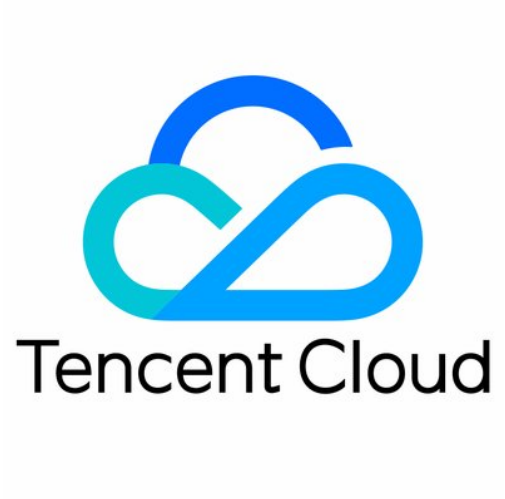
Chat

Push Feature

Product Documentation



Contents

Push Feature
Push Function Introduction

Last updated：2024-06-13 10:21:45

Overview

Instant Messaging provides you with two integration methods: TIMPush (recommended) and Self-integrated Push. Both integration methods support Xiaomi, Huawei, Honor, OPPO, vivo, Meizu, APNs, OnePlus, realme, iQOO, FCM and other manufacturer channels, but they are different in access integration, push type, data statistics, link tracking, etc., see the table below for details:

Comparison Items		TIMPush	Self-integrated Push
Price		299 USD/month	Free
Supported Platforms	Android & iOS	✓	✓
	uni-app	✓	×
	Flutter	✓	×
	React Native	✓	×
Access Integration	Manufacturer SDK Integration	One-click integration	Need to integrate 7 Android vendors and 1 iOS vendor one by one, a total of 8 SDKs
	SDK Local Deployment	One-click configuration	Need to configure 8 manufacturers one by one
	Push registration, token reporting	✓	Need to develop reporting logic by yourself
	Access test tool	✓	✓
	Access cycle	1 hour	1 week
Push Type	Ordinary Message Push	✓	✓
	All/Tag Users Push：On-site push	✓	-

	All/Tag Users Push：Off-site push	✓	-
Statistics	Ordinary Message Push Record Query	✓	-
	All/Tag Users Push Record Query	✓	-
	Ordinary Message Push Data Statistics （Actual delivery rate, reach rate, click rate, etc.）	✓	-
	All/Tag Users Push Data Statistics （Actual delivery rate, reach rate, click rate, etc.）	✓	-
Link Tracking	Push Channel Query	✓	-
	Push Device Status Query	✓	-
	Push Full Link Status Query (including IM server > manufacturer server > terminal device > the entire link clicked by the user)	✓	-
Offline Push Reach		Active users within 30 days	Active users within 7 days

Contact us

If you encounter problems during use, you can solve them by checking the FAQ, or you can enter the communication group for consultation: Telegram communication group: click to join.
WhatsApp communication group: click to join.

Plugin Integration Push (Recommend) Overview

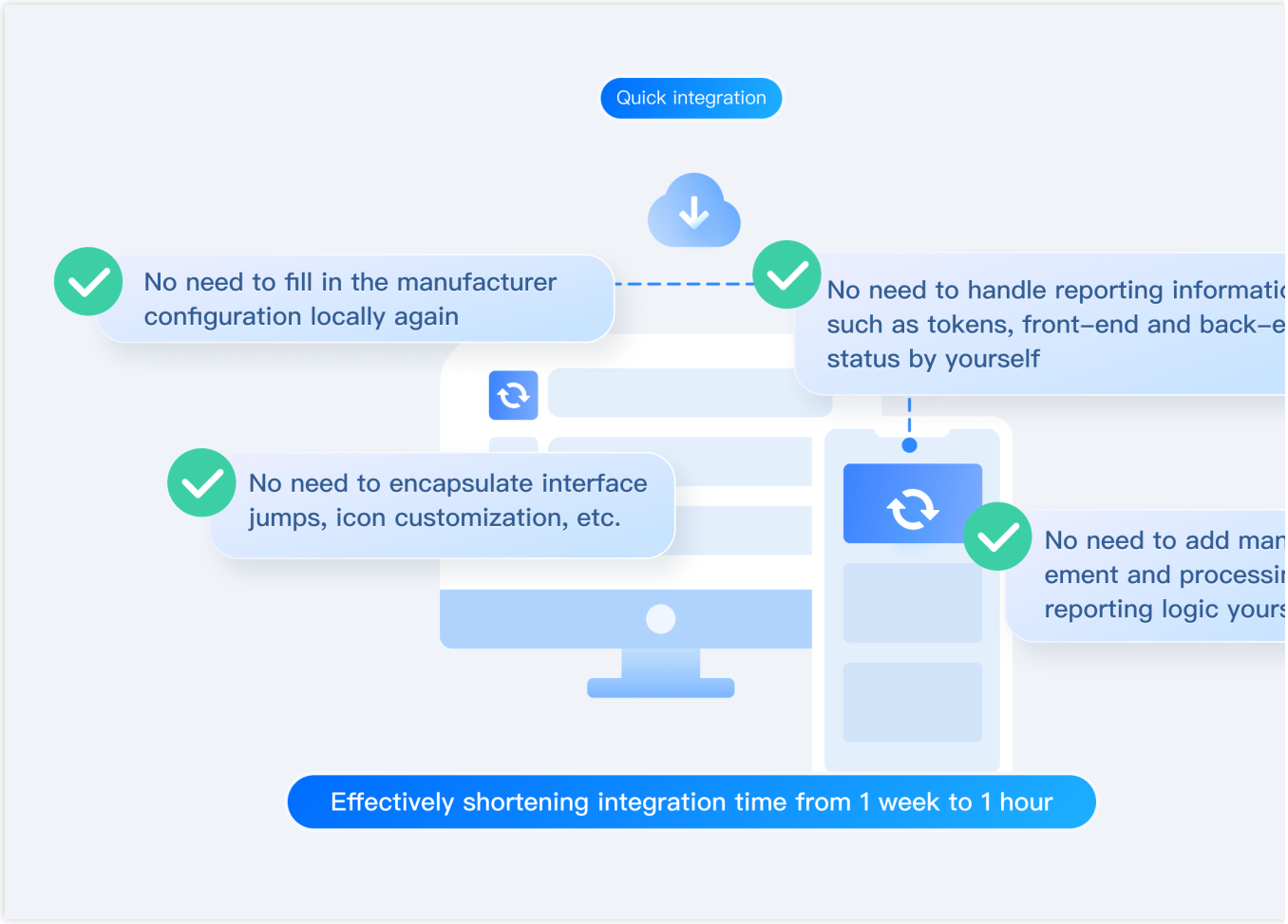
Last updated：2024-06-13 10:21:45

TIMPush provides you with stable, timely and diversified push services. Compared with self-integrated push, TIMPush only requires simple configuration and can integrate and access push services from multiple manufacturers with one click. TIMPush supports ordinary message push and all/tag users push, and provides complete push life cycle query, data statistics, and problem troubleshooting services.
If you have chatting, audio and video calls, signaling and other scenarios and need to be able to reach them in time even in offline scenarios, you can pay attention to the **ordinary message push function**.
If you have marketing advertisements, notifications, news information, etc. that need to be pushed to all users or designated groups, you can pay attention to the **all/tag users push function**.
Offline push manufacturers support Xiaomi, Huawei, Honor, OPPO, vivo, Meizu, APNs, including sub-brands of various manufacturers such as OnePlus, realme, iQOO, etc., and support Google FCM overseas.
The push management console provides you with **full-link troubleshooting tools**, **push records**, and **statistical data on various indicators**, making it easy for you to view various indicators such as push reach rate, click-through rate, and conversion rate.

Features

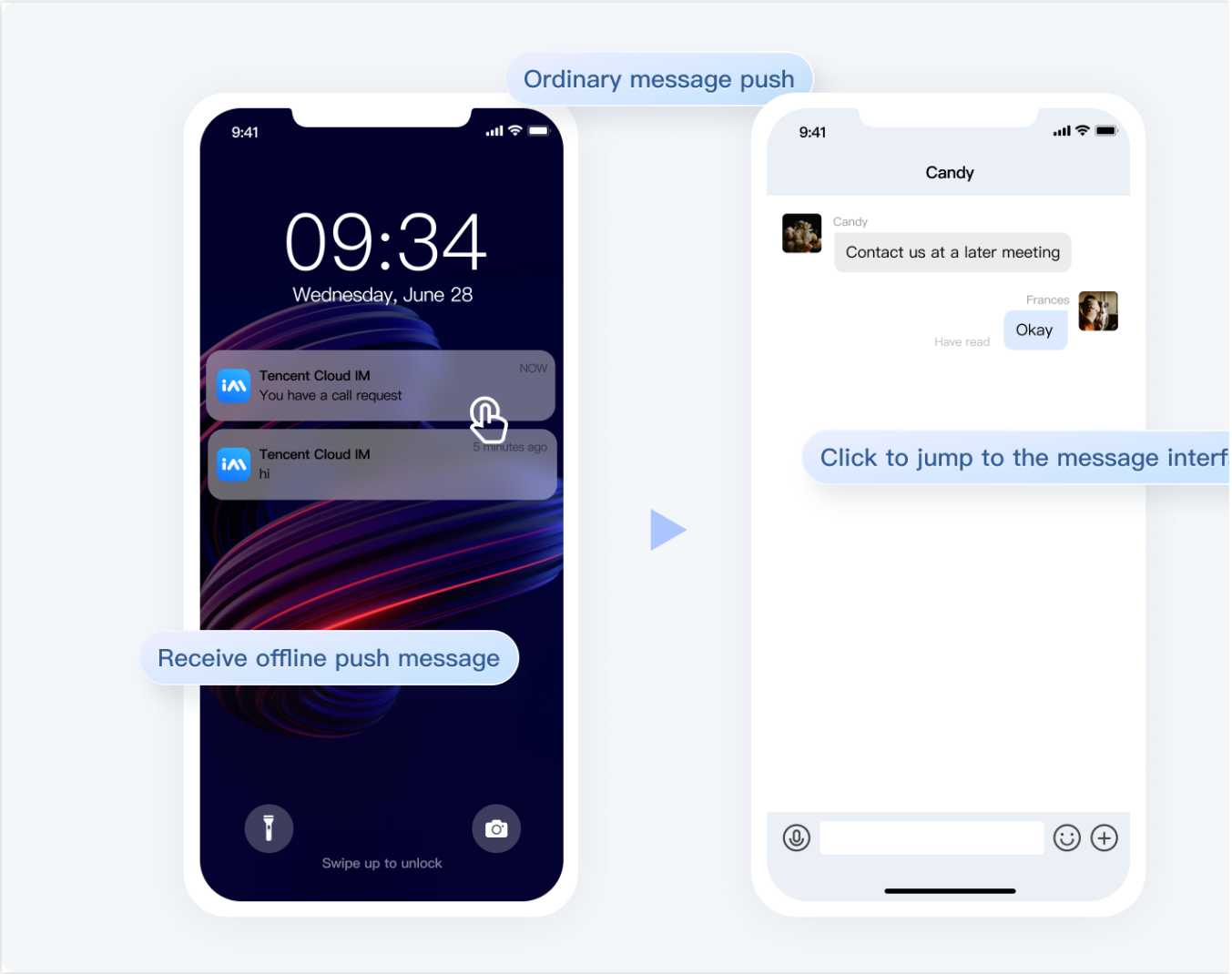
Quick integration in 3 minutes

It is no longer necessary to configure push information for each manufacturer separately. You only need to download and import the json configuration file in the IM console, and you can complete the push information configuration for all mobile phone manufacturers with one click.
Supports the integration of one or more push channel packages from corresponding manufacturers on demand to easily cope with compliance requirements.
There is no need to handle push registration, token reporting, front-end and back-end status reporting, etc. by yourself. The push plug-in is self-closed.
There is no need to add management and processing reporting logic by yourself. The push plug-in reports and summarizes by itself. It also supports link troubleshooting and indicator statistics.
The plug-in encapsulates interface jumps, icon customization and other methods, and can be used directly.



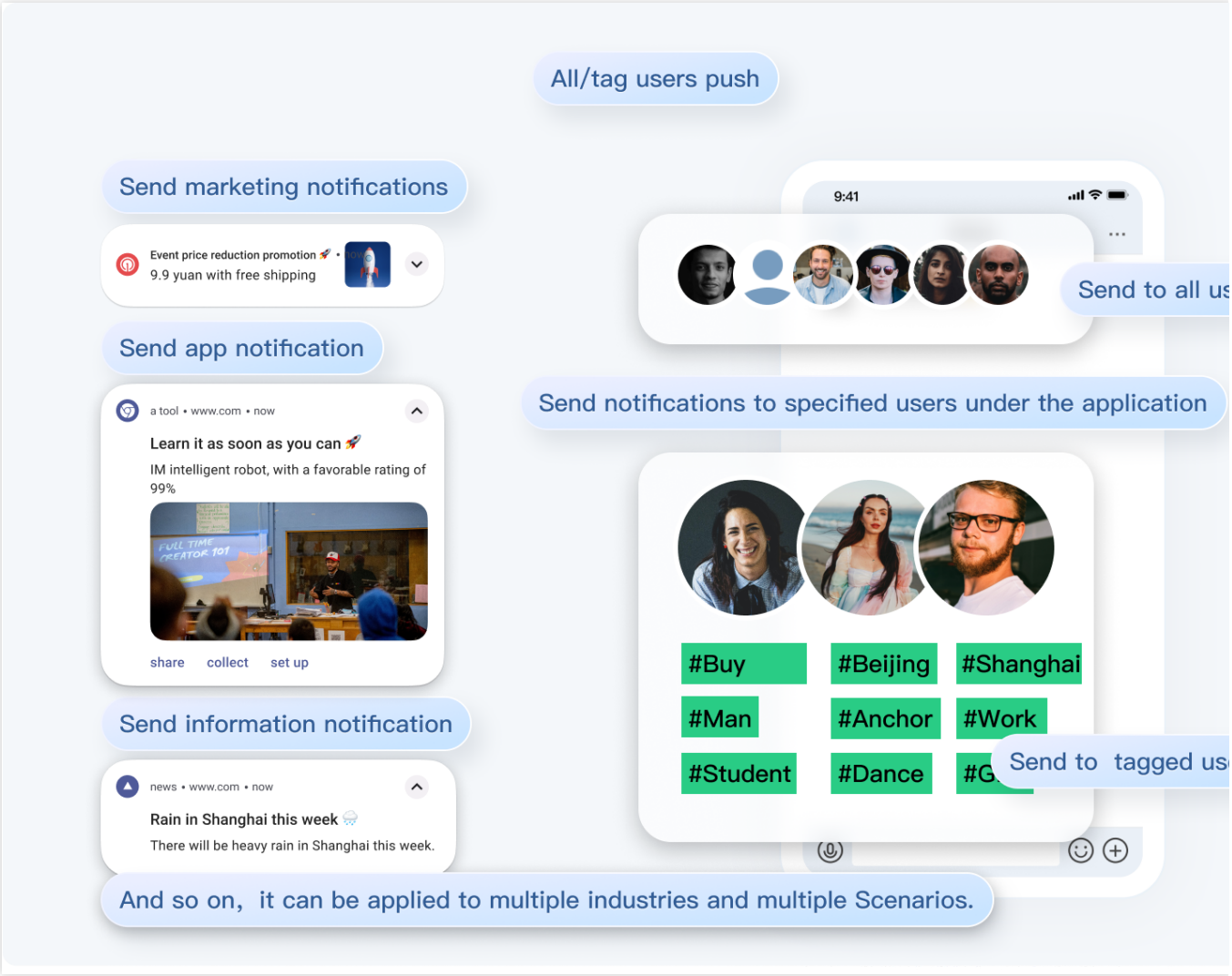
Ordinary message push

In ordinary IM message sending and receiving scenarios, messages can reach the device in time even when the application is offline and support customizable jump interfaces.



All/tag users push

The all/tag users push function is designed to help you push marketing, advertising, notifications, etc. to all users or tag users, ensuring that the right messages are sent to the right users in the right way at the right time.



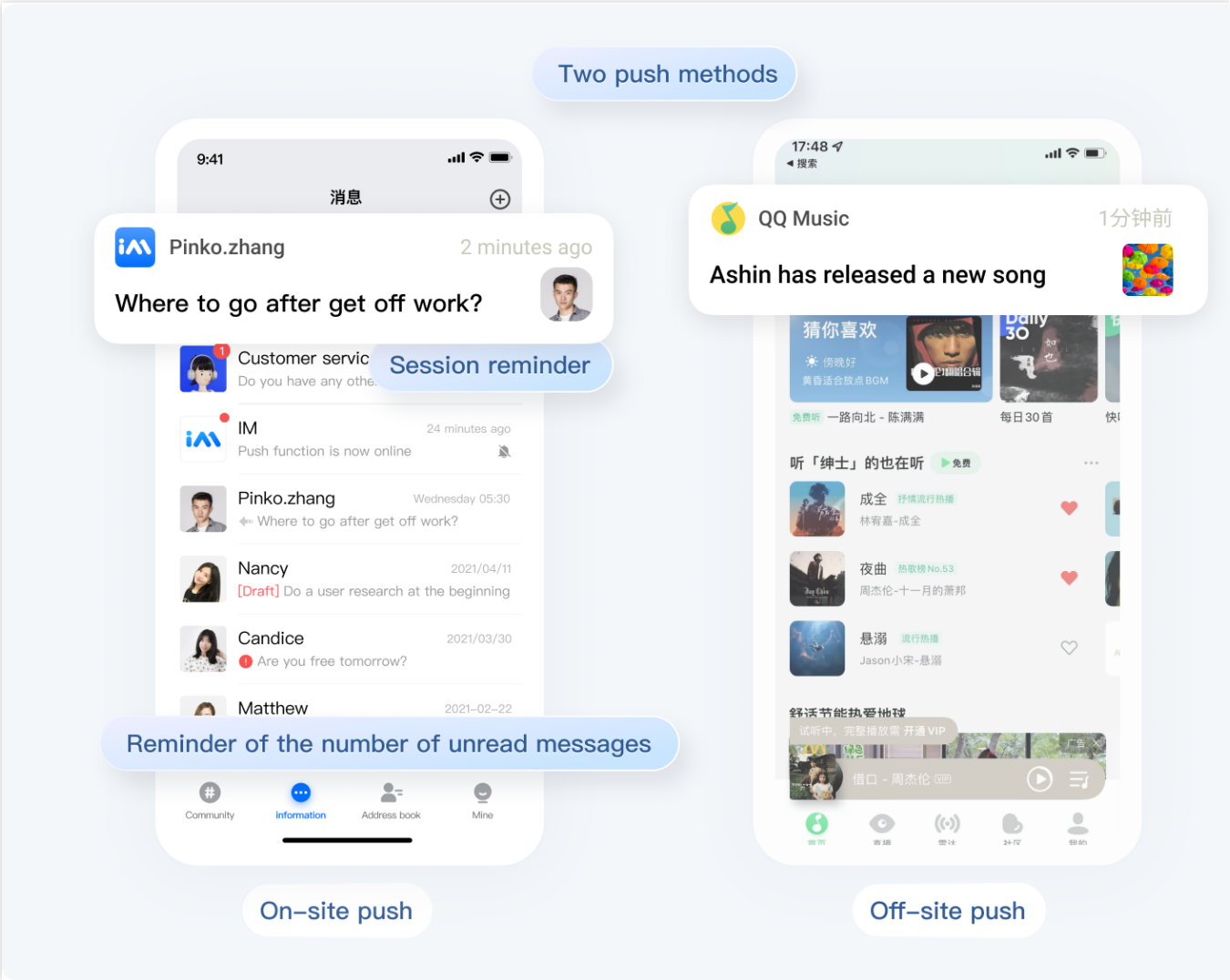
All/tag users push supports two push methods: on-site and off-site push:

On-site push

Supports message roaming. Push messages will also enter the IM message system, which will trigger updates of corresponding sessions, messages, and unread modules. Users can receive push messages when they are online. When users are not online, they can automatically receive push messages after logging in next time. news.

