appCode	String	appId+"-"+workspaceId
autoInstall	Integer	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.
clientVersionMax	String	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.
clientVersionMin	String	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.
creator	String	Creator. Currently not in use.
debugUrl	String	Has no meaning in current response.
downloadUrl	String	Download address of offline package ARM file.
extendInfo	String	The extended fields passed in the offline package upload request.
extraData	String	Extended parameters.

fallbackBaseUrl	String	Offline package fallback address, delimited by semicolon (`;`). The address before `;` is intranet address, and the address after `;` is internet address.
fileSize	String	File size
gmtCreate	Date	Time of creation
gmtModified	Date	Time of update
h5Id	String	ID of the HTML5 app.
h5Name	String	Name of the HTML5 app.
h5Version	String	Version of the current offline package package
id	Long	Primary key.
installType	Integer	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.
lazyLoad	Integer	Lazy loading. Currently the value is 0.
mainUrl	String	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.
md5	String	md 5 of the offline package file.
memo	String	Download address of the offline package `h5.json` file.
metaId	Long	Has no meaning in current response.
modifier	String	Modifier. Currently not in use.
platform	String	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.

		Release status.
publishPeriod	Integer	0: Initialization 1: Internal gray release 2: External gray release 3: Formal release 4: Rollback release 5: Release task ends
releaseVersion	String	Release version.
resourceType	Integer	The meaning is the same as the parameter in the offline package upload request. See Upload offline package for details.
status	Integer	Status

Create Offline package release task

Request - CreateMcubeNebulaTaskRequest

Parameter	Type	Required	Description
publishType	Integer	Yes	Release type. • 2: Gray release • 3: Official release
publishMode	Integer	Yes	Release mode. If publishType is `3`, this field should be empty. • 0: Unknown • 1: Whitelist • 2: Time window • 3: Percentage • 4: Full quantity • 5: Third party grayscale
memo	String	No	Release note.
id	Long	Yes	Only `0` is allowed. The ID indicates creation, and cannot be modified.

			End time of time window grey release. Required when `publishMode` is `2`. Format: `YYYY-MM-dd HH:mm:ss`. The time must be greater than the current time and less than 7 days from the current time.
greyEndtimeData	String	No	
greyEndTime	Date	No	Date type. The value is same as that of `greyEndTimeData`.
greyNum	Integer	No	Number of users in time window grey release. Required when `publishMode` is `2`.
whitelistIds	String	No	Primary key ID of whitelist. Required when `publishMode` is `1`. Separate multiple IDs with comma `,`.
packageId	Long	Yes	Primary key ID of the resource package.
greyConfigInfo	String	No	Advanced rule, JSON string. See the table below for meanings. Example: [{"ruleElement":"city","operation":1,"value":"Shanghai,Beijing,Tianjin"}, {"ruleElement":"mobileModel","operation":2,"value":"REDMI NOTE 3,VIVO X5M"}, {"ruleElement":"osVersion","operation":3,"value2":"9.2.1","value1":"9.2.1","value":"9.2.1-9.2.1"}]

Advanced rule description

Name	Type	Description
ruleElement	String	Rule type: • city: City • mobileModel: Mobile phone model • netType: Network • osVersion: Device OS version

value	String	Rule value. Separate multiple values with comma (,). When operation is `3` or `4` , the value is in `aa-bb` format, in which `aa` is the smaller value, and `bb` is the greater value.
operation	Integer	Operation: • 1: Include • 2: Exclude • 3: Within range • 4: Out of range If `ruleElement` is `city` , `mobileModel` and `netType` , operation value can only be `1` or `2` If `ruleElement` is `osVersion` , the value of operation can be any one of the four value.

Response - CreateMcubeNebulaTaskResponse

```
{
    "createMcubeNebulaTaskResult":{
        "nebulaTaskId":"6664",
        "resultMsg":"",
        "success":true
    },
    "requestId":"BBDF54E1-2783-4E5A-AE19-F7BC3A1BB3C2",
    "resultCode":"OK"
}
```

`nebulaTaskId` is the created release task ID.

Obtain release task list

Request - ListMcubeNebulaTasksRequest

Parameter	Type	Description
id	Long	ID of the offline package corresponding to the release task.