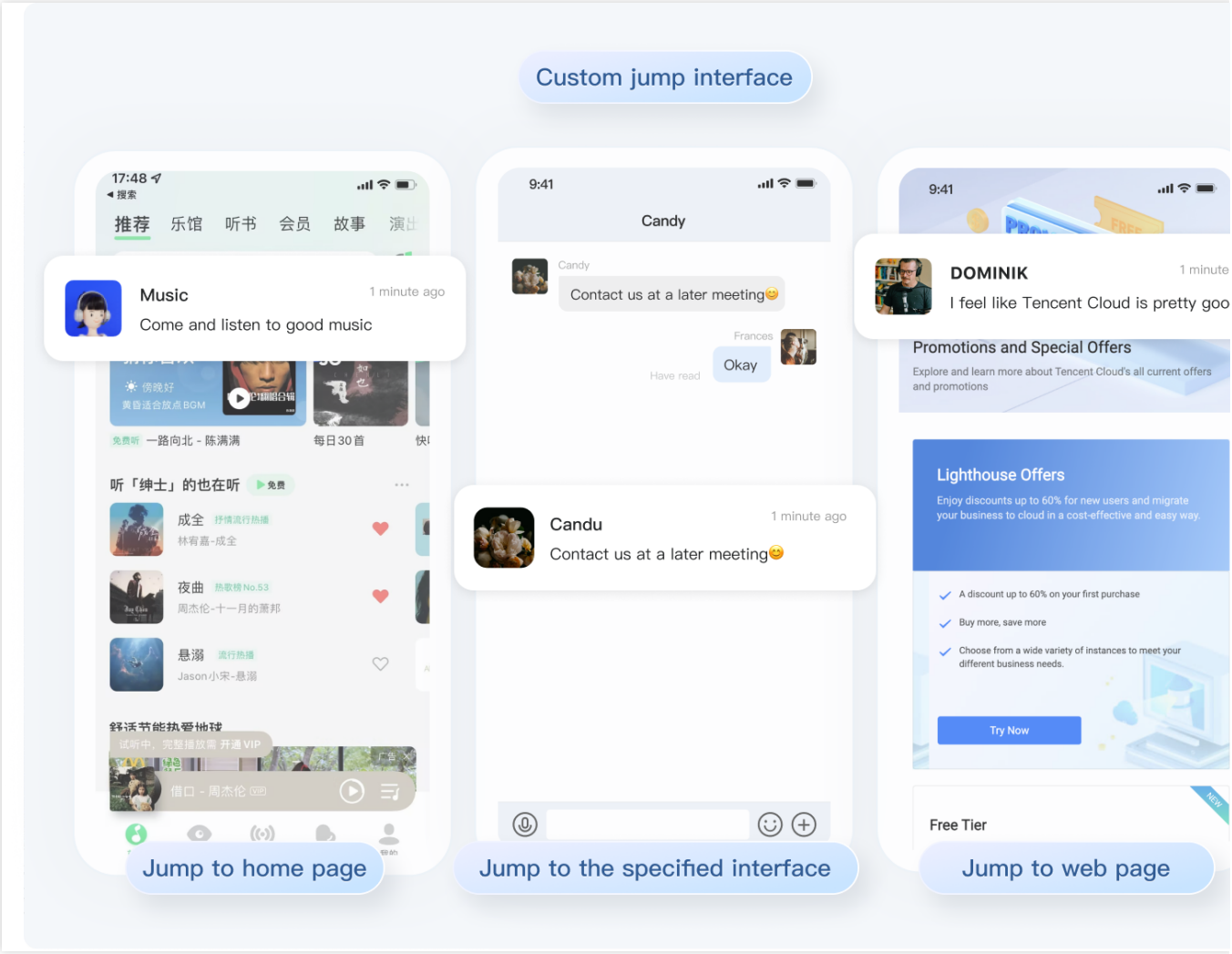
Off-site push

Push messages arrive on the device as system notifications and are not saved in the IM system.



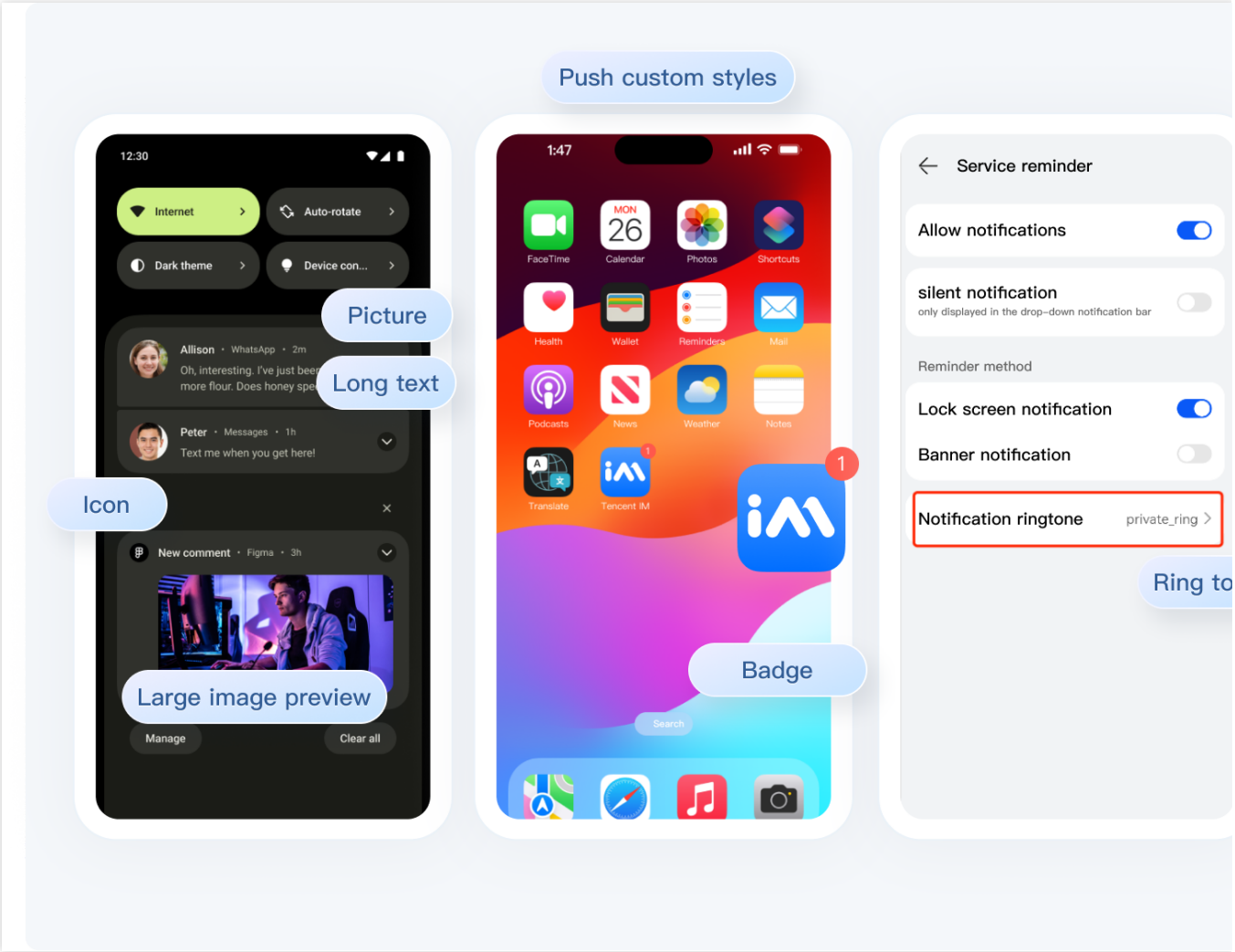
Custom jump interface

After the user receives the push, click on the notification bar to see the customized jump interface.



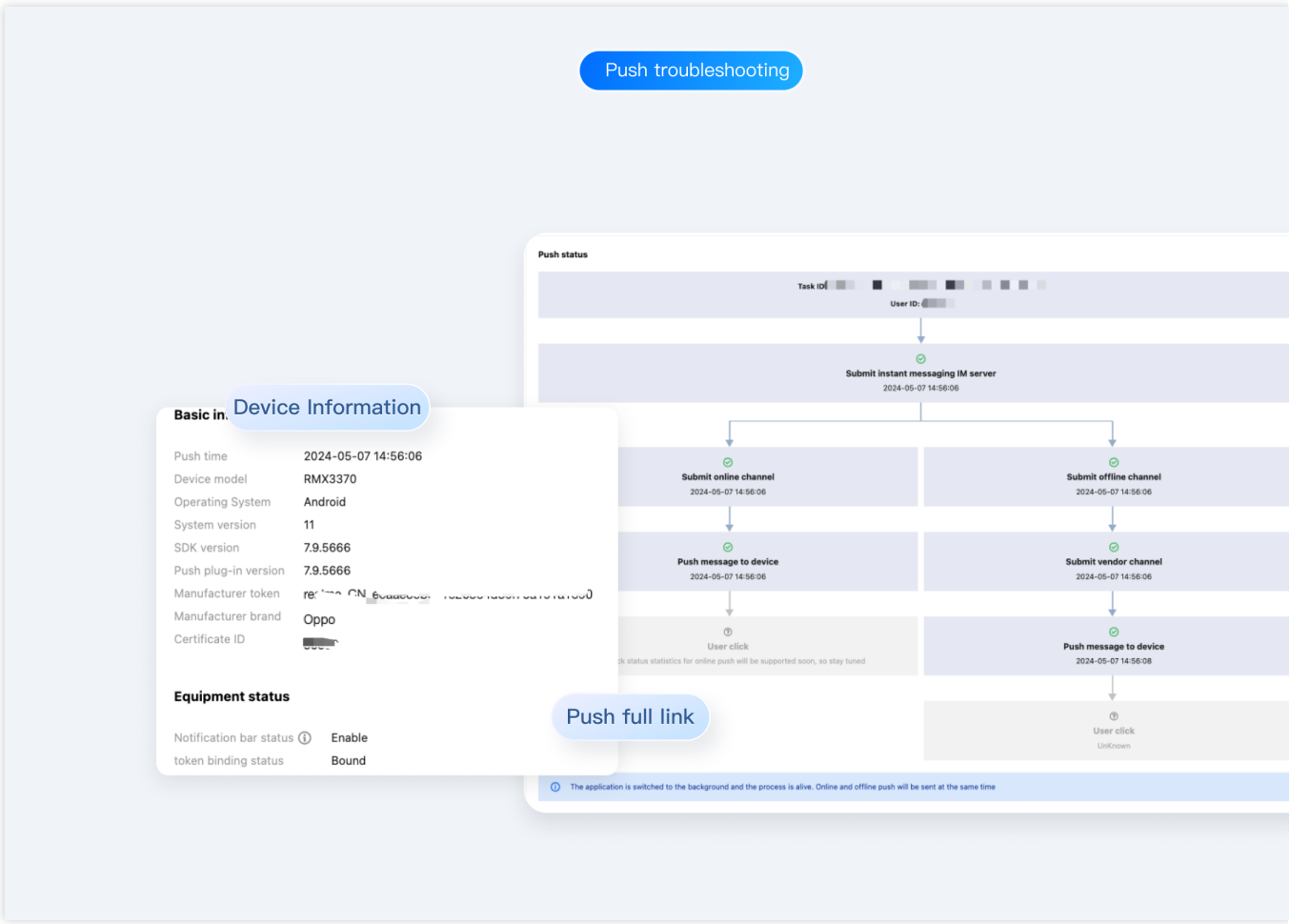
Push custom styles

Supports custom styles of small icons, right icons, long text, large pictures, badge and ringtones.



Full-link troubleshooting tools

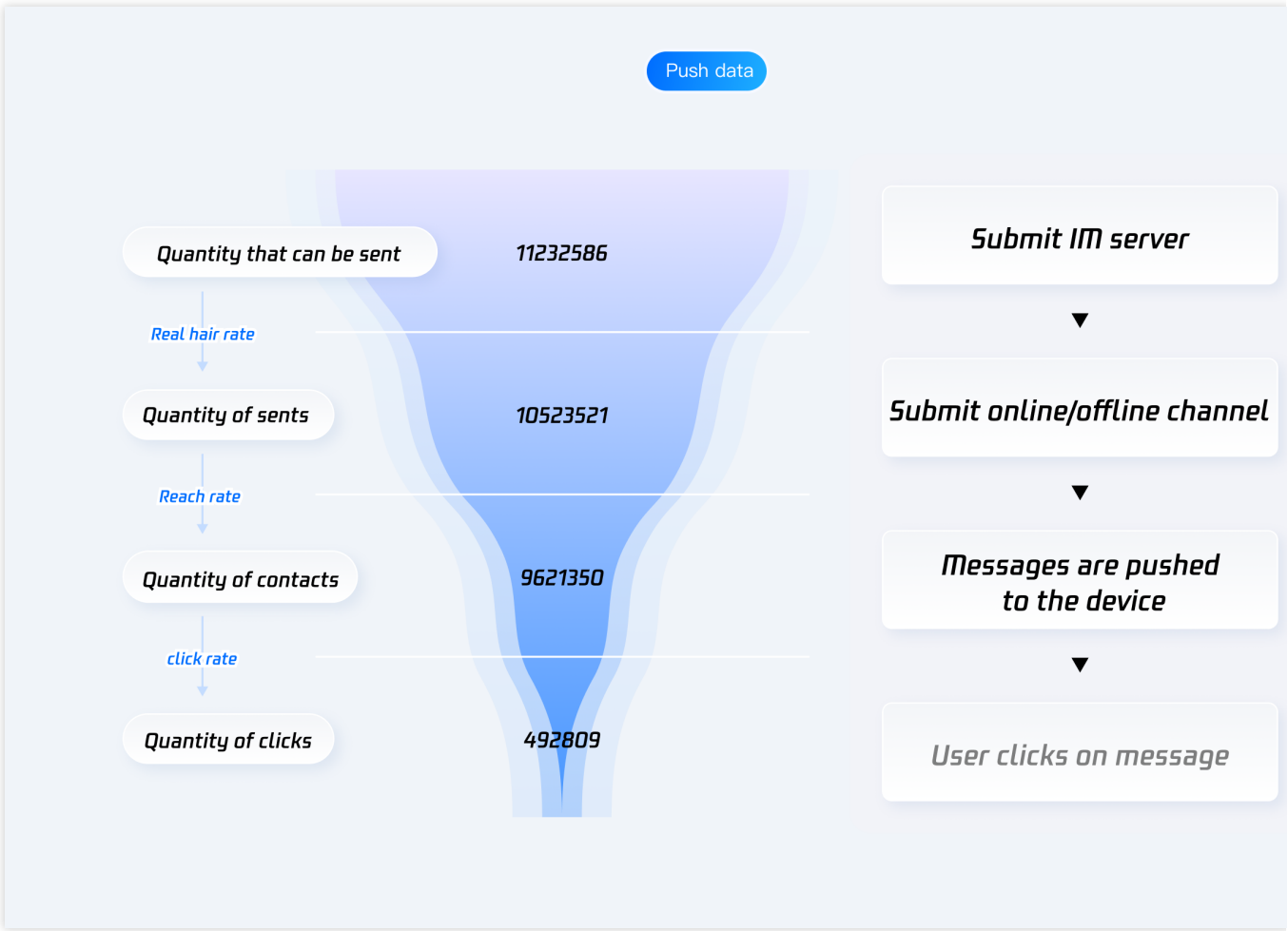
Provide self-service troubleshooting tools to view the entire push link details, analyze push failure and click failure reasons, and improve conversions.



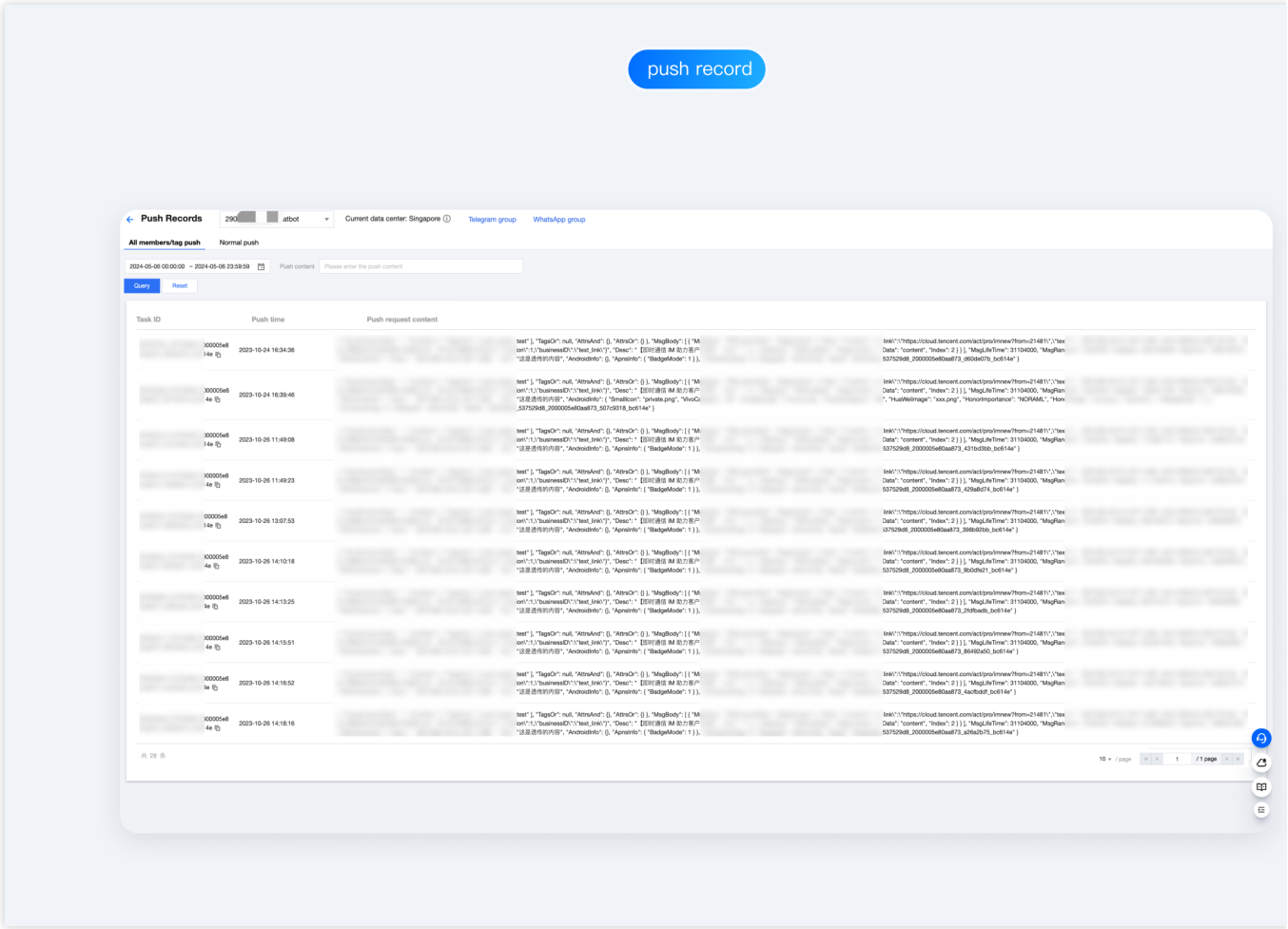
Statistical data

Record and query all push data of users, organize and analyze various types of indicator data pushed by users every day, and generate recent delivery > reach > click funnel conversion charts, which can be viewed according to manufacturer channel classification.

Push indicator analysis



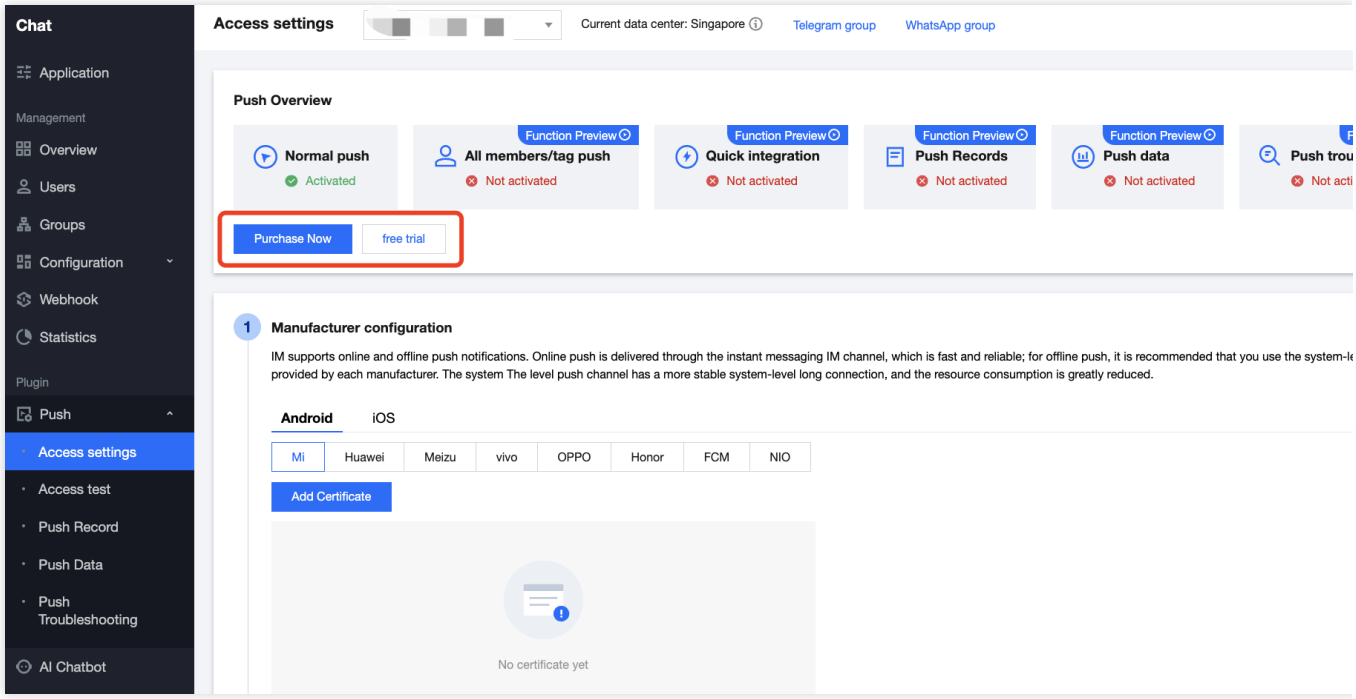
Push record



Activate

Step 1: Activate the TIMPush

Go to IM Console > Push, click Buy Now or Free Trial . (Each application can be tried once for free, valid for 7 days)



Notice：

After the trial or purchase of the TIMPush expires, **push services (including offline push of ordinary messages, all/tag users push, etc.) will be automatically stopped.** To avoid affecting the normal use of your business, please purchase/renew in advance.

Step 2: Integrate and use TIMPush

Vendor configuration Android & iOS & Flutter & uniapp。
Quickly access Android & iOS & Flutter & uniapp。

Step 3: Send all/tag users push

For details, please see push to all/tags.

Step 4: Push diverse implementations

For details, please see Advanced Features.

Step 5: Push Management and Analysis

Contact us

If you encounter problems during use, you can solve them by checking the FAQ, or you can enter the communication group for consultation: Telegram communication group: click to join.
WhatsApp communication group: click to join.

Manufacturer Configuration
Android

Last updated：2024-06-13 10:21:45

Operation step

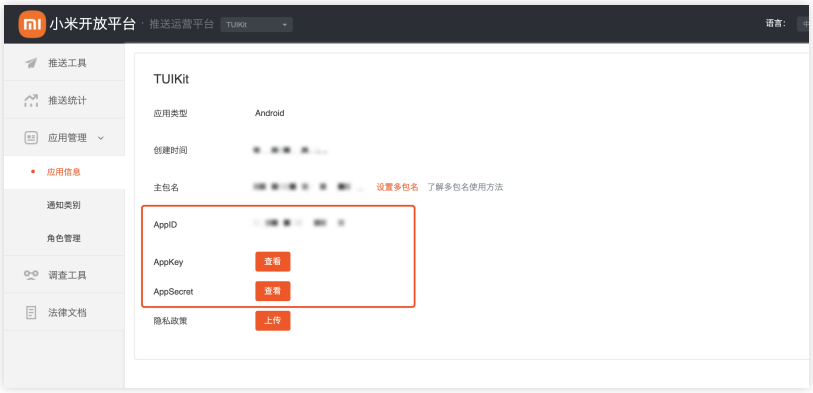
Step 1. Register your app with vendor push platforms

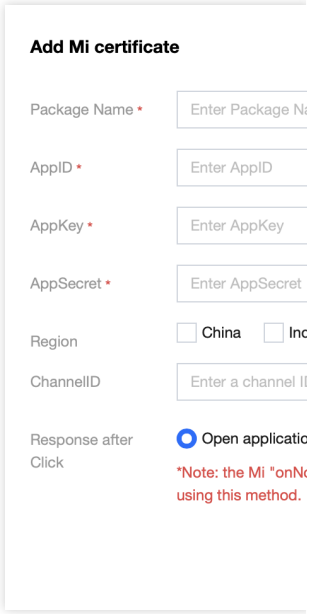
To utilize the offline push feature, you need to register your app on each vendor's push platform to obtain parameters such as AppID and AppKey. Currently, the mobile manufacturers supported in China include: Mi, Huawei, Honor, OPPO, Vivo, Meizu, and internationally Google FCM is supported.

Step 2. Create resources in the IM console

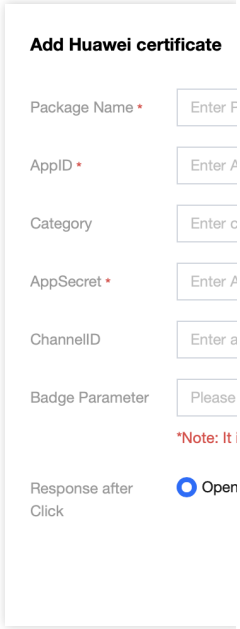
Log in to Tencent Cloud Chat Console, then in the **Push Management** > **Access Settings** feature section, add each vendor's push certificate, and configure the AppID, AppKey, AppSecret, and other parameters obtained in Step 1 to the added push certificate.
Explanation of the **Subsequent Actions** option:
Open Application: Clicking the notification bar launches the app, by default starting the app's Launcher interface;
Open Web Page: Clicking the notification bar will redirect to the configured web link;
Open the specified interface within the app: clicking the notification bar will redirect the interface based on the configured self Definition, see Custom Redirect on Click.
Mi
Huawei
OPPO
vivo
Meizu
HONOR
Google FCM

Vendor Push Platform	Configuring in the IM console



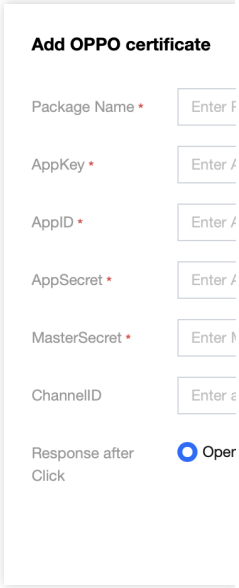


Vendor Push Platform	Configuring in the IM cons

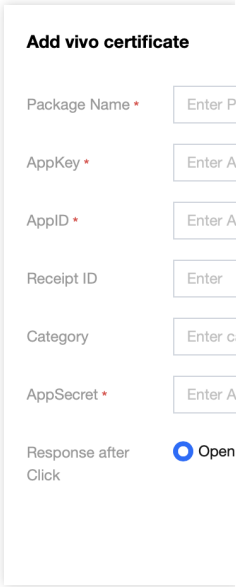


Note:
Client ID corresponds to A

Vendor Push Platform	Configuring in the IM cons

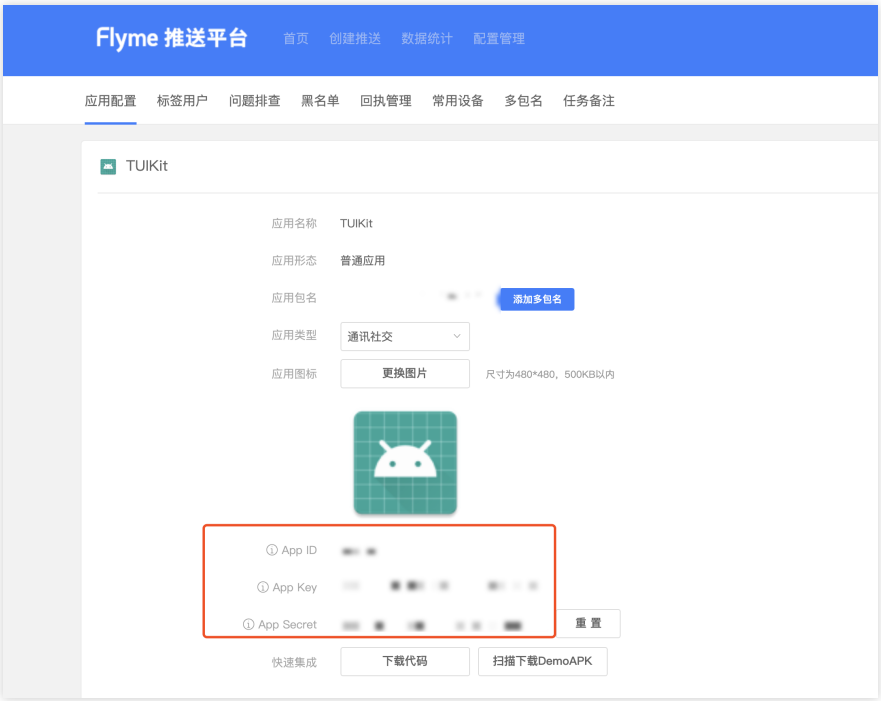


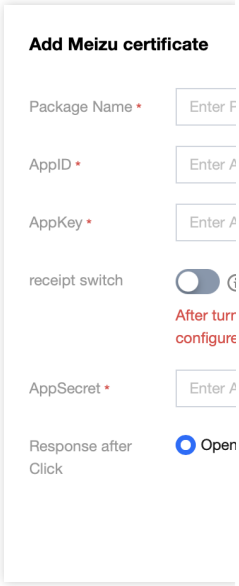
Vendor Push Platform	Configuring in the IM cons



For receipt configuration, please refer to: Message reach statistics configuration - vivo

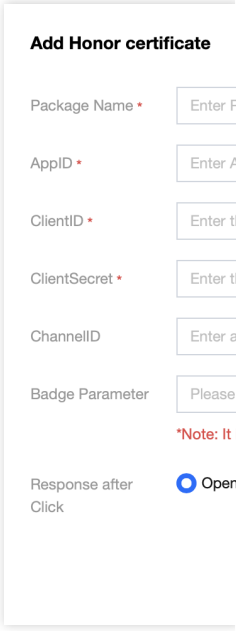
Vendor Push Platform	Configuring in the IM cons

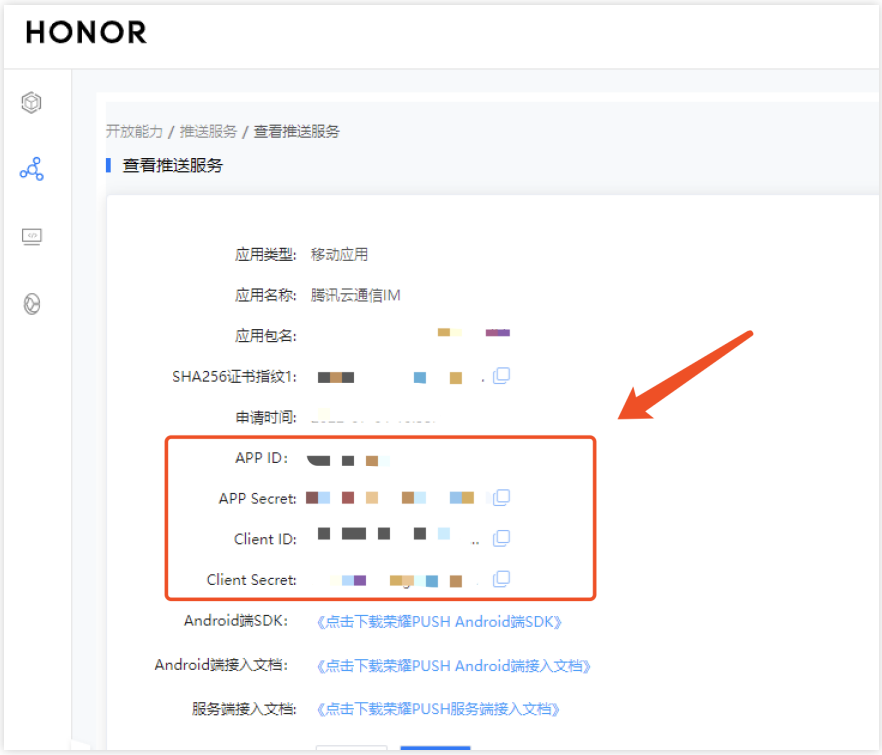




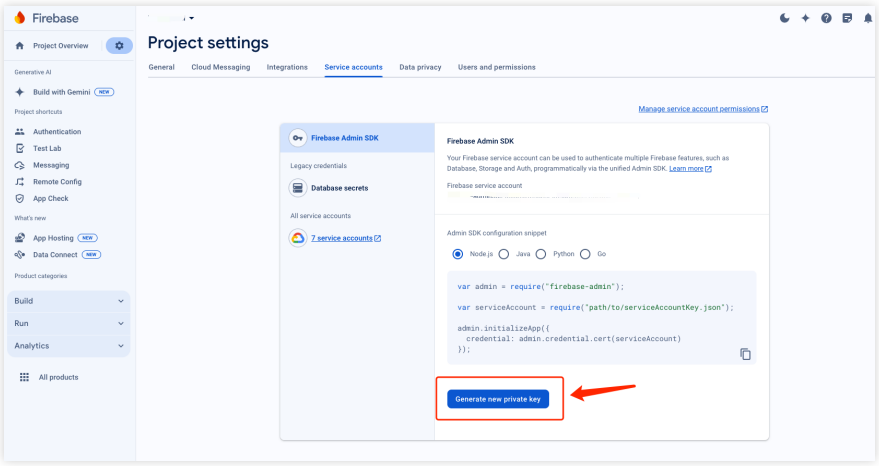
For receipt configuration, please refer to: Message reach statistics configuration - Meizu

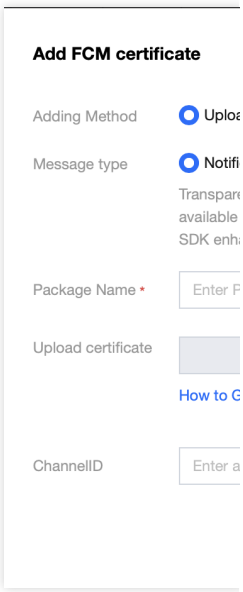
Vendor Push Platform	Configuring in the IM cons





Vendor Push Platform	Configuring in the IM cons





Note:

Regarding **Click for Subsequent Actions** supports the Report Statistics feature:

1. If you choose to open an app or a web page, purchasing the plugin will by default support reporting statistics.

2. If you choose to open a specified interface within the application:

For new certificate status, please directly use the auto-fill default value to support click statistics reporting.

If there was a certificate configured previously, continue using the old certificate and modify it to the default value to support reporting statistics, or regenerate a new certificate.

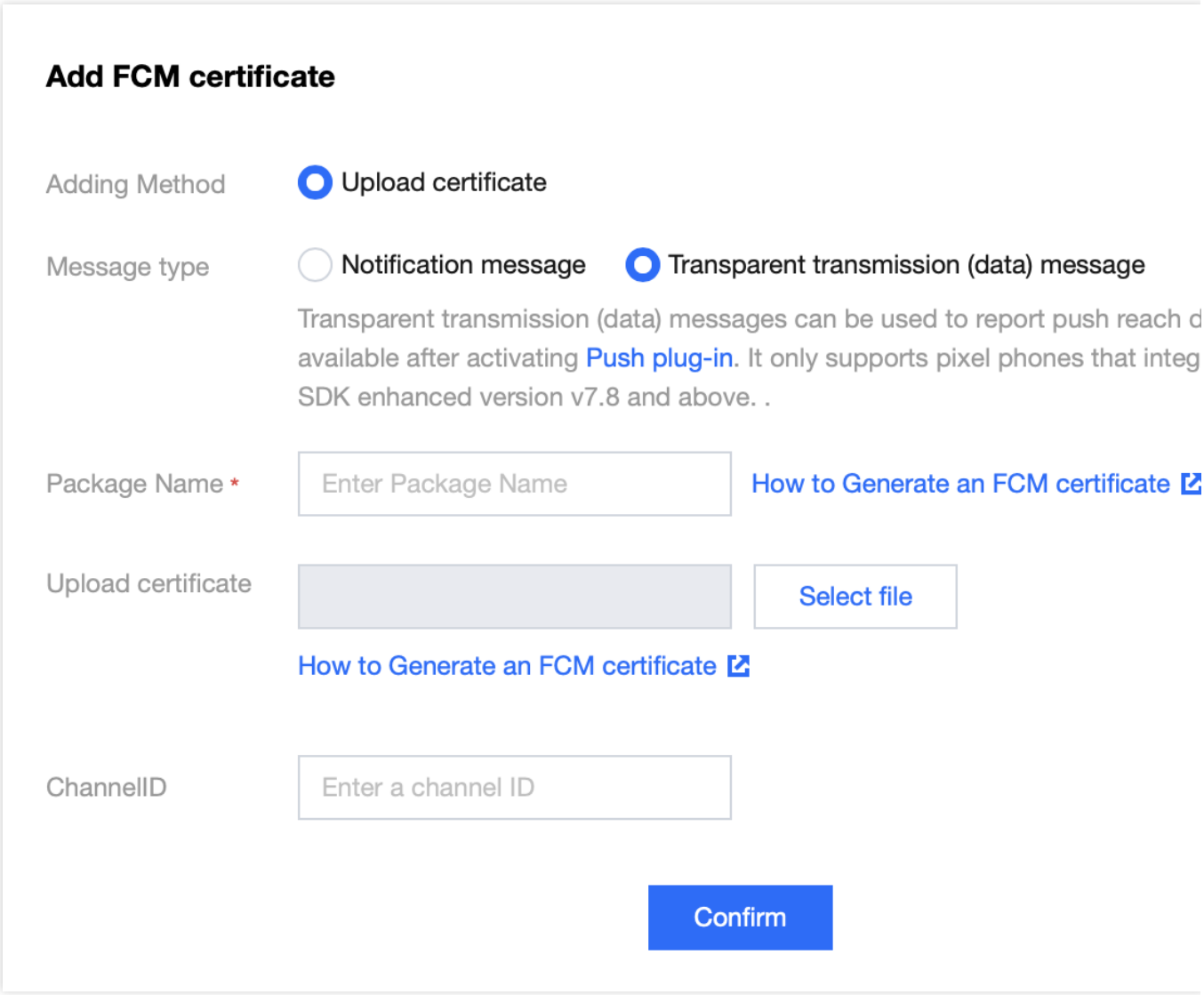
About FCM Data Messaging

FCM provides two push methods: Notification Message and Data Messaging.

Notification Message, with a simple style not differentiated by device, can support offline push once successfully integrated;

Data Messaging, offering rich customization for specific devices, supports reach and click reporting, and requires testing on the device before going live after integration.

The console defaults to Notification Message, and switching between modes can be done in the IM Console:



Note:

FCM Data Messaging capability is only supported on Pixel phones with TIMPush 7.8 and above, other manufacturers' devices need to be self-tested for support;

iOS

Last updated：2024-06-13 10:21:45

Before integrating the TIMPush component, you need to apply for an APNs Push Certificate from Apple and then upload the Push Certificate to the IM console. After that, you can proceed with the quick access steps.
There are currently two mainstream types of certificates for Apple Manufacturer Configuration: p12 certificates and p8 certificates. Each type of certificate has its advantages and drawbacks, and you can choose one according to your needs.

Certificate Type:

p12 Certificate: A p12 certificate is a binary file containing both a public key and a private key, used for certificate-based authentication. It bundles the public key certificate and the private key into one file, with extensions .p12 or .pfx.
p8 Certificate: A p8 certificate is an Auth Key, used for token-based authentication. It is a text file containing a private key, with an extension of .p8.

Validity and Management:

p12 Certificate: A p12 certificate typically has a one-year validity period, after which it needs to be regenerated and deployed. Each application requires a separate P12 certificate to handle push notifications.
p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a P8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications.

Security:

p12 Certificate: A p12 certificate uses certificate-based authentication and requires the private key to be stored on the server. This could increase security risks, as the private key could be accessed by unauthorized users.
p8 Certificate: A p8 certificate uses token-based authentication, which means your server periodically generates a JSON Web Token (JWT) to establish a connection with APNs. This method is more secure, as it doesn't require storing a private key on the server.

Dynamic Island:

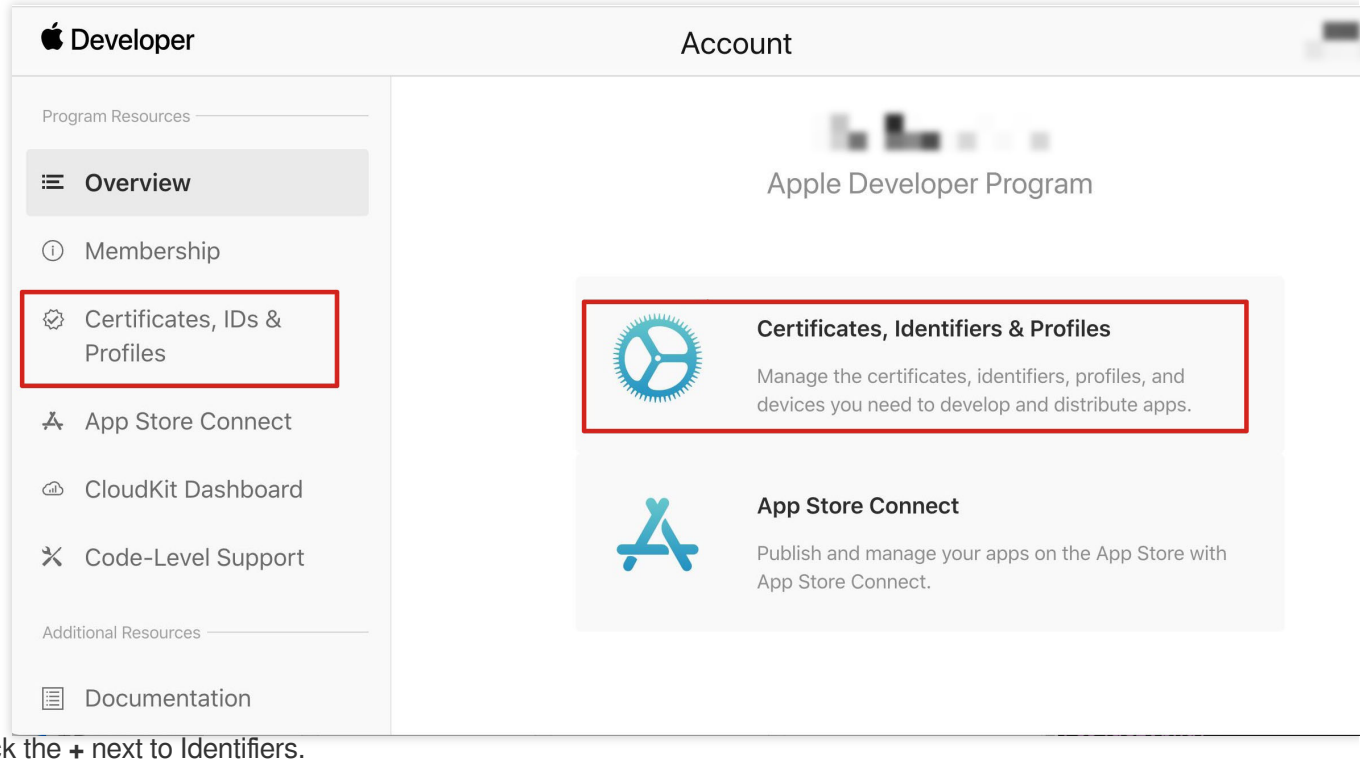
p12 Certificate: Not supported.
p8 Certificate: Supports Dynamic Island push notifications.