Response - ListMcubeNebulaTasksResponse

```json
{
    "listMcubeNebulaTaskResult":{
        "nebulaTaskInfos":[
            {
                "appCode":"ONEX97C5D29290957-default",
                "bizType":"nebula",
                "creator":"",
                "gmtCreate":"2021-02-01 14:16:58",
                "gmtModified":"2021-02-01 14:16:58",
                "gmtModifiedStr":"2021-02-01 14:16:58",
                "greyConfigInfo":"",
                "greyEndtimeData":"",
                "greyNum":0,
                "greyUrl":"",
                "id":6664,
                "memo":"test",
                "modifier":"",
                "packageId":4154,
                "percent":0,
                "platform":"all",
                "productId":"ONEX97C5D29290957-default-12345678",
                "productVersion":"1.0.0.1",
                "publishMode":4,
                "publishType":3,
                "releaseVersion":"20210201141121",
                "status":1,
                "syncResult":"",
                "taskName":"12345678",
                "taskStatus":1,
                "taskType":0,
                "taskVersion":1612160218556,
                "upgradeNoticeNum":0,
                "upgradeProgress":"",
                "whitelistIds":""
            }
        ],
        "resultMsg":"",
        "success":true
    },
    "requestId":"B9A07543-4B8B-43D0-AB33-7F2ACB954909",
    "resultCode":"OK"
}
```

Response description

Name	Type	Description
appCode	String	appId+workspaceId
bizType	String	The value for offline package is `nebula`.
bundles	Array	Currently not in use.
creator	String	Currently not in use.
gmtCreate	Date	Time of creation.

gmtModified	Date		Time of update.
gmtModifiedStr	String		Update time string.
greyConfigInfo	String		Advanced rule string, different from that in the upload request. See greyConfigInfo explanation for details.
greyEndtime	Date		End time of time window grey release.
greyEndtimeData	String		End time string of time window grey release.
greyNum	Integer		Number of users in time window grey release.
id	Long		Primary key ID of current release task.
memo	String		Release note.
modifier	String		Modifier. Currently not in use.
packageId	Long		ID of the offline package corresponding to the current task.
percent	Integer		Grey percent. Currently the value is 0.
platform	String		Platform of the release task. Includes all, Android and iOS.
productId	String		Product ID. The format is `appId + workspaceId + h5id`.
productVersion	String		ID of the offline package.
publishMode	Integer		Release mode: • 0: Default • 1: Whitelist • 2: Time window
publishType	Integer		Release type: • 2: Gray release • 3: Official release
releaseVersion	String		Internal release version.
resIds	String		ID of the corresponding offline package.

status	Integer	Status: • 0: Invalid • 1: Valid
syncResult	String	Currently not in use
taskName	String	Task name, same as the HTML5 app name.
taskStatus	Integer	Task status: • 0: To be released • 1: Release in progress • 2: Finished • 3: Paused
taskType	Integer	Task type: • 0: Ordinary task. • 1: Rollback task.
taskVersion	Long	Task version, uses the time of task creation.
upgradeNoticeNum	Integer	Currently not in use
upgradeProgress	String	Currently not in use
whitelistIds	String	Primary key ID of whitelist, delimited by comma (`,`).

greyConfigInfo explanation

Name	Type	Description
operator	String	Relationship of the rules. `and` means all the rules in `subRules` must be met at the same time.
defaultResult	boolean	The default returned result.
subRules	List	Rule list.
operator	String	Rule name: • contains: Include • excludes: Exclude • vLimitIn: Within range • vLimitOut: Out of range

left	List<String>/Object	• When `operator` value is `contains` or `excludes`, the value is a list of elements, and each element represents a rule value. • When `operator` value is `vLimitIn` or `vLimitOut`, it is an object, and `lower` represents the smaller value, and `upper` represents the greater value.
right	String	Rule type name.
defaultResult	Boolean	Default result.

Obtain release task details by ID

Request - GetMcubeNebulaTaskDetailRequest

Parameter	Type	Description
taskId	Long	Primary key ID of the release task.