1. Using a p12 certificate (traditional push certificate)

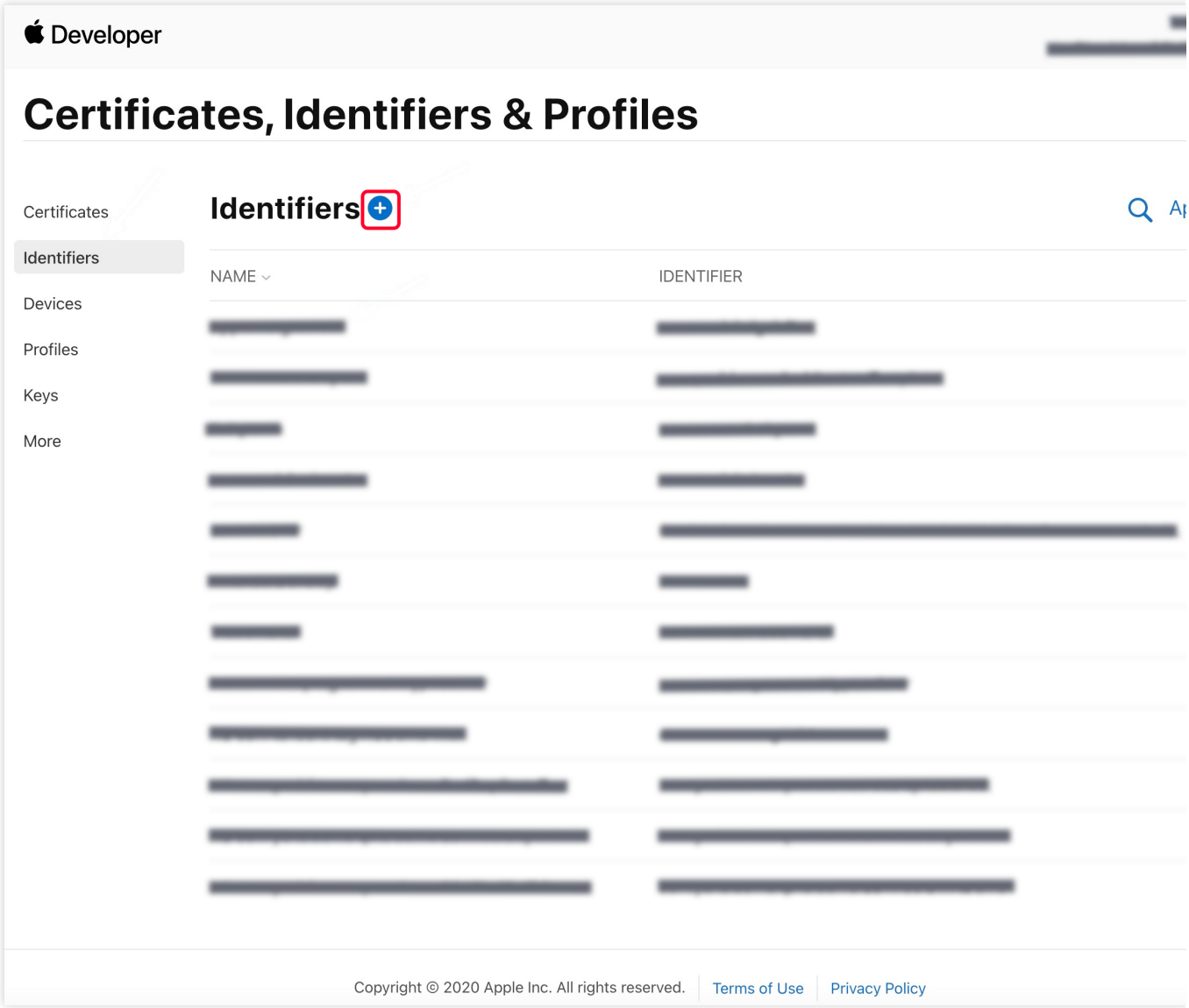
Step 1: Apply for an APNs certificate

Enable remote push for the app

1. log in to Apple Developer Center website, click **Certificates, Identifiers & Profiles** or the sidebar's **Certificates, IDs & Profiles**, enter the **Certificates, IDS & Profiles** page.



2. click the + next to Identifiers.

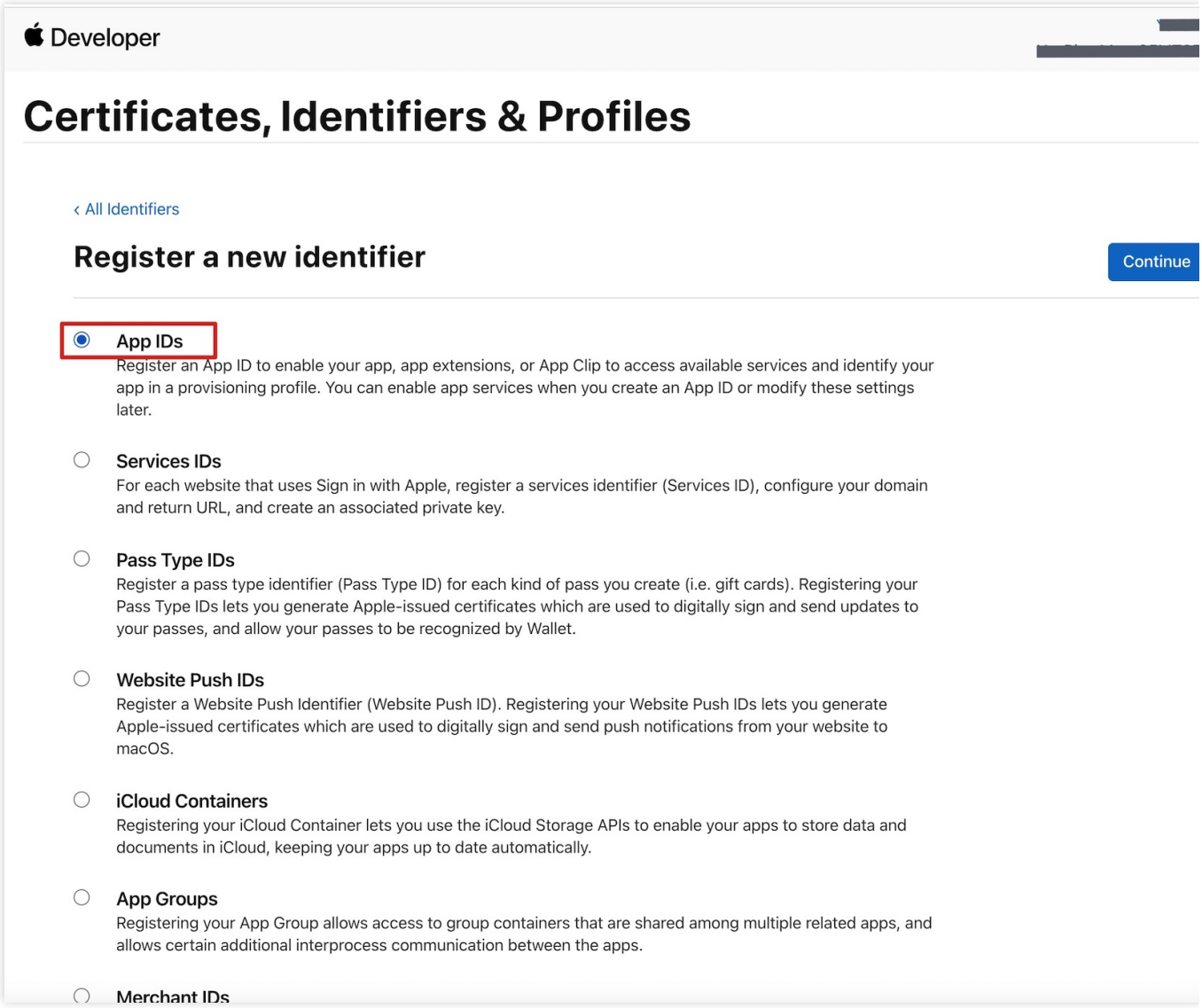


3. You can follow the steps below to create a new AppID or add a `Push Notification Service` to your existing AppID.

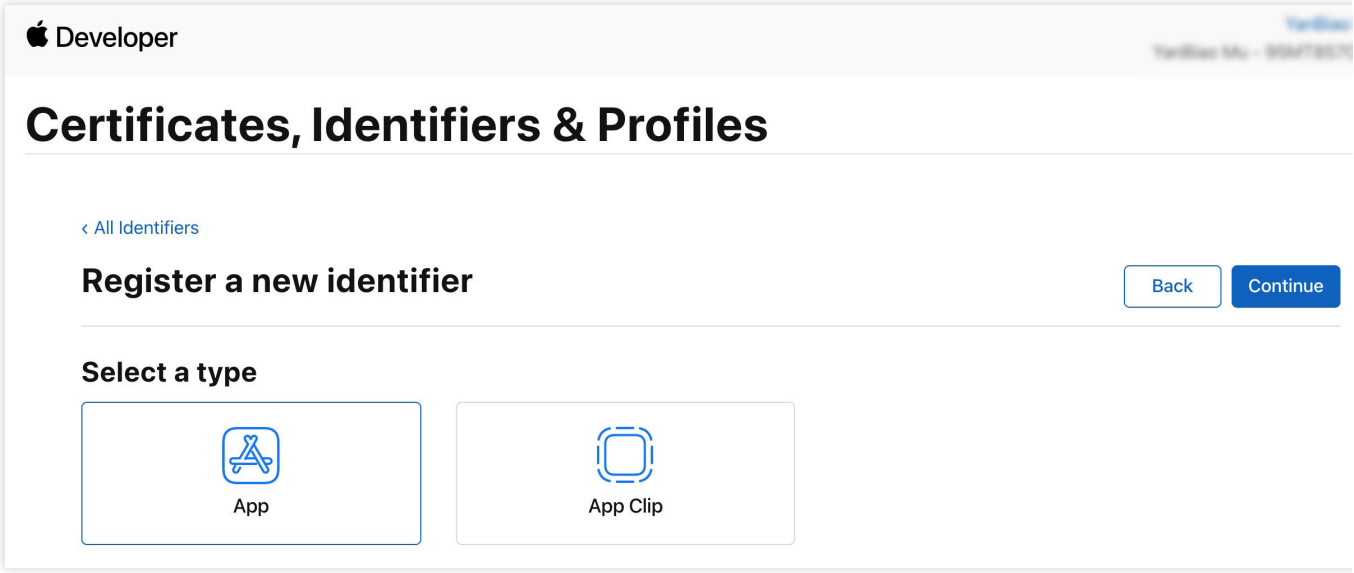
Note:

Your App's `Bundle ID` cannot use the wildcard `*`, otherwise, the remote push service cannot be used.

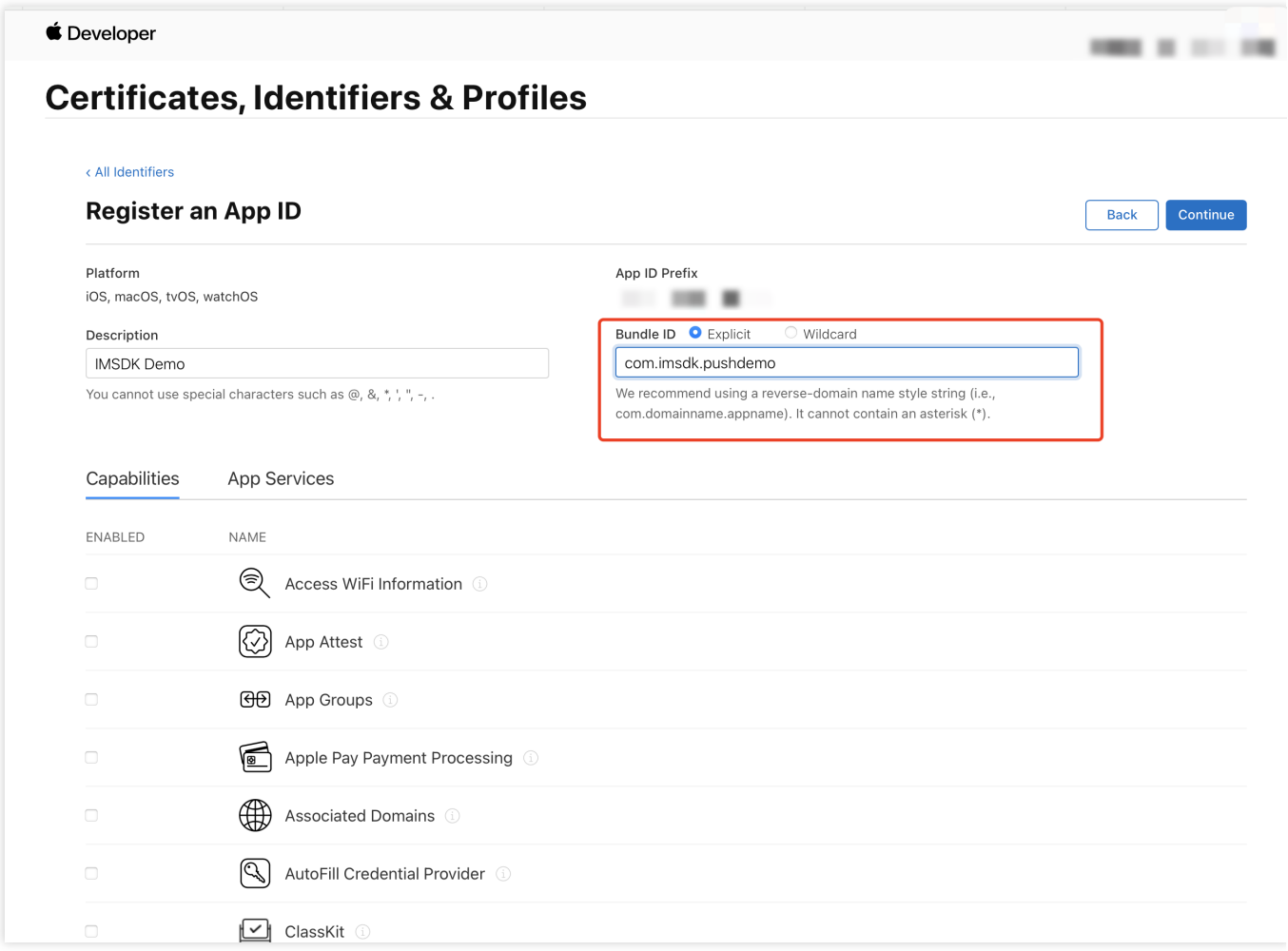
4. Check the **App IDs** box, click **Continue** to proceed to the next step.



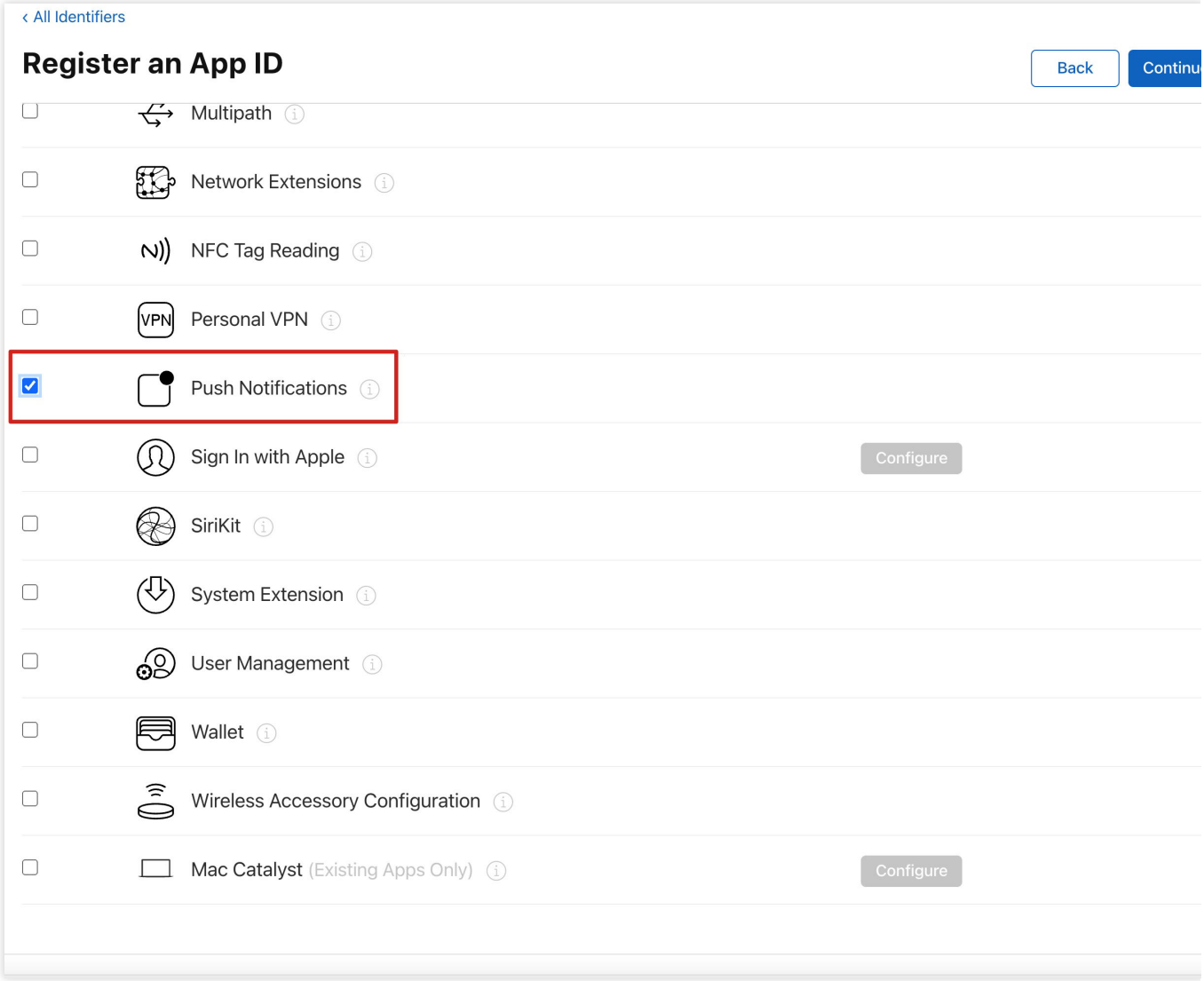
5. Select **App**, click **Continue** to proceed to the next step.



6. Configure the `Bundle ID` and other information, click **Continue** to proceed to the next step.

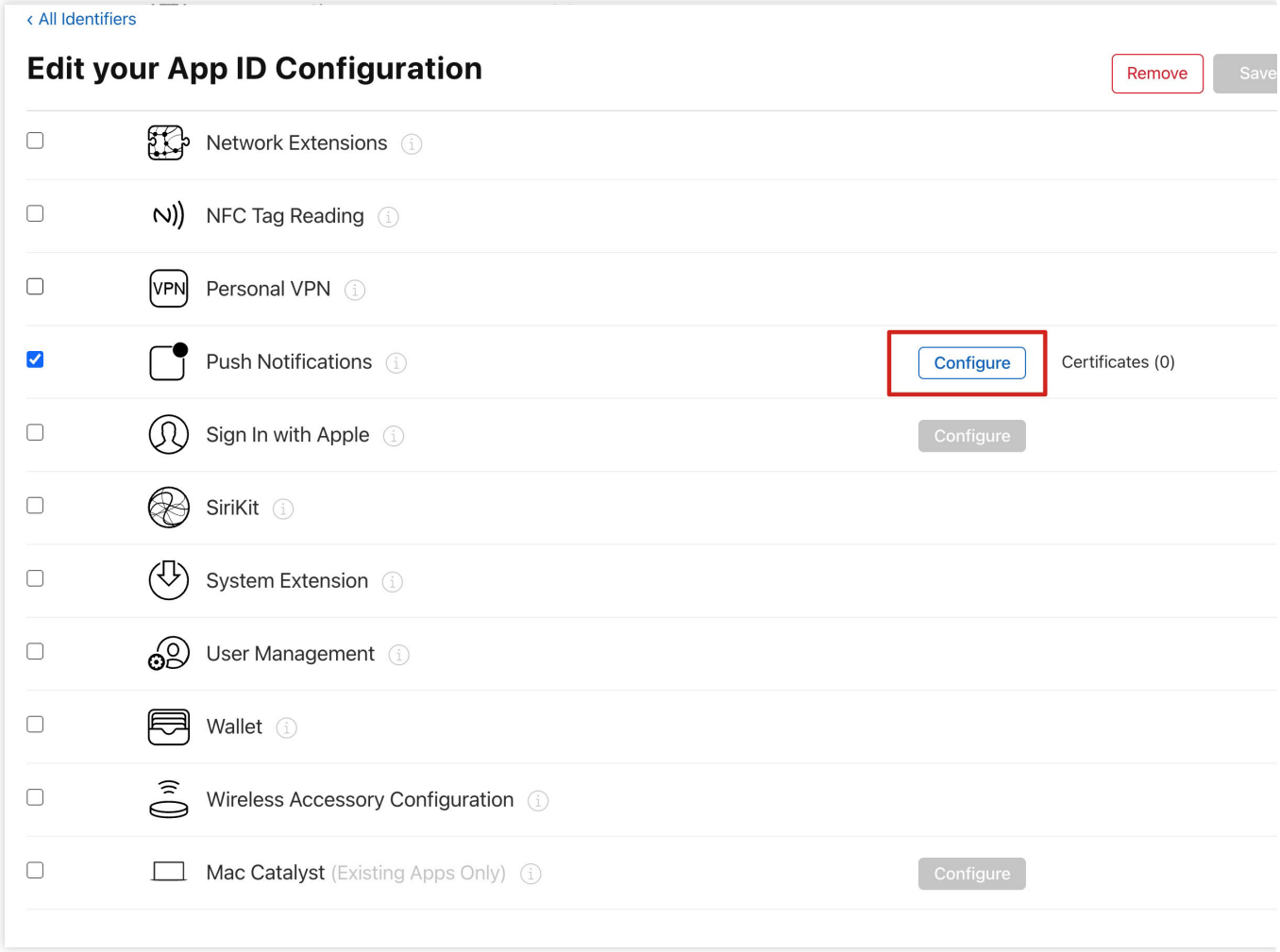


7. Check the **Push Notifications** box to enable the remote push service.

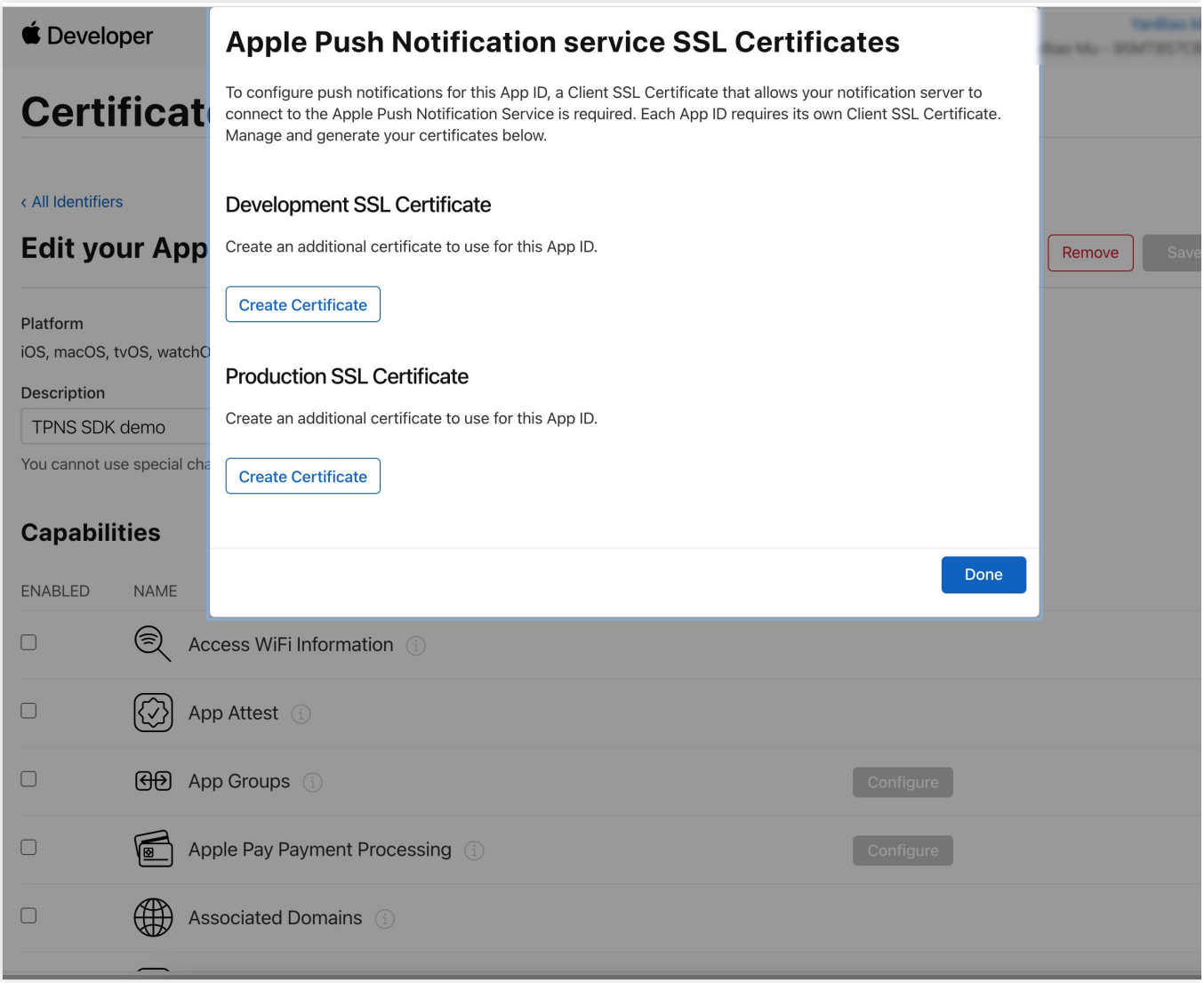


Certificate Generation

1. Select your AppID and choose **Configure**.



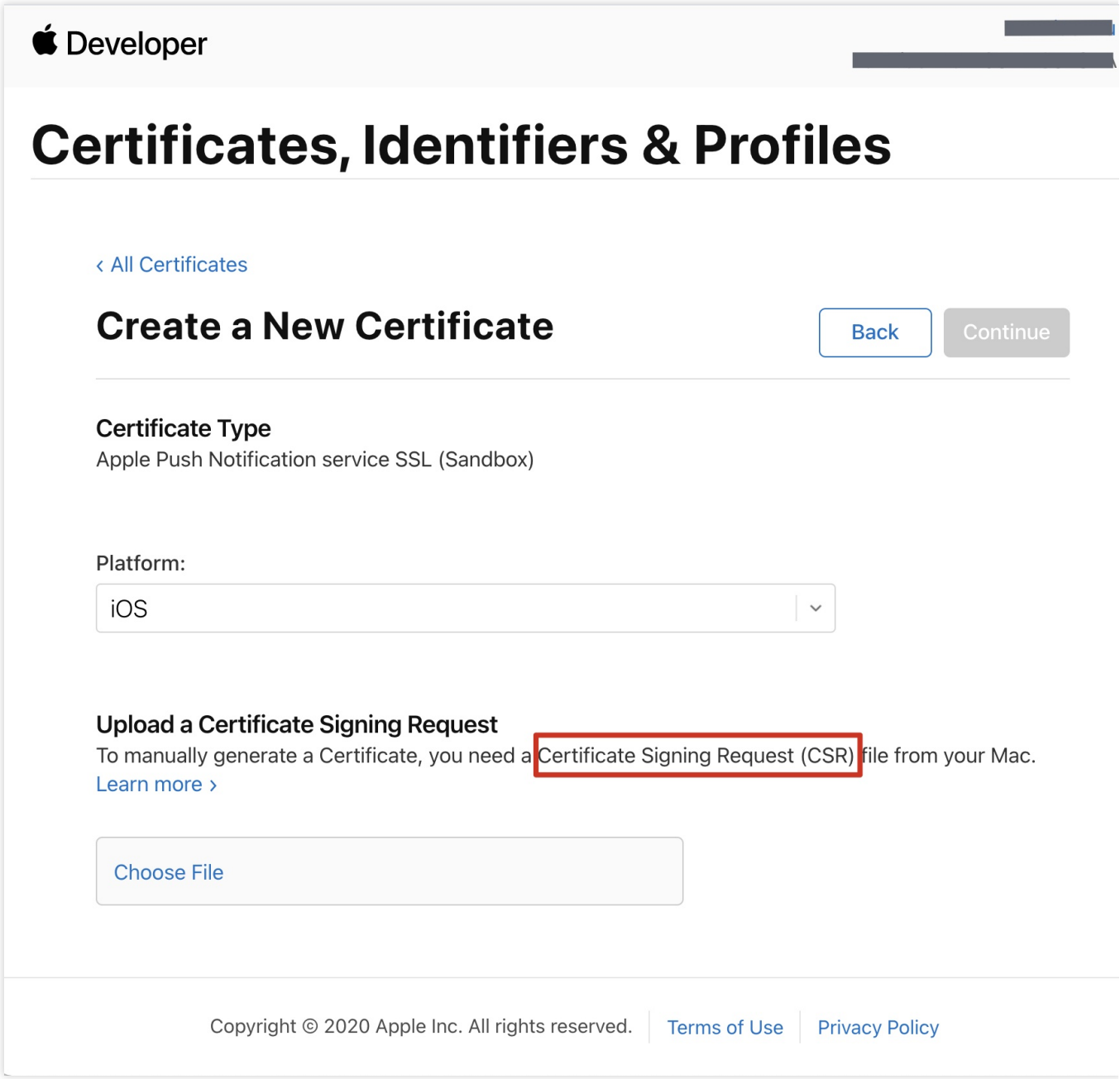
2. In the **Apple Push Notification service SSL Certificates** window, there are two `SSL Certificates` for the development environment (Development) and the production environment (Production), as shown below:



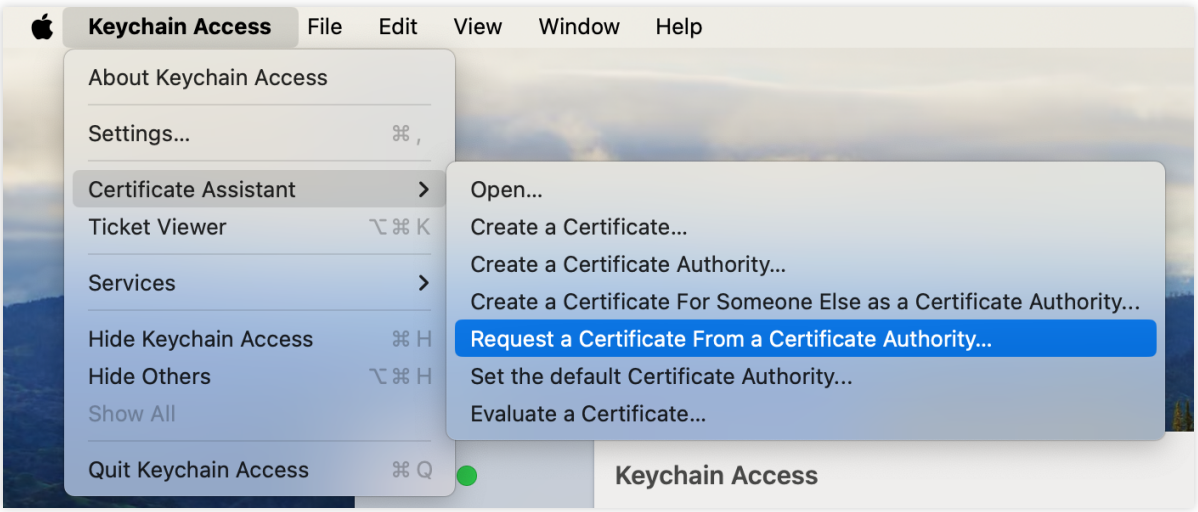
3.

We

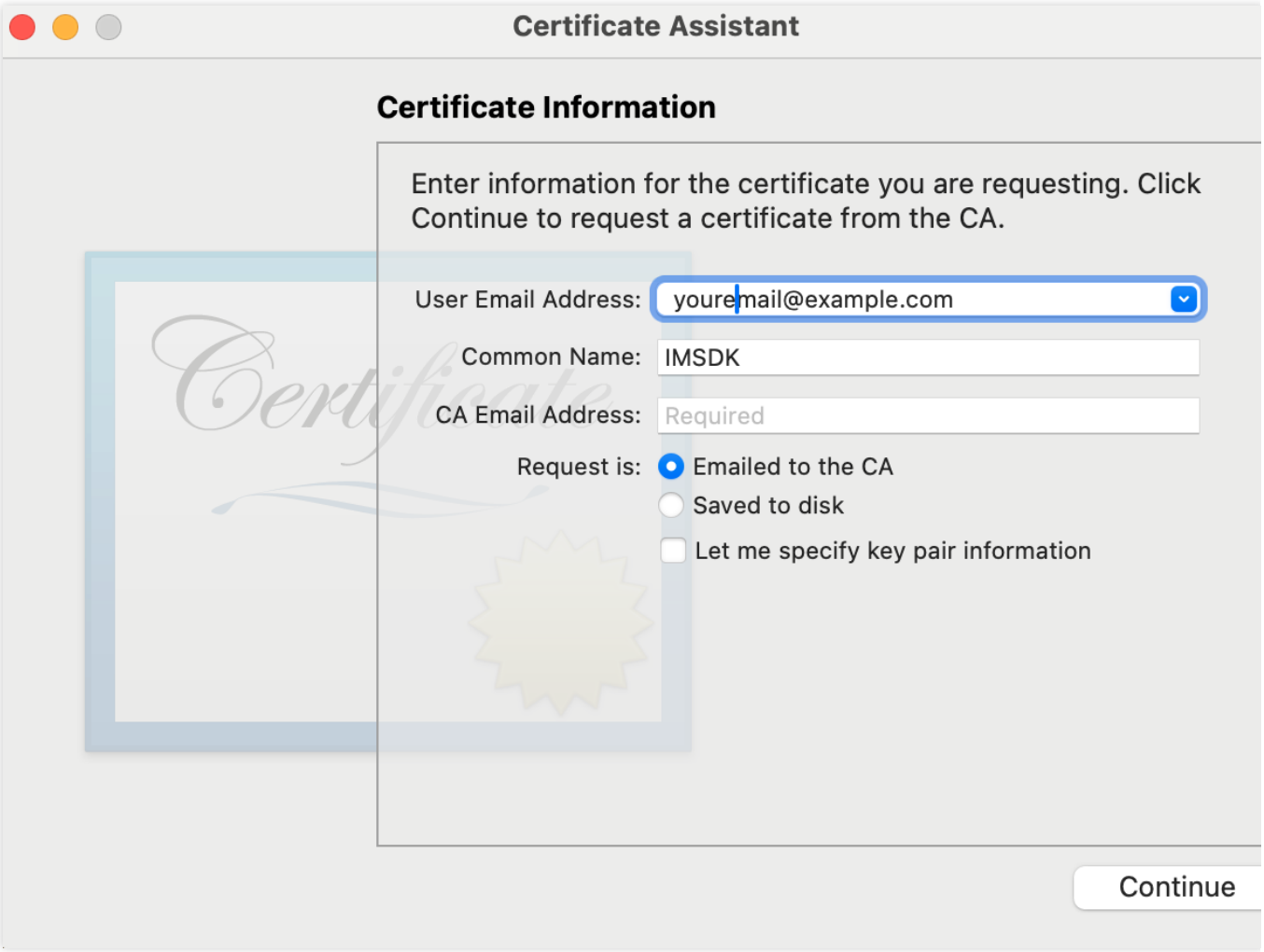
first select the **Create Certificate** for the Development environment, the system will prompt us that we need a Certificate Signing Request (CSR).



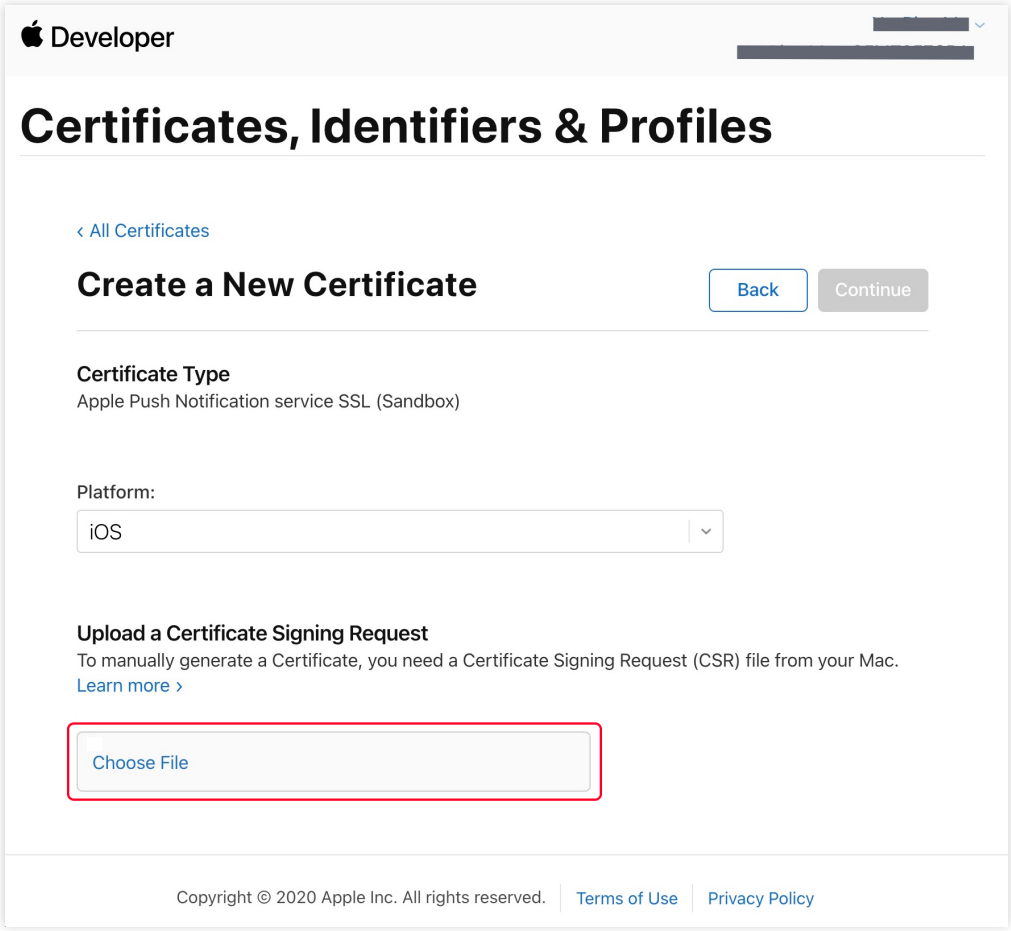
4. On a Mac, open **Keychain Access tool**, in the menu select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority** (`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`).