Response - GetMcubeNebulaTaskDetailResponse

```
{
    "getMcubeNebulaTaskDetailResult":{
        "nebulaTaskDetail":{
            "appCode":"ONEX97C5D29290957-default",
            "appId":"",
            "atomic":0,
            "baseInfoId":0,
            "bizType":"nebula",
            "creator":"",
            "cronexpress":0,
            "downloadUrl":"https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345
678/1.0.0.1_all/nebula/12345678_1.0.0.1.amr;https://pre-mpaas.cn-
hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/12345678_1.0.0.1.amr",
            "extraData":"{"resourceType":"1"}",
            "fileSize":"0",
            "fullRepair":0,
            "gmtCreate":"2021-02-01 14:16:58",
            "gmtModified":"2021-02-01 14:16:58",
            "gmtModifiedStr":"2021-02-01 14:16:58",
            "greyConfigInfo":"",
            "greyEndtimeData":"",
            "greyNum":0,
            "greyUrl":"",
            "id":6664,
            "issueDesc":"",
            "memo":"test",
            "modifier":"",
            "ossPath":"",
            "packageId":4154,
            "percent":0,
            "platform":"all",
            "productId":"ONEX97C5D29290957-default-12345678",
            "productVersion":"1.0.0.1",
            "publishMode":4,
            "publishPeriod":3,
            "publishType":3,
            "quickRollback":0,
            "releaseVersion":"20210201141121",
            "ruleJsonList":[

            ],
            "sourceId":"",
            "sourceName":"",
            "sourceType":"",
            "status":1,
            "syncResult":"",
            "syncType":0,
            "taskName":"12345678",
            "taskStatus":1,
            "taskType":0,
            "taskVersion":1612160218556,
            "upgradeNoticeNum":0,
            "upgradeProgress":"",
            "whitelistIds":"",
            "workspaceId":""
        },
        "resultMsg":"",
        "success":true
    },
    "requestId":"072AE251-B9F8-4A44-A621-9F0325EECC1E",
    "resultCode":"OK"
}
```

Response description

Response name	Type	Description
appCode	String	appId+workspaceId
appId	String	Currently not in use.
atomic	Integer	Whether is package is atomic or not. Currently can be ignored. 1: Yes. 0: No.
baseInfoId	Long	The primary key ID of basic information. Currently can be ignored.
bizType	String	The value is `nebula` for offline package.
bundles	List	Currently not in use.
creator	String	Currently not in use.
cronexpress	Integer	0: Execute once. 1: Execute multiple times. The value is `0` for iOS.
downloadUrl	String	Offline package download address. The address before `;` is intranet address, and the address after `;` is internet address.
extraData	String	Extended data in JSON format.
fileSize	String	File size
gmtCreate	Date	Time of creation
gmtModified	Date	Time of update
greyConfigInfo	String	Advanced rule string.
greyEndTime	Date	End time of time window grey release.
greyEndtimeData	String	End time string of time window grey release.
id	Long	Primary key ID

issueDesc	String	Issue description. Currently not in use.
mds	String	md 5 of the offline package file.
memo	String	Release note.
modifier	String	Modifier. Currently not in use.
ossPath	Stirng	Currently not in use.
packageId	Long	Offline package ID.
percent	Integer	Gray percent. Currently not in use.
platform	String	Platform, all, iOS, Android.
product_id	Sting	appId+workspaceId + H5Appid
productVersion	String	Offline package version.
resIds	String	Offline package ID.
ruleJsonList	List	Release advanced rules. See the sample above for details.
sourceId	String	Source ID. Currently not in use for offline package.
sourceName	String	Currently not in use for offline package.
sourceType	String	Source type. Currently not in use for offline package.
status	Integer	Status. 0: Invalid, 1: Valid.
syncResult	String	Currently not in use for offline package.
syncType	String	Currently not in use for offline package.
taskName	String	Task name

taskStatus	Integer	Task status: • 0: To be released • 1: Release in progress • 2: Finished • 3: Paused
taskType	Integer	Task type: • 0: Ordinary task. • 1: Rollback task.
taskVersion	Long	Task version, uses the time of task creation.
upgradeNoticeNum	Integer	Currently not in use
upgradeProgress	String	Currently not in use
vmType	Stirng	Android emulator type, separated by comma. 1: art 2: dalvik 3: lemur 4: aoc
whitelist	List	Whitelist information of the offline package release task. Refer to Manage whitelists for detalis.