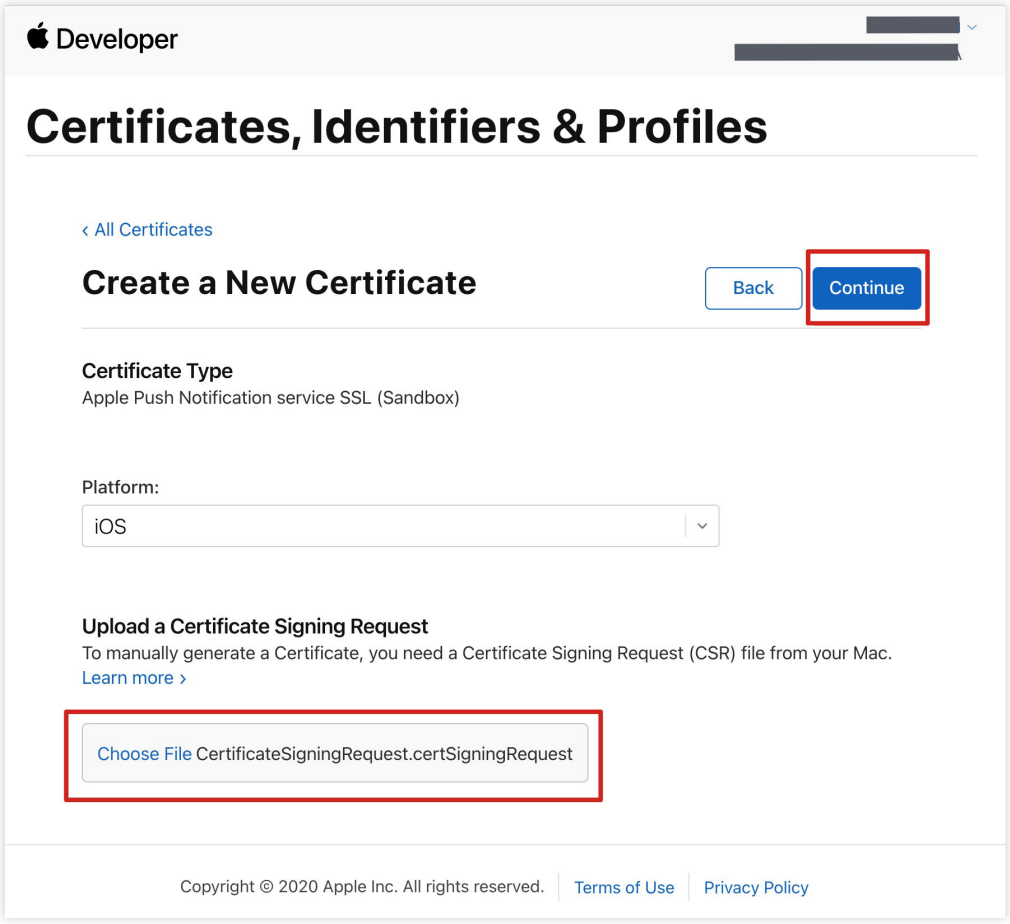
5. Enter your email address, Common Name (your name or company name), select **Save to disk**, click **continue**, the system will generate a `*.certSigningRequest` file.



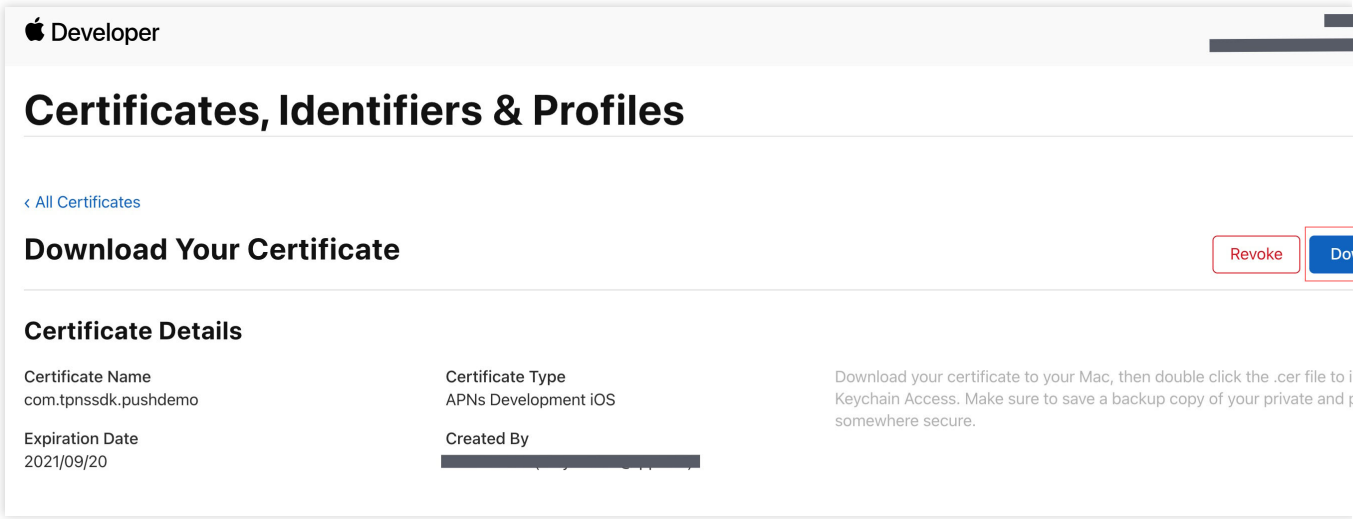
6. Go back to the page on the Apple Developer website mentioned in Step 3, click **Choose File** to upload the generated `*.certSigningRequest` file.



7. Click **Continue** to generate the push certificate.



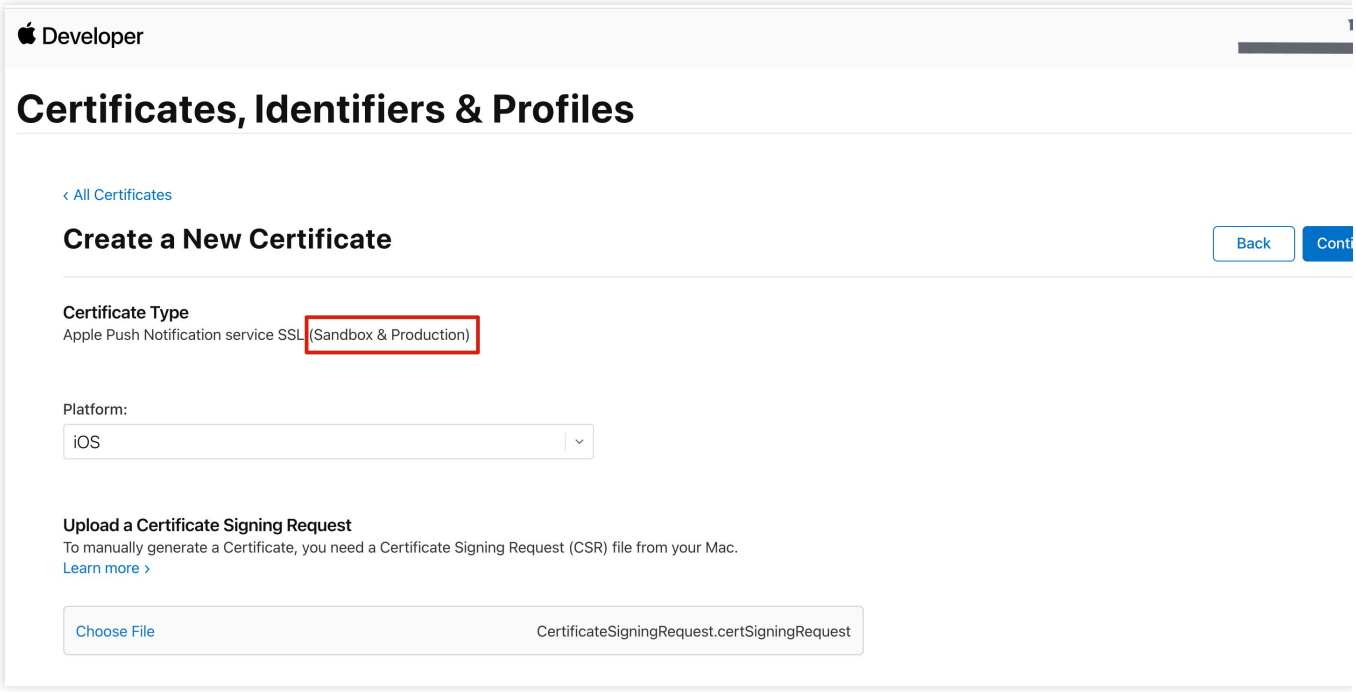
8. click **Download** to download the `Development SSL Certificate` to your local environment.

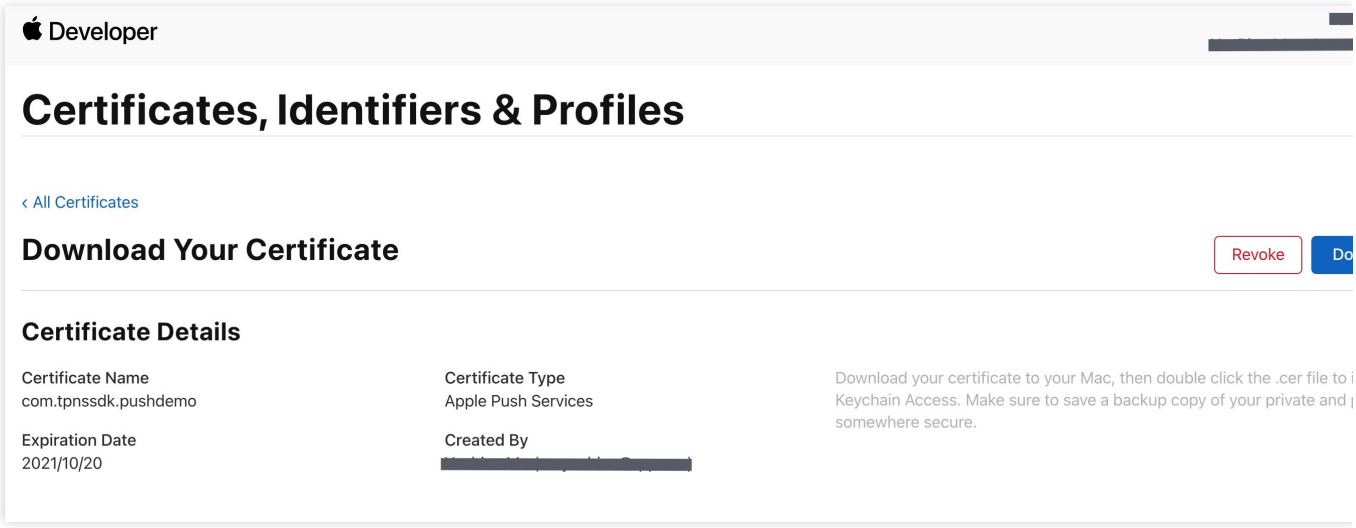


9. Repeat steps 1 - 8 above to download the `Production SSL Certificate` for the production environment to your local machine.

Note

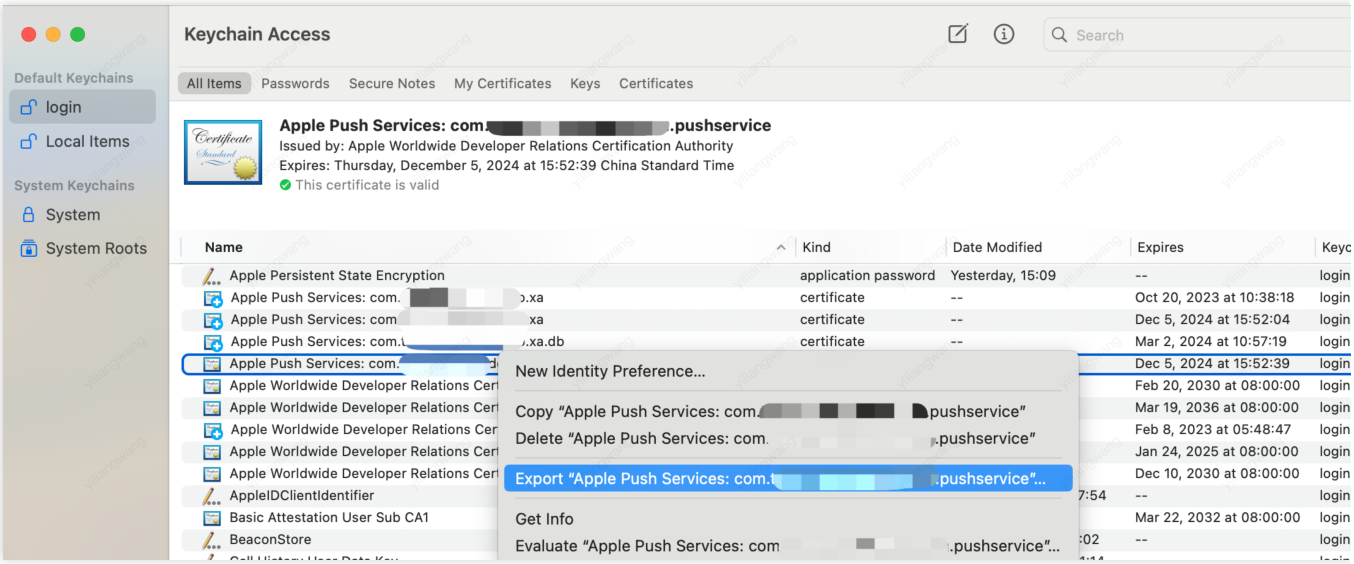
The certificate for the production environment is actually a combined certificate of Development (Sandbox) + Production, and it can be used as a certificate for both the development and production environments.





10. Double-click the downloaded `SSL Certificate` for the development and production environments. The system will import it into the keychain.

11. Open the Keychain App, go to **log in to** > **My Certificates**, right-click to export the newly created `Apple Development IOS Push Services` and `Apple Push Services` for the development and production environments as `p12` files respectively.

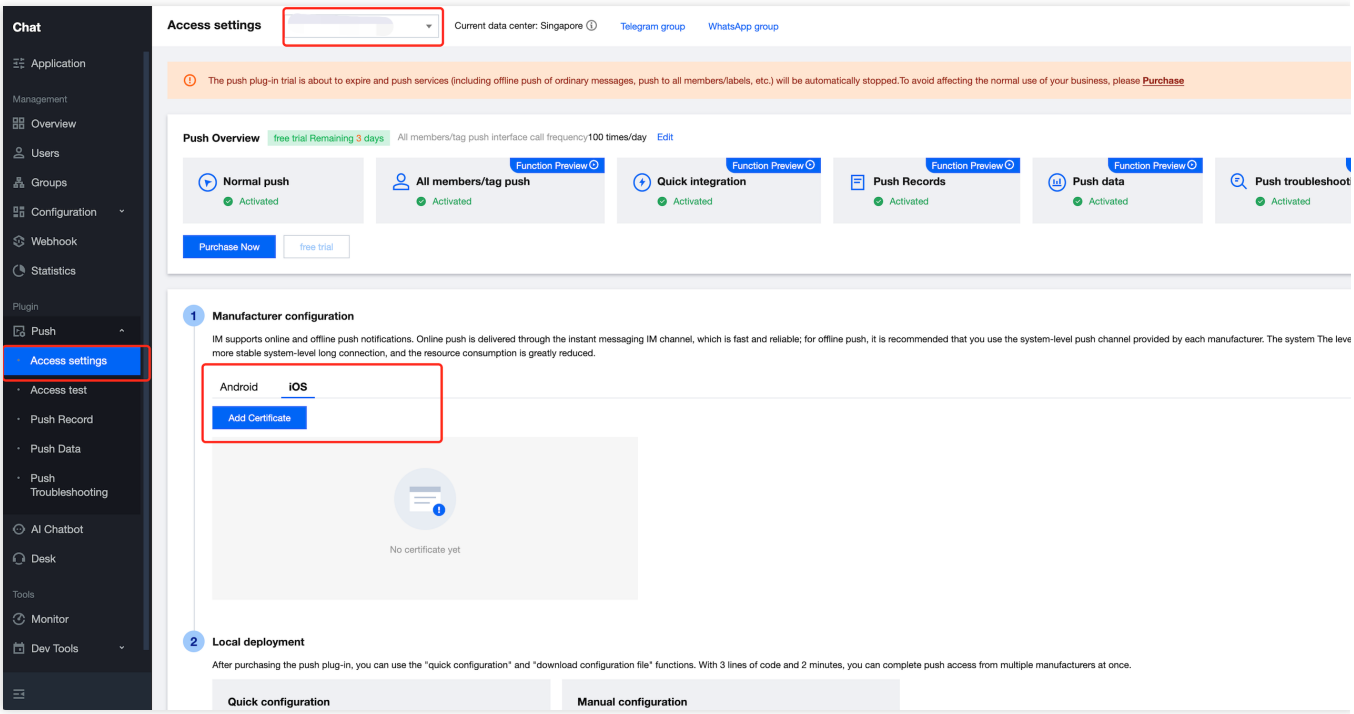


Note

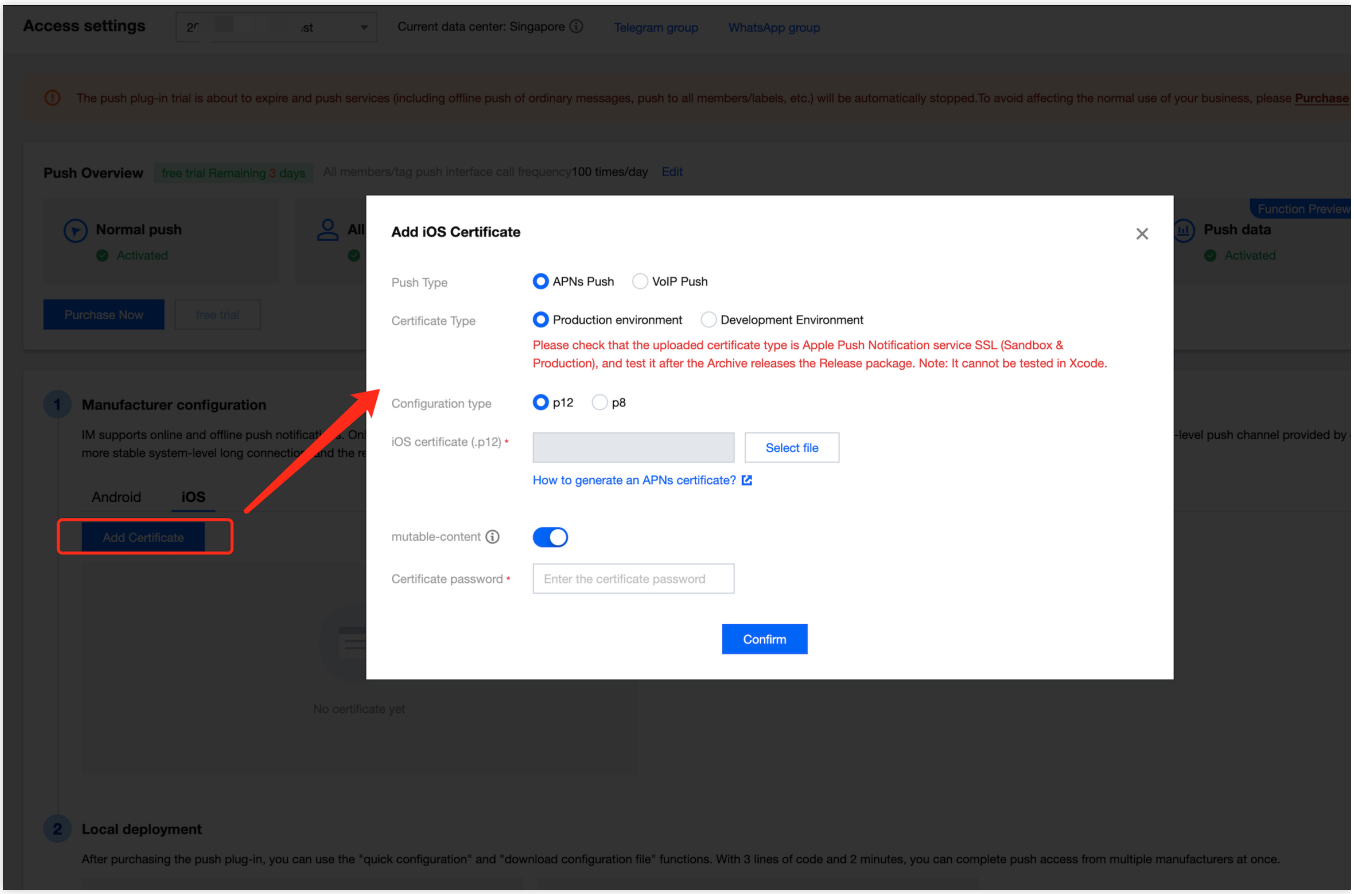
Set a password when saving the `.p12` file.

Step 2: Upload the certificate to the console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page



3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the certificate type, upload the iOS Certificate (.p12), set the certificate password, and click **Confirm**.



Note:

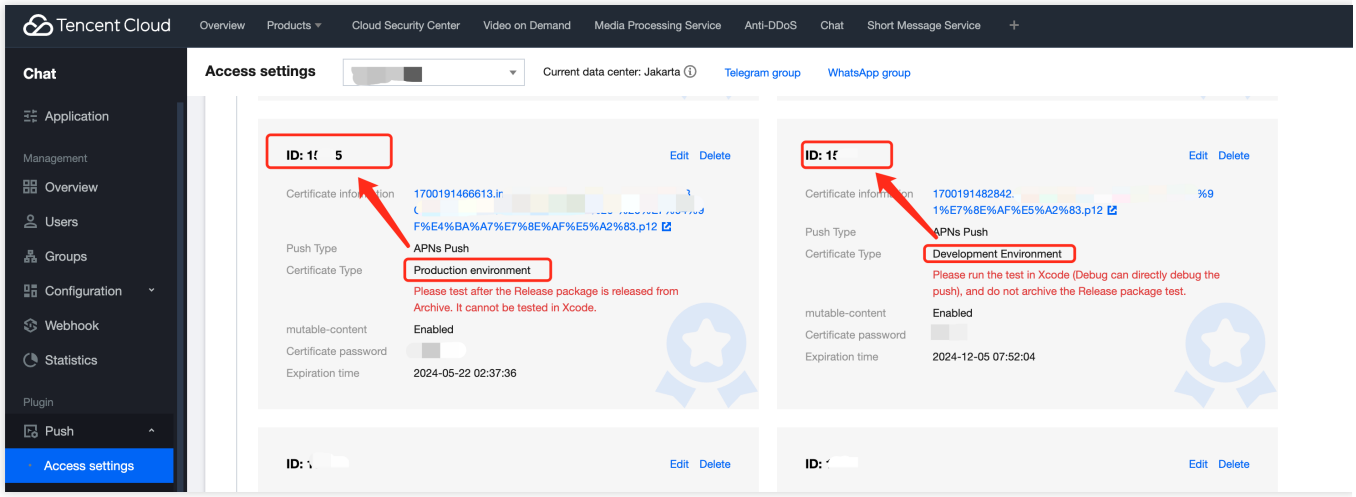
We recommend naming the uploaded certificate in English (special characters such as brackets are not allowed).