Change offline package task status

Request - ChangeMcubeNebulaTaskStatusRequest

Parameter	Type	Description
bizType	String	Pass `nebula` for offline package.
packageId	Long	Offline package ID.
taskId	Long	Release task ID.
taskStatus	Integer	The status to change to. 0: To be released 1: Release in progress 2: Finished 3: Paused

Response - ChangeMcubeNebulaTaskStatusResponse

```
{
    "changeMcubeNebulaTaskStatusResult":{
        "resultMsg":"",
        "success":true
    },
    "requestId":"595F4CB4-ACFE-4A5B-AF5B-4ED837CAEF95",
    "resultCode":"OK"
}
```

6.Configuration Management

6.1. Android

This topic briefly describes how to fast integrate the switch configuration function to the Android client.

The switch configuration service enables a client to dynamically modify the processing logic in the client code without releasing a new client version. The client controls related processing based on the switch value dynamically obtained from the backend. With the switch configuration service, you can configure, modify, and push various switches. A switch is a `key-value` pair.

Currently, you can access switch configuration function through Native AAR or Portal & Bundle.

The complete access process mainly includes the following 3 steps:

1. Add SDK
2. Initialize mPaaS (only required for Native AAR)
3. Use the SDK

Prerequisites

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

Add SDK

Native AAR mode

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

Componentized access mode

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

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

Initialize mPaaS

If you access 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.

Use the SDK

mPaaS provides the switch configuration management API `MPConfigService` to implement switch configuration.

Complete the following steps to implement switch configuration:

1. In the mPaaS console, go to the **Mobile Delivery Service** > **Manage configuration** page, add desired switch configuration items, and set them based on information such as the platform, whitelist, percentage, version number, model, and Android version. For more information, see Configuration management.

2. After a switch key is released in the console, the client can call the specified API to obtain the value of the switch key.

The switch configuration management API `MPConfigService` provides many APIs externally. You can understand the function of each API based on its name. A list of APIs is provided as follows:

```
public class MPConfigService {
    /**
     * Obtain switch configurations
     *
     * @param key
     * @return
     */
    public static String getConfig(String key);
    /**
     *  Load switch configurations. The latest switch configurations are obtained every half an hour
by default.
     */
    public static void loadConfig();
    /**
     * Load switch configurations immediately.
     *
     * @param delay Delay after which switch configurations are loaded, in ms. 0 indicates that
switch configurations are loaded immediately.
     */
    public static void loadConfigImmediately(long delay);
    /**
     * Register a listener for listening to switch configuration changes.
     * @param configChangeListener listener
     * @return
     */
    public static boolean addConfigChangeListener(ConfigService.ConfigChangeListener
configChangeListener);
    /**
     * Remove the listener for listening to switch configuration changes
     * @param configChangeListener listener
     */
    public static void removeConfigChangeListener(ConfigService.ConfigChangeListener
configChangeListener);
}
```

> ⓘ **Important**
>
> As a soft reference object, the listener will be reclaimed when the system has run out of memory. Therefore, please perform switch monitoring instead of global monitoring; meanwhile, register the listener at regular intervals and remove it after use.

6.2. iOS

> **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.

Switch configuration refers to the capability of dynamically modifying client-side code processing logic without releasing a new version on the client. The client controls relevant processing according to the extracted switch value that is dynamically configured at backend. By using the switch configuration service, you can configure, modify and push various types of switches. The switch refers to `key/value` pair.

mPaaS provides the configuration management service ConfigService to realize switch configuration. By default, the syncing logic is syncing a switch value at the cold start of the application, or syncing a switch value if **half an hour** passed since last sync after the application returns to the foreground. In addition, ConfigService also provides an interface for immediate sync and the logic for monitoring configuration changes, by which configuration can be refreshed as soon as it is changed.

To realize switch configuration management, you need to add the corresponding iOS SDK, configure the project and read the configuration.
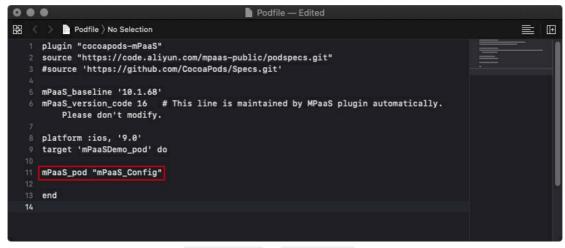
Prerequisites

The project is connected to mPaaS. For more information, refer to: Access based on native framework and using Cocoapods.