You need to set a password for the uploaded certificate. Without a password, push notifications cannot be received.

For an app published on App Store, the environment of the certificate must be the production environment. Otherwise, push notifications cannot be received.

The uploaded .p12 certificate must be your own authentic and valid certificate.

5. After the pending certificate information is generated, record the certificate's ID.

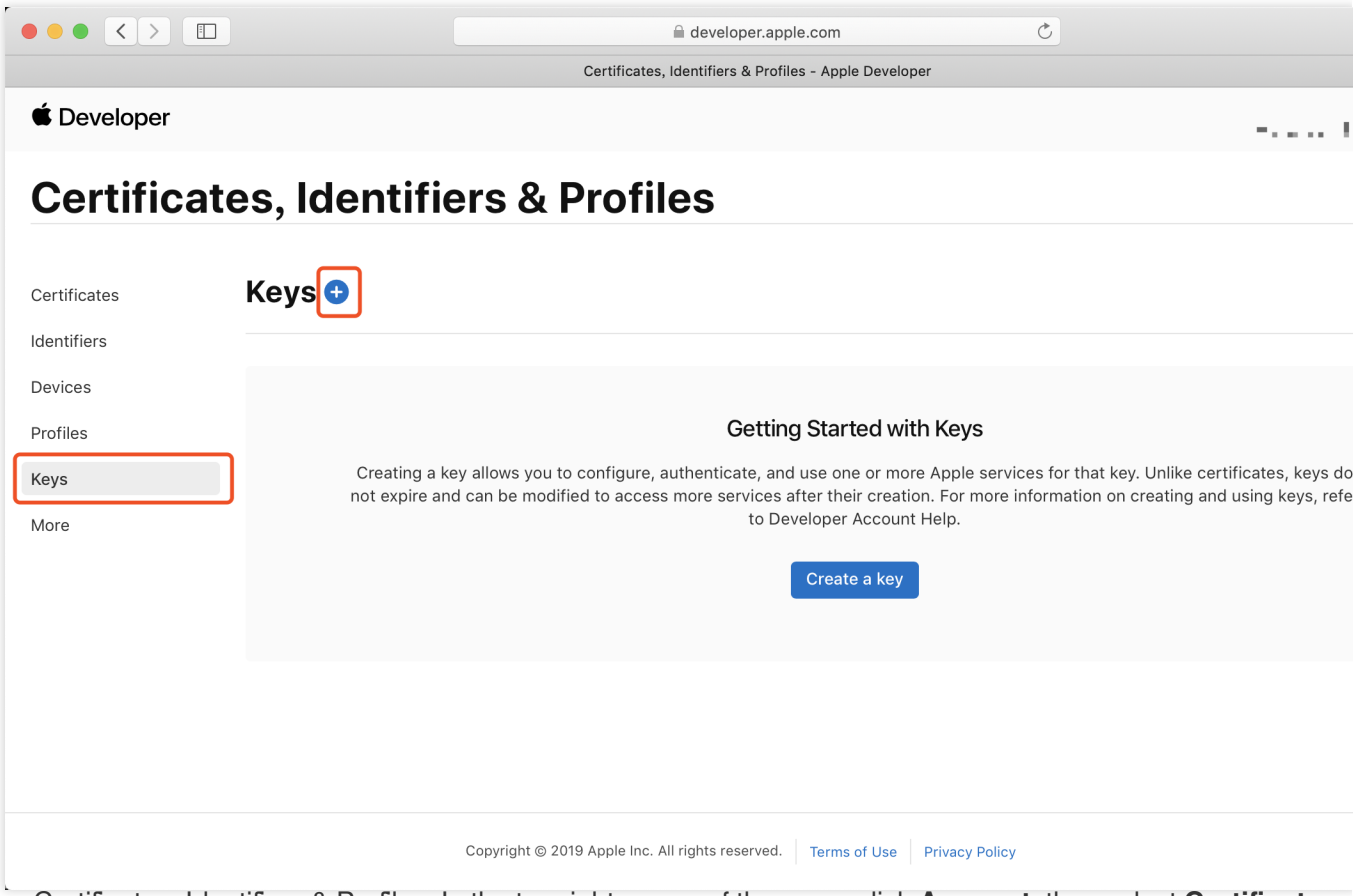


2. Using a p8 certificate (supports Dynamic Island push notifications)

p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a p8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications. In addition, p8 certificates support Dynamic Island push notifications.

Step 1: Apply for an APNs certificate

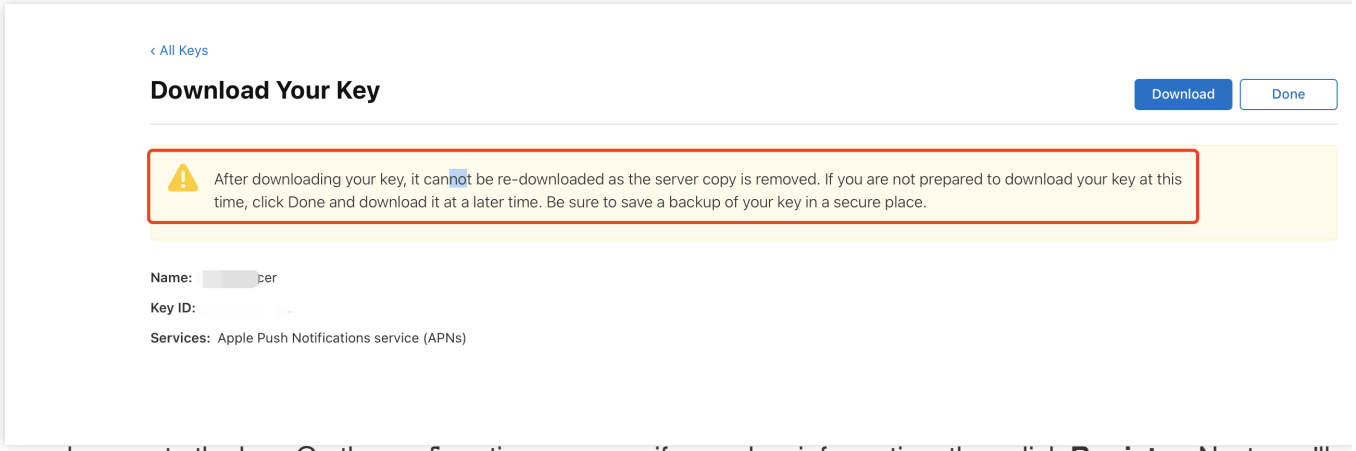
To create a p8 certificate file, first log in to Apple Developer Center.



1. Enter Certificates, Identifiers & Profiles: In the top right corner of the page, click **Account**, then select **Certificates, Identifiers & Profiles** from the dropdown menu.

2. To create a new App ID: in the left-hand menu, click **Identifiers**, then click the **+** on the right to create a new App ID. Fill in the relevant information and click **Continue**.

3. To create a new key: in the left-hand menu, click **Keys**, then click the **+** on the right to create a new key. Enter the name of the key, then check **Apple Push Notifications service (APNs)** and click **Continue**.



Confirm and generate the key: On the confirmation page, verify your key information, then click **Register**. Next, you'll see a page prompting you to download the key. Click **Download** and save the generated .p8 file to your computer.

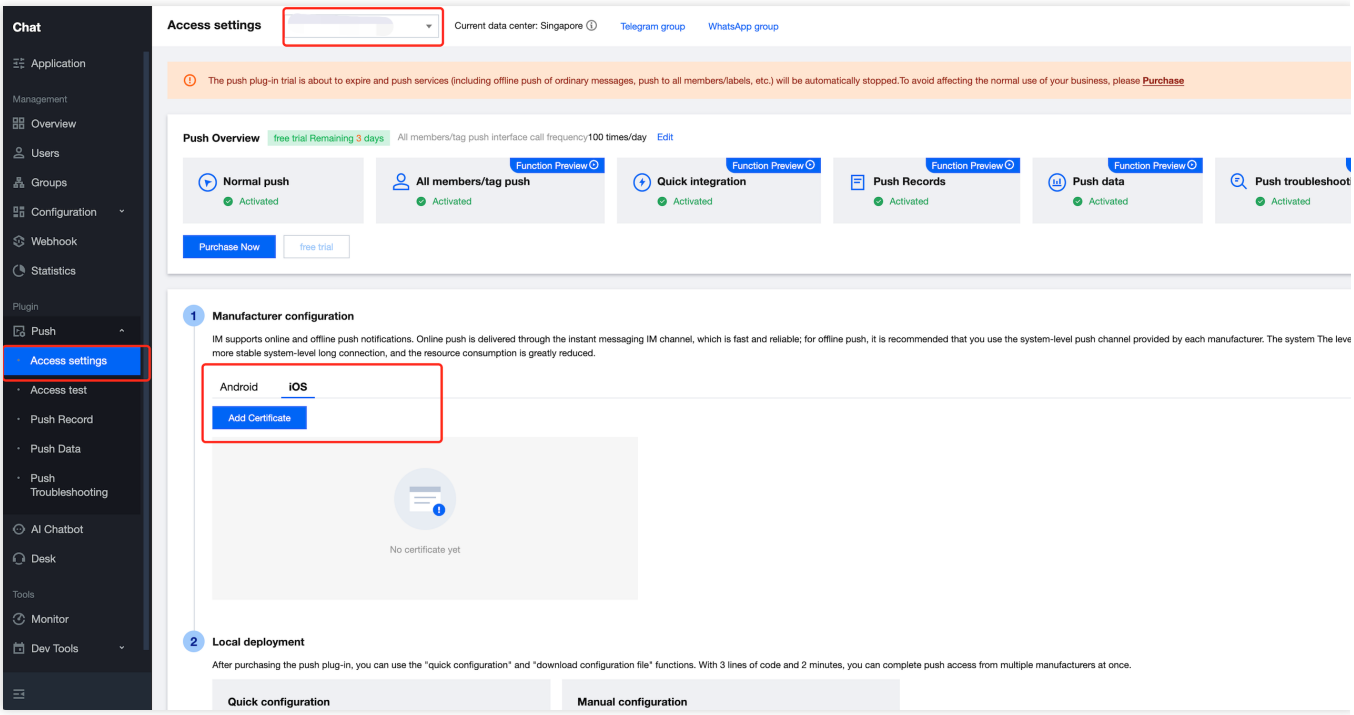
Note:

The p8 certificate can only be downloaded once; please save it properly.

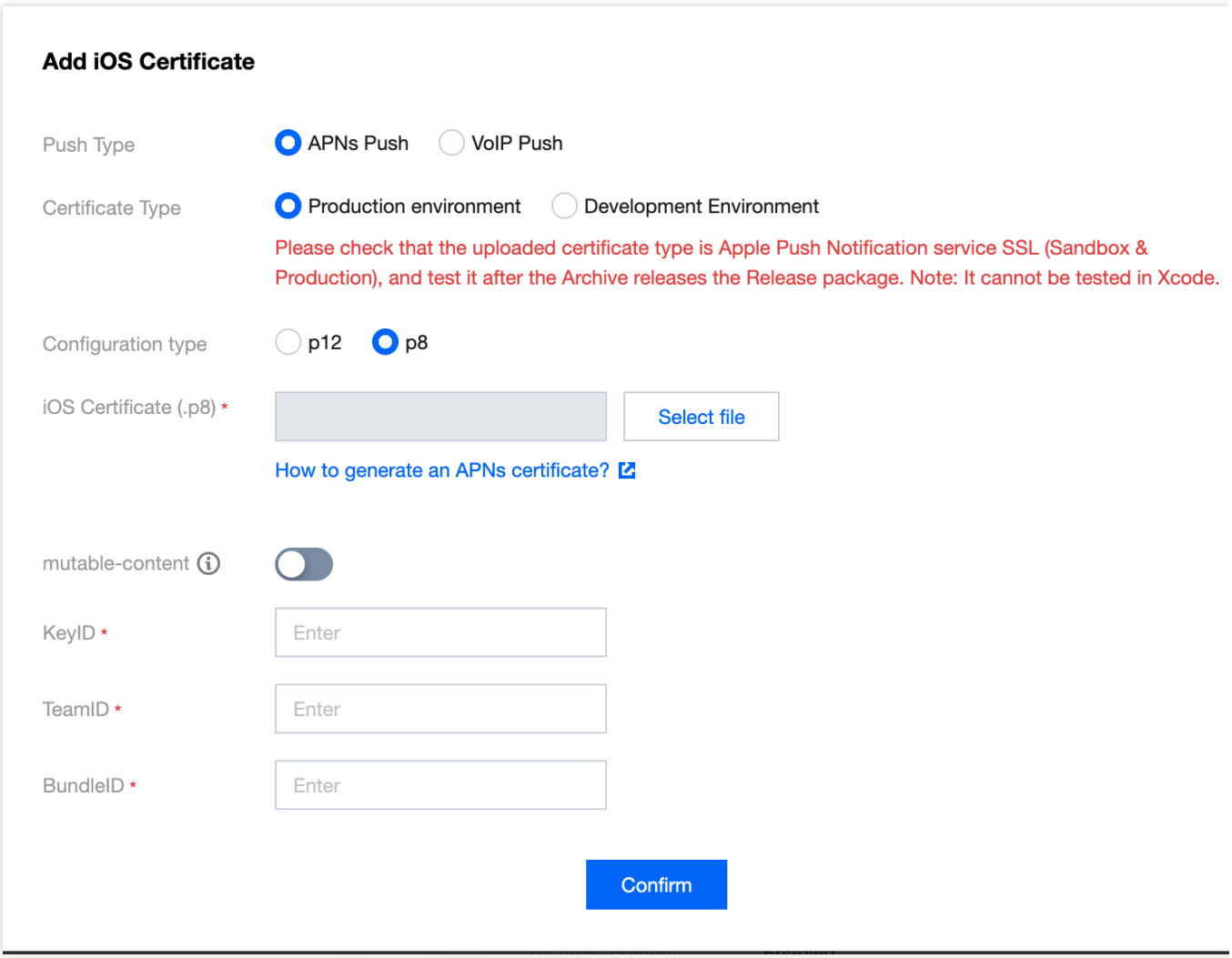
Please safeguard the downloaded p8 file, as you will not be able to download it again. You can use this p8 certificate to configure your iOS applications to receive push notifications.

Step 2: Upload the p8 certificate to the IM console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page



3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the .p8 certificate



Note:

Key ID: This is the unique identifier for your APNs Auth Key. When you create a new APNs Auth Key in the Apple Developer Center, a Key ID will be generated for you. You can find it in the "Certificates, Identifiers & Profiles" section under "Keys".

Team ID: This is the unique identifier for your developer account. You can find it on the account details page of the Apple Developer Center. Click "Membership" in the upper right corner, and you can find your Team ID in the "Membership Details" section.

Bundle ID: This is the unique identifier for your application, also known as the app ID. You can find it in the "Certificates, Identifiers & Profiles" section of the Apple Developer Center. Select "Identifiers", then find the corresponding Bundle ID in your list of applications.

uniapp

Last updated：2024-06-13 10:21:45

TencentCloud-TIMPush is a Tencent Cloud Chat Push plugin. Currently, the push supports channels from manufacturers such as Mi, Huawei, Honor, OPPO, Vivo, Meizu, APNs, OnePlus, realme, iQOO, and Apple.
Android
iOS

Register an application with the vendor push platform

To utilize the offline push feature, you need to register your app on each vendor's push platform to obtain parameters such as AppID and AppKey. Currently, the mobile manufacturers supported in China include: Mi, Huawei, Honor, OPPO, Vivo, Meizu, and internationally Google FCM is supported.
Mi
Huawei
OPPO
vivo
Meizu
HONOR
Google FCM

Step 1: Register a Xiaomi Developer Account

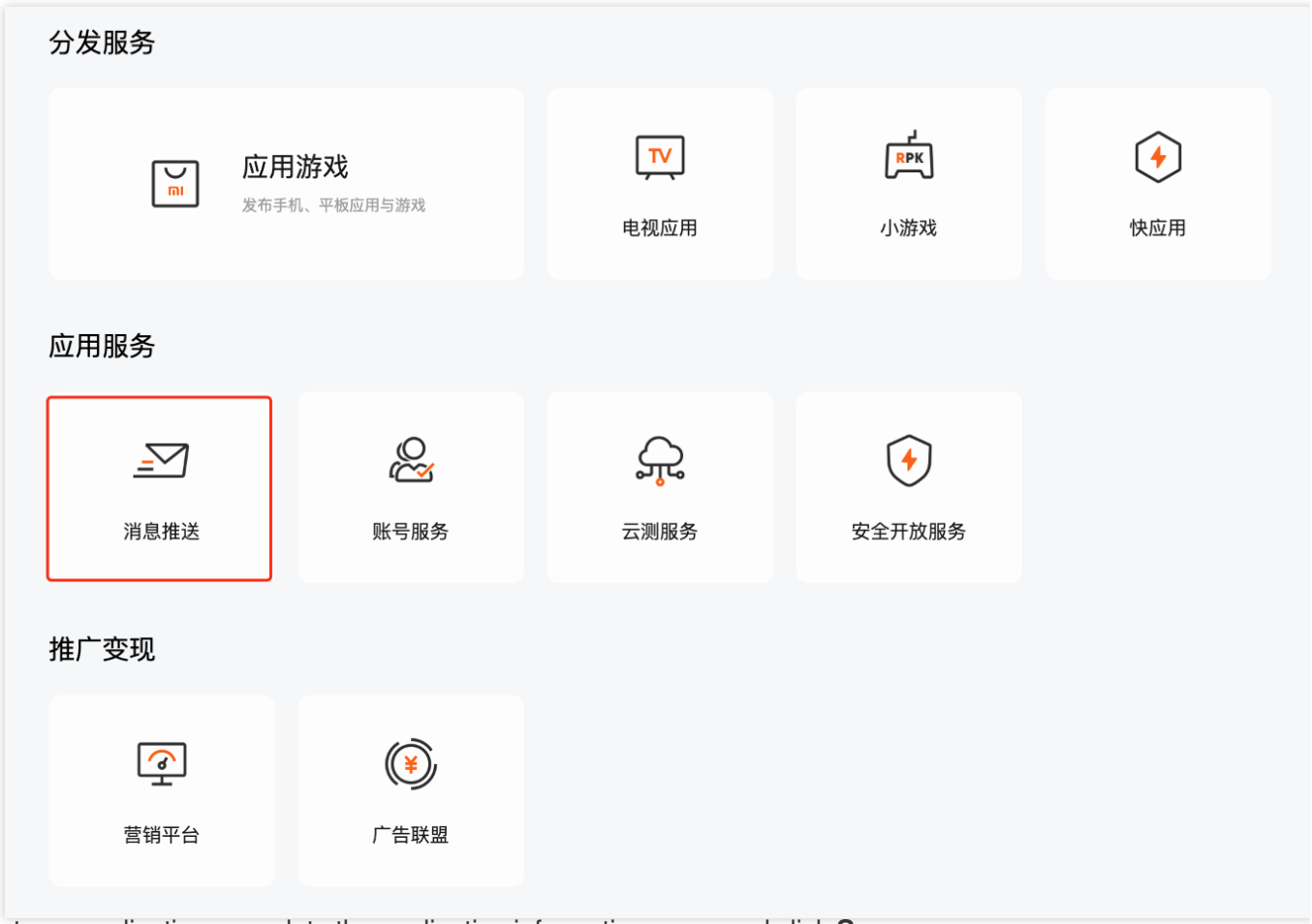
Visit Xiaomi Open Platform, register a Xiaomi developer account. For details, refer to Enterprise Developer Account Registration Process.