About this task

This document introduce how to integrate the switch configuration in details based on the switch configuration code sample.

Add an SDK

1. In the Podfile, define `mPaaS_pod "mPaaS_Config"` to add the dependencies of the switch configuration component.
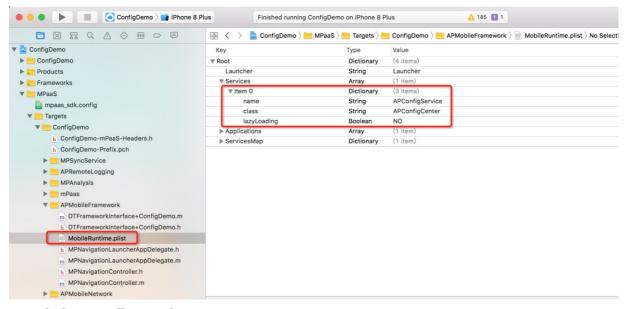

2. Based on requirements, run the `pod install` or `pod update` command.

Configure a project

Note: This step is applicable to baseline 10.1.32. The project configuration feature is built in baselines 10.1.60 and 10.1.68. Therefore, you can ignore this step when baseline 10.1.60 or 10.1.68 is used.

mPaaS encapsulates the switch configuration capability to a service. and you need to register this service in service manager before using it, as shown in the following figure.

Read the configurations

Values of switch keys can be dynamically released in the mPaaS console. On the left navigation pane, choose **Mobile Delivery Service** > **Configuration management**. Click **Configuration Keys** to view the detailed configurations.

What to do next

Obtain switch value

In the mPaaS Console, add necessary switch configuration items in **Mobile Delivery Service** > **Configuration management**, and deliver the configuration items by platform, whitelist, percentage, version, device model, iOS version and other information. For detailed operations, see Manage configurations.

When the switch key is released through the console, the client can get the key value corresponding to the switch key by calling relevant interface.

```
+ (void)testStringForKey
{

    id<APConfigService>configService = [DTContextGet() findServiceByName:@"APConfigService"];
    NSString *configValue = [configService stringValueForKey:@"BillEntrance"];
    assert (configValue && [configValue isKindOfClass:[NSString class]]);
}
```
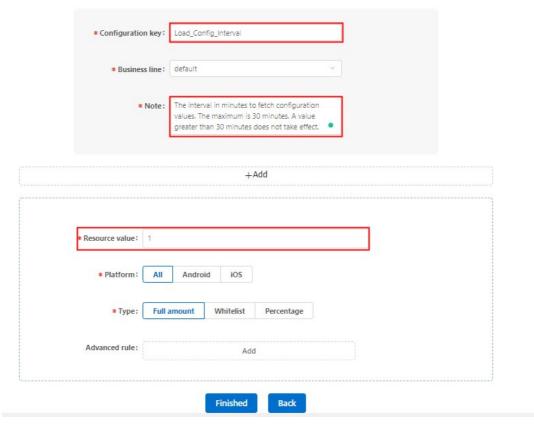
Note: Switch key values are obtained by calling the RPC API. The call to the RPC API is not necessarily successful. Therefore, you must consider the local processing logic on the client to deal with failures in obtaining key values. We recommend that you set a default switch value in the local logic on the client. You can enable the client to use a new configuration logic when the console delivers a new switch value and use the local default logic when the client fails to obtain a key value.

Advanced operations

- Set the time when the client pulls a switch value:
  - During cold start of the application
  - 30 minutes later than last pull after the application returns to the foreground

> ⓘ **Note**
>
> The interval of 30 minutes is the default value. You can modify the interval by adding switch `Load_Config_Interval` on the **Configuration Switch Management** page under **Mobile Delivery Service** in the mPaaS console. For more information about the steps, see Manage configurations.

- Dynamically monitor switch value changes

○ You can add an observer for a specified key to dynamically monitor switch value changes.

○ When the client pulls switch configurations, you can obtain the latest switch value of the specified key in the callback method.

• You can forcibly pull a switch value from the console by using an SDK.
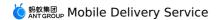

6.3. Manage configurations

As a developer, you can add necessary switch configuration items in **Mobile Delivery Service** > **Configuration management** in the mPaaS Console and deliver the configuration items by platform, whitelist or percentage, version, device model, Android or iOS version and other information.

Prerequisite

Install the switch configuration service SDK on the Android or iOS client.

Add configuration items

You can add configuration items one by one, or import a JSON file to add the configuration items in batches.

The switch configuration list shows information including the configuration keys, creation time, update time, creator, and modifier. The newly created configuration item is activated by default.

Add a configuration item

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

1. From the navigation bar on the left, click **Mobile Delivery service** > **Configuration management**.

2. Click **Add configuration**, add **Configuration key**, **Resource value**, and choose **Platform** and **Type**. The resource value is the value of the configuration key. A configuration key may have multiple resource values. Click the **Add** button to set multiple resource values.

3. (Optional) Click **Add** on the right of **Advanced rule**, and set version, osVersion, device model, and city to deliver the configuration selectively.

4. Click **Finished** after you complete the configurations, and the items will appear in the configuration list.

Batch import

On the **Configuration Switch Management** page, click **More operations** > **Import file**, and import a JSON file containing configuration items. The configuration items will appear in the list after import succeeded.

> ⊙ **Important**
>
> If the configuration item in the file already exists in the file, the item will not be imported.

In addition to batch import function, MDS also supports exporting the configuration items. On the **Configuration Switch Management** page, click **More Operations**> **Configuration Export** to download the JSON configuration file to the local.

Search for a configuration item

On the **Configuration switch management** page, enter a keyword to search for the configuration item.

Modify a configuration item

Complete the following steps to modify a configuration item:

1. On the **Configuration switch management** page, select the target configuration item and click **Modify**.

2. Edit the **Note**, **Resource value**, **Platform**, **Type** or **Advanced rule** field based on your needs, and click **Finished**. The configuration key cannot be modified.

Activate/Deactivate a configuration item

The newly created configuration item is activated by default. If you don't need a configuration, you can click **Disable** to deactivate the configuration item. Likewise, for disabled configuration items, click **Activate** to make them take effect again.

What to do next

After the switch key is released through the console, the client can obtain the key value corresponding to the switch key by calling interfaces:

- Android client

- iOS client

7.Manage whitelists

Whitelist management is a basic function of Mobile Delivery Service, which provides a whitelist management platform where you can easily create hundreds of thousands of whitelist data for the use by real-time release.

On the whitelist management page, you can perform the following operations:

- Create a whitelist
- Add user information to a whitelist
- Delete a whitelist

Create a whitelist

1. Log in to the mPaaS console, and click **Mobile Delivery Service** > **Whitelist management** on the left navigation bar.
2. On the Whitelists page, click **Add a whitelist**, and enter a whitelist name, select a whitelist type and click **OK** in the dialog box to create a whitelist.
   - **Normal**: Contents of the whitelist are user IDs. The whitelist is hit only when there is an exact match.
   - **Regular expression mode**: Contents of the whitelist are multiple regular expressions. The whitelist is hit if any of the regular expressions is matched.

Add user information to a whitelist

1. Click **Add** right to the target whitelist in the whitelist list, and then enter the user IDs or regular expressions to be added to the whitelist in the dialog box.
   - Add users to a whitelist of normal type

     User IDs are configured by the clients. To learn how to do so, see User ID configuration. Multiple user IDs must be separated by commas or line breaks. You can also upload a whitelist file containing user information to add whitelist users in batch.
   - Add users to a whitelist of regular expression mode

     Enter regular expressions. Separate with line breaks. Up to 20 expressions are allowed.

2. Click **OK** after you have entered the information.

Delete a whitelist

Click **Delete** right to the target whitelist in the whitelist list to delete it.

8.Manage release rules

Resource configuration management is a basic function of Mobile Delivery Service. With this function, you can predefine various configuration data required for Mobile Delivery Service and don't have to manually input the data every time, with work efficiency improved and error occurrence decreased.

The configuration data, such as city and device model, is also referred to as resources. When adding configuration data, the resource name is shown to users. Only the resource value is used to match request parameters of clients.

On the resource configuration management page, you can perform the following operations:

- Add a resource
- Modify resource configurations
- Delete a resource

Add a resource

1. Log in to the mPaaS console, and click **Mobile Delivery Service** > **Release rule management** from the navigation bar on the left to go to the resource rule management page.

2. On the release rule management page, click **Add resource**. In the pop-up window, select resource type and platform type, enter the resource name and resource value, and then click **OK** to create the resource.

   - **Resource type**: The type of the resource. City, Device Model, Network, and Device System Version are supported.

   - **Platform type**: Select a mobile platform. You can choose Android, iOS, or All.

   - **Resource name**: The name of the resource. You can define the name as needed. The resource name is shown to users and generally consistent with the resource value.

   - **Resource value**: Only one resource value is supported. The following content describes the values of various types of resources:

     - City: The name of the city at the prefecture and city level. The name must contain the administrative unit, such as the city, region, autonomous prefecture, league. For example, the name can be Shanghai Municipality, Haidong District, Qiannan Buyi and Miao Autonomous Prefecture, and Xing'an League.

     - Model: The model of the mobile device, such as VIVO X5M and iPhone 6S.

     - Network: The type of the network. Valid values: 2G, 3G, 4G, 5G, Wi-Fi, and WLAN.

     - Device OS version: The system version of the mobile device, such as 10.0.1 and 5.1.1.

   If you do not know the model, network, and device system version of the mobile device, you can query the information about the mobile client by calling the specified API. For more information, see Call API operations to query resource configurations.

Modify resource configurations

To modify resource configurations, find the resource and click **Modify** in the **Operation** column. Edit the resource configurations as required. Click **OK** to save the modification.

Delete a resource

To delete a resource, find the resource and then click **Delete** in the **Operation** column. To delete multiple resources, select multiple resources, click **Batch delete**, and then click **OK**.

Call API operations to query resource configurations

When you add a resource, if you do not know the resource values of the network, device model, and device system version, you can call the API to query configurations of the resource.

Perform the following steps:

1. Open a local project and call the following API to obtain the information of the mobile client:

   - Android clients

```
DeviceInfo deviceInfo = DeviceInfo.createInstance(context);
 AppInfo appInfo = AppInfo.createInstance(context);

 deviceInfo.getOsVersion(); //The version of the device system.
 deviceInfo.getmMobileModel(); //The model of the device.
 appInfo.getmProductVersion(); //The version of the device.

 int networkType = NetworkUtils.getNetworkType(context);//The network type of the device.
 networkType = 1 (2G)
 networkType = 2 (3G)
 networkType = 3 (Wi-Fi)
 networkType = 4 (4G)
```

- iOS clients

Type	Network	Device system version (system API)	Device model (mPaaS-encapsulated API)
Switch configuration	None	[[UIDevice currentDevice] systemVersion]	- Baseline version before 10.1.68.32: [APMobileIdentifier shareIdentifier].deviceModel - Baseline version 10.1.68.32 and later: [MPaaSDVInfo sharedInstance].deviceModel
Upgrade detection	Wireless networks: Wi-Fi Mobile networks: WWAN	[[UIDevice currentDevice] systemVersion]	- Baseline version before 10.1.68.32: [APMobileIdentifier shareIdentifier].deviceModel - Baseline version 10.1.68.32 and later: [MPaaSDVInfo sharedInstance].deviceModel
Hotfix management Offline package management Mini Program management	[DTReachability networkName]	[[UIDevice currentDevice] systemVersion]	- Baseline version before 10.1.68.32: [APMobileIdentifier shareIdentifier].deviceModel - Baseline version 10.1.68.32 and later: [MPaaSDVInfo sharedInstance].deviceModel

2. Report the client resource information to the server by uploading the log, and then view the resource configuration information on the server.

9.Reference

9.1. API

Learn about how to use the relevant APIs of Android update SDK.

Learn about how to use the relevant APIs of Android update SDK.

- MPaaSCheckVersionService API
- MPaaSCheckCallBack API

MPaaSCheckVersionService API

checkNewVersion

Check if a new version is available. This method starts an asynchronous task to check the updates and calls the relevant callback method of `MPaaSCheckCallBack` whether or not a new version is available.

```
void checkNewVersion(Activity activity)
```

setIntervalTime

Set the interval of reminder:

```
void setIntervalTime(long interval202)
```

3 days by default, in milliseconds.

setMPaasCheckCallBack

An example of the callback to be called when setting the update SDK for checking updates:

```
void setMPaaSCheckCallBack(MPaaSCheckCallBack mPaaSCheckCallBack)
```

installApk

To install the package of the new version, in `MPaaSCheckCallBack.alreadyDownloaded` method, you can call:

```
void installApk(String filePath)
void installApk(ClientUpgradeRes res)
```

update

To download the package of the new version, in `MPaaSCheckCallBack.showUpgradeDialog` method, you can call:

```
void update(ClientUpgradeRes res)
```

MPaaSCheckCallBack API

startCheck

Called after calling the update checking interface. In this method, you can prompt the users that the checking is in loading:

```
void startCheck()
```

isUpdating

Called when the update checking interface is repeatedly called:

```
void isUpdating()
```

onException

Called when exceptions occur in update checking:

```
void onException(Throwable throwable)
```

dealDataInValid

Called if the returned update information is valid:

```
void dealDataInValid(Activity activity, ClientUpgradeRes result)
```

dealHasNoNewVersion

Called if the returned update information is invalid:

```
void dealHasNoNewVersion(Activity activity, ClientUpgradeRes result)
```

alreadyDownloaded

Called if the new version package has already been downloaded: You can prompt users to install this package at this time. If users choose to install, then `MPaaSCheckVersionService.installApk` method is called for installation:

```
void alreadyDownloaded(Activity activity, ClientUpgradeRes result)
```

showUpgradeDialog

Called when a new version is available, but the package is not downloaded. You can prompt and ask users whether to update, if users choose to update, then `MPaaSCheckVersionService.update` method is called for triggering download:

```
void showUpgradeDialog(Activity activity, ClientUpgradeRes result)
```

onLimit

Called when a new version is available, but the time from the last checking is less than the set interval. It is valid only when the configuration is **Single reminder**.

```
void onLimit(Activity activity, ClientUpgradeRes result, String reason)
```

9.2. Code sample

9.2.1. Version update code sample

Android code sample

To check the style and interaction effect of this function in mobile device, download Android code sample, then compile bundle in local Android Studio and install `.apk` file in your mobile device. See Get code sample for more information.

iOS code sample

Check for updates

mPaaS automatically connect the release function by calling update check interface to check whether a new version is available. If a new version is available, a update window automatically pop up to remind user for update. User tap **Update** to start auto update, no other encoding is required. To custom update prompt window, see UI of custom update prompt below.

```
- (void)checkUpdate
{
    UpgradeCheckService *service = [UpgradeCheckService sharedService];
    service.delegate = self;
    [service checkUpgradeAndShowAlertWith:YES];
}
```

> ⑦ **Note**
>
> Note: When you add SDK, the dependency on release service gateway mPaaS > Targets > MPHttpClient > DTRpcInterface+upgradeComp.m is automatically added, thus you only need to call checkUpgradeAndShowAlertWith method, the release component automatically connect the release service in background.

UI of custom update prompt

You can custom the update prompt by implementing delegate.

```
# pragma mark UpgradeViewDelegate
- (UIImage *)upgradeViewHeader
{
    return [UIImage imageNamed:@"FinancialCloud"];
}
- (void)showProgressHUD:(BOOL)animation
{
    self.toast = [APToastView presentToastWithin:self.view withIcon:APToastIconLoading text:nil];
}
- (void)hideProgressHUD:(BOOL)animation
{
    [self.toast dismissToast];
}

- (void)showToastViewWith:(NSString *)message duration:(NSTimeInterval)timeInterval
{
    [self showAlert:message];
}
```

9.2.2. Hotpatch Code Sample

Android code sample

mPaaS framework based

See Get Code Sample to obtain code sample.

Native framework based

Demo address

See Get Code Sample to obtain code sample.

9.2.3. Switch configuration code sample

Download the code sample corresponding to your client type:

Download the code sample corresponding to your client type:

- iOS: Switch configuration code sample (For accessing using Cocoapods)
- Android: Switch configuration code sample (For accessing through mPaaS Inside and Native AAR)

For more information, refer to iOS Code Sample and Android Code Sample.