Step 2: Create an Application

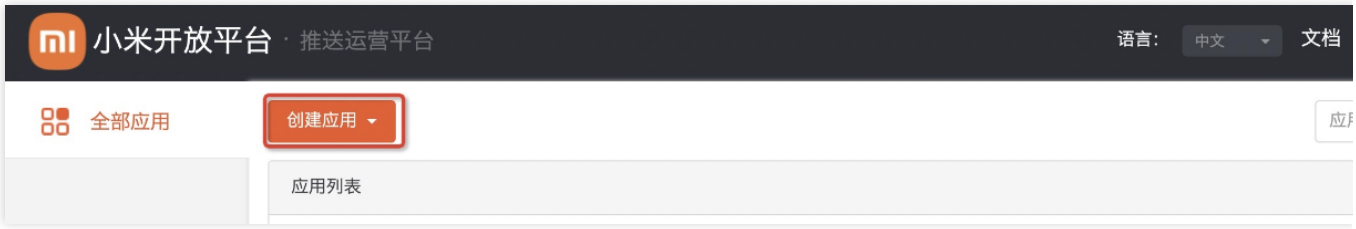
1. In the management console, click **Message Push**.

Note:

If you log in with a personal account, it will display "Sorry, you currently do not have push/review permissions".



2. Create an application, complete the application information page, and click **Save**.



Note:

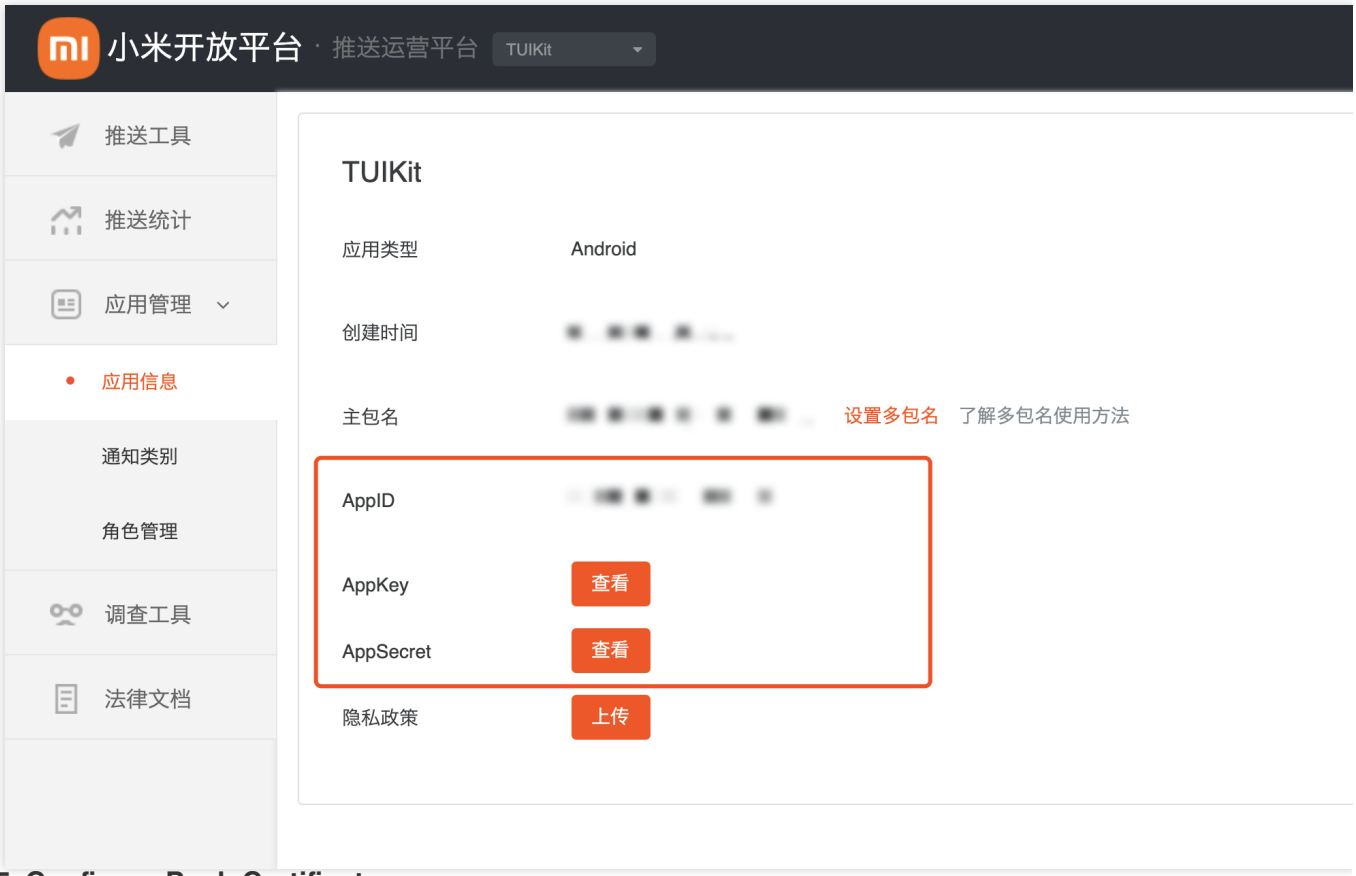
The application package name should be consistent with the plugin application package name.

Step 3: Enable Push

Go to the **Application List** page on the Push Operation Platform, click on the corresponding application name to **Enable Push**, and confirm to enable.

Step 4: View and Obtain Application Information

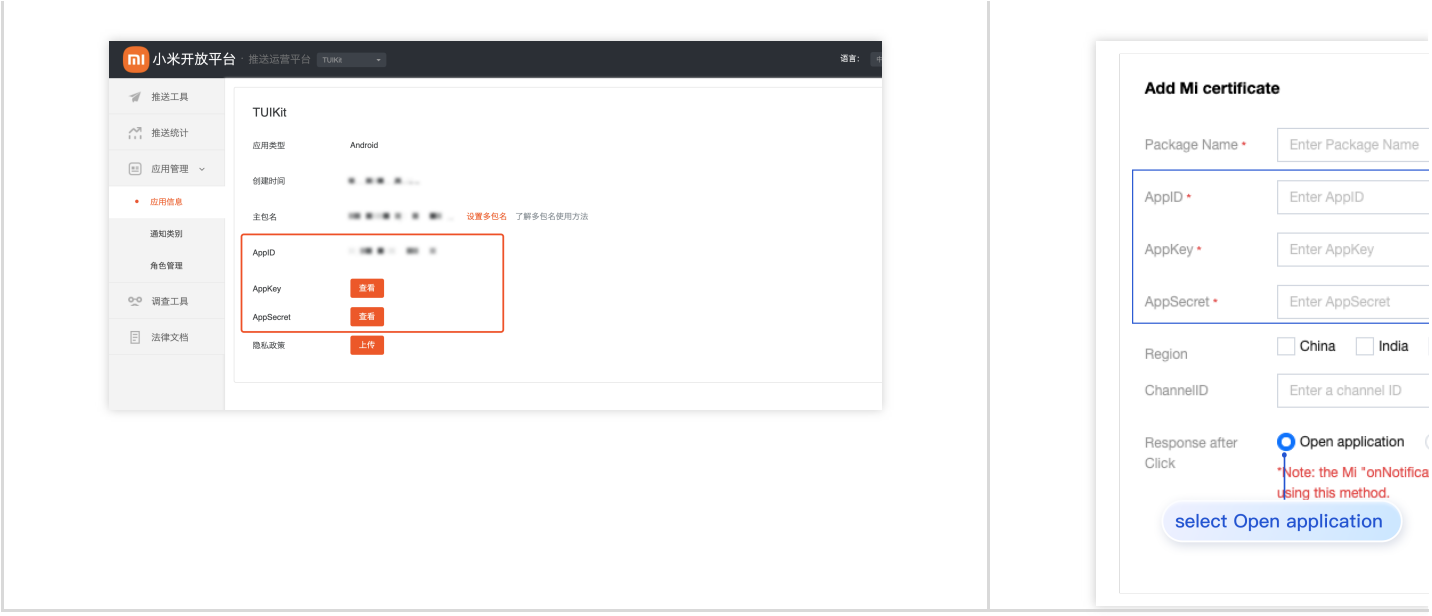
Enter the **Application Information** page on the Push Operation Platform, and review the **Application Information**.



Step 5: Configure Push Certificate

Log in to Tencent Cloud Chat Console, go to **Push** > **Access Settings** feature section to add vendor push certificates, and configure the parameters such as AppId, AppKey, AppSecret obtained from the vendor to the added push certificate.

Vendor Push Platform	Configuring in the IM console
	Note: **Specified in-app page** link canno dispatching event listening for the configuration of in-app page navig



Step 1: Register a Huawei Developer Account

Go to Huawei Developer Alliance, register for a Huawei Developer Account. For details, please refer to Register Account.

Step 2: Create an Application

1. In the Account Center, click **My Projects** to add a new project.

2. In the **Project Settings** section, click **Push Service** > **Activate Now**.

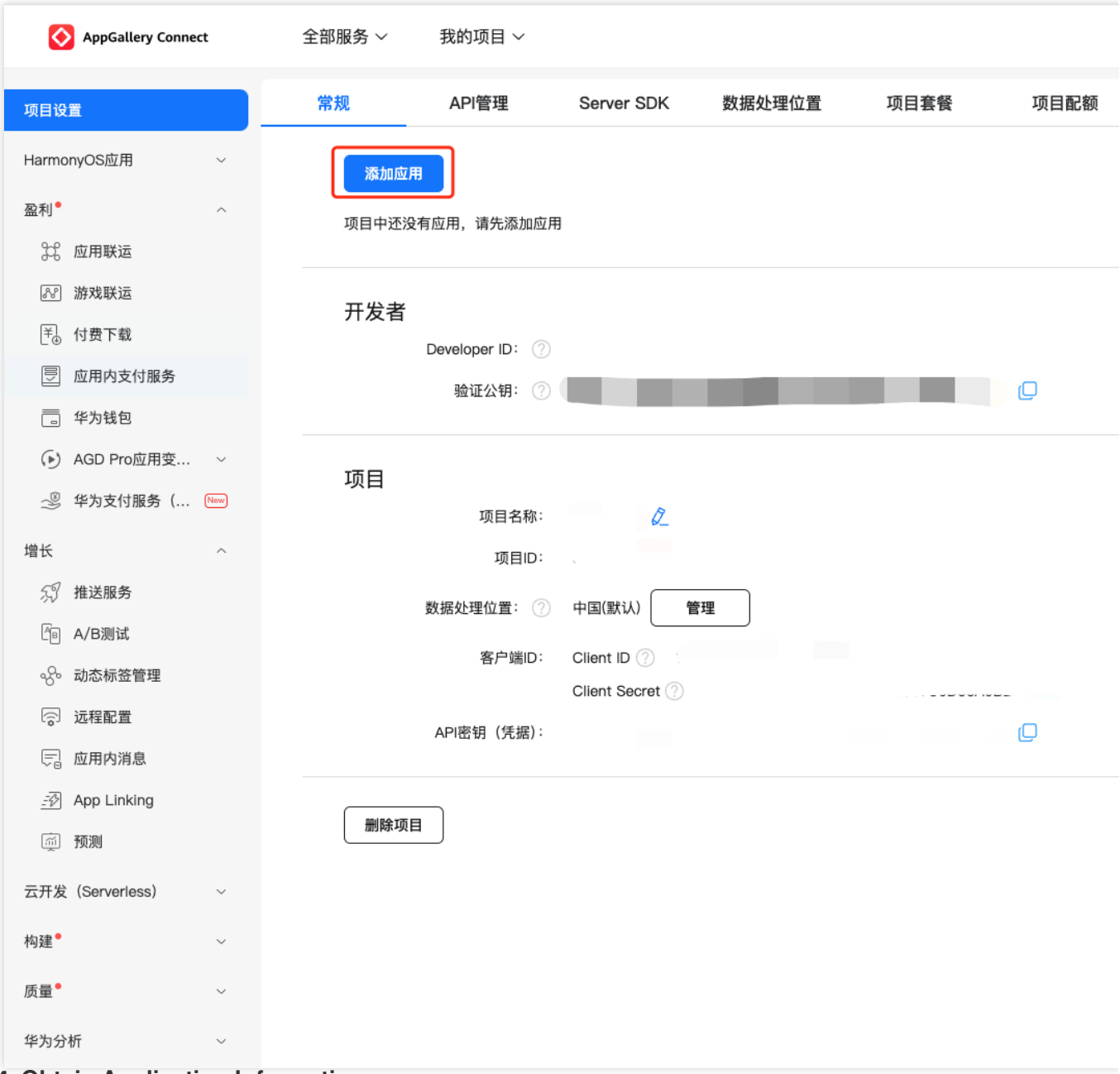
3. Click Project Settings > API Management to enable permissions for the push service.

Step 3. Add Application

Click **Project Settings > General** to add an application.

Note:

The application package name should be consistent with the plugin application package name.



Step 4: Obtain Application Information

Click **Project Settings > General** to obtain application information.

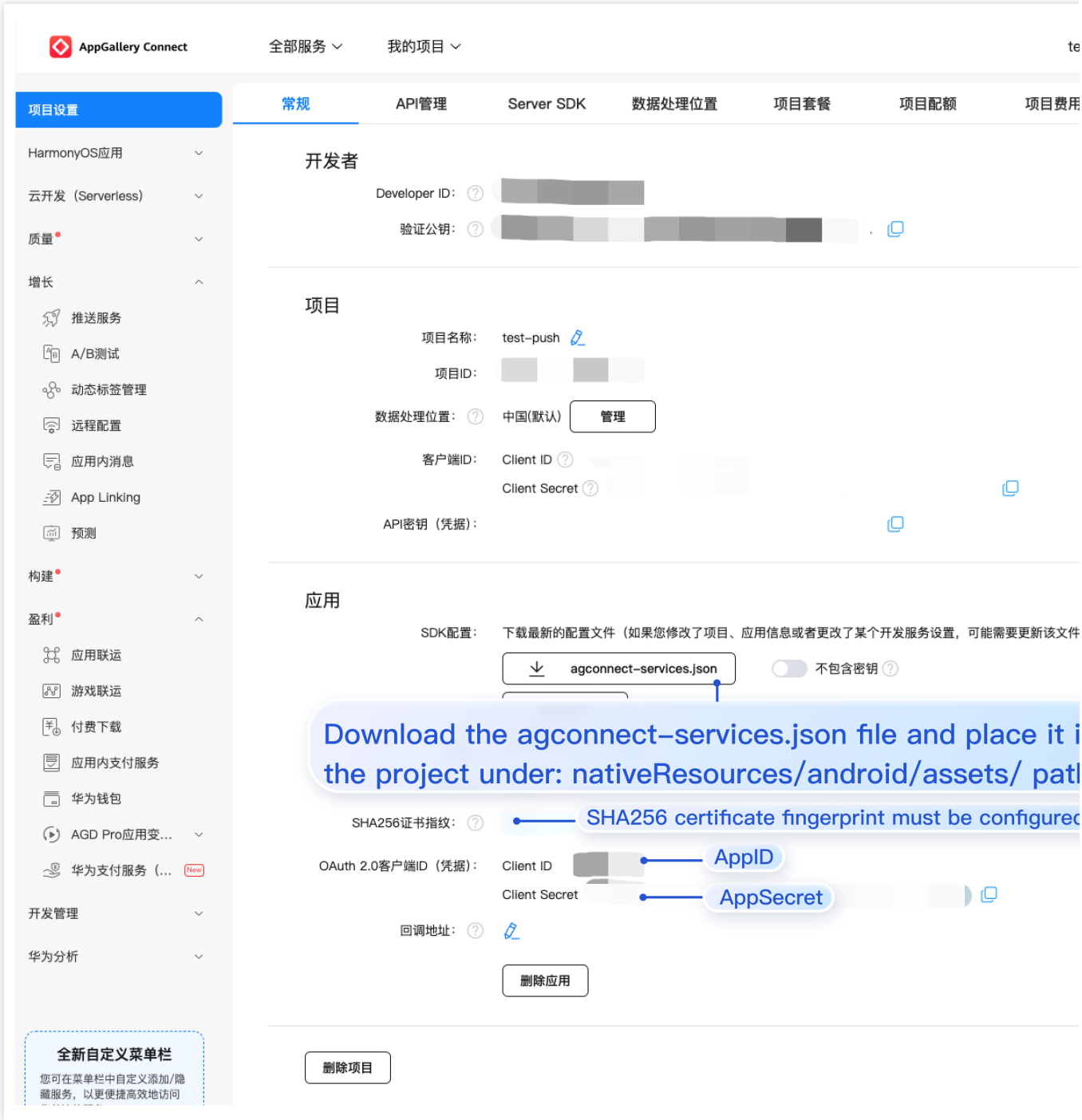
Note:

The General page contains the Client ID and Client Secret for both projects and applications, which have inconsistent parameters. Please scroll down to the bottom of the page to obtain the application's Client ID and Client Secret.

You must add the SHA256 Certificate Fingerprint, which should match the SHA256 Certificate Fingerprint of your own packaging certificate.

Download the agconnect-services.json file and place it in the project under: **nativeResources/android/assets/** path.

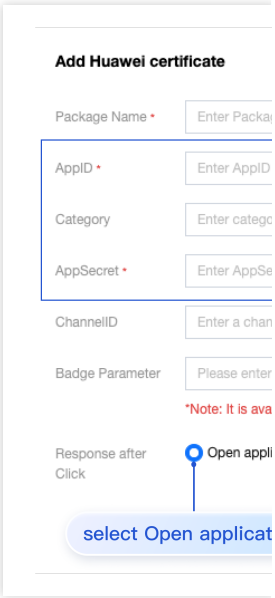
If you modify the project, application information, or development service settings, you need to re-download the agconnect-services.json file.



Step 5: Add Push Certificate

Log in to Tencent Cloud Chat Console, click **Push** > **Access Settings** to add vendor push certificates, and configure the obtained vendor's AppId, AppKey, AppSecret, and other parameters to the added push certificate.

Vendor Push Platform	Configuring in the IM console
	Note: Client ID corresponds to App **Specified in-app page** link dispatching event listening fo configuration of in-app page



Note:

Notification Bar Push: The app needs to be published on the OPPO Software Store;

Notification Bar Push Test Permission: Only 1,000 messages can be pushed per day, limited to testing purposes. After the app is published, you need to reapply for "Notification Bar Push" permission to obtain a normal message push quantity;

The platform will return review results within 1 business day. Developers can view the review results on the application page. For other inquiries, consult the Open Platform Customer Service.

Step 1: Register an OPPO Developer Account

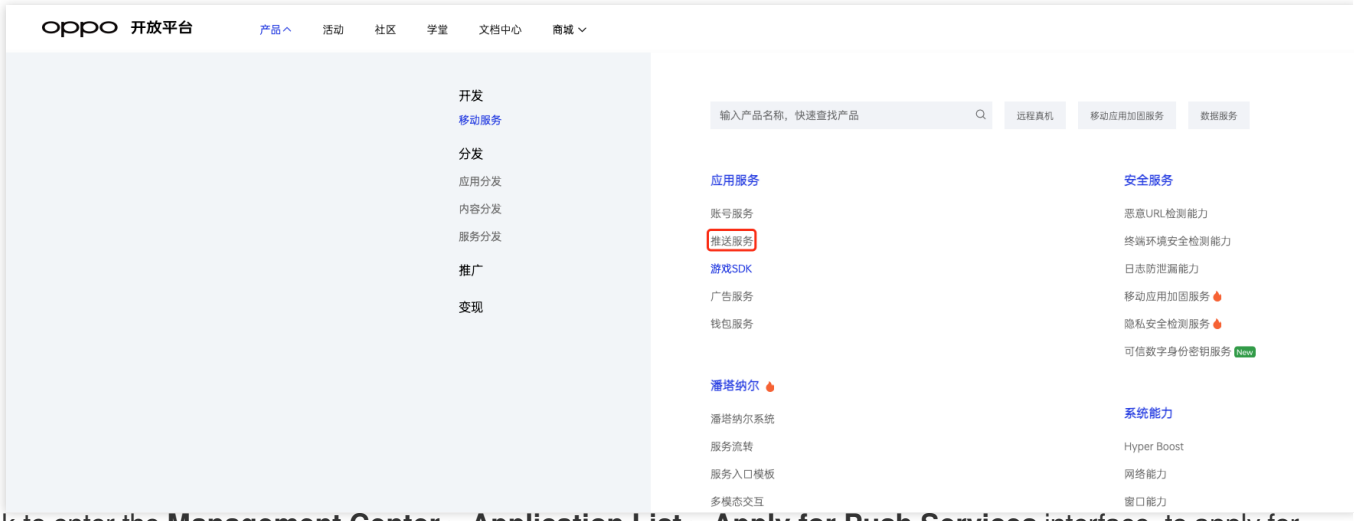
Enter OPPO Open Platform, register an OPPO Developer Account. For details, see OPPO Enterprise Developer Account Registration.

Step 2: Create an Application

Enter the OPPO Open Platform, click **Product** > **Application Distribution**> **OPPO Software Store** > **Release Application** to access the Management Center and create an application.

Step 3: Activate the PUSH Service

1. Enter the OPPO Open Platform, click **Product** > **Mobile Services** > **Push Service** to access the Push Homepage, then click **Apply for** access to activate the Push Service.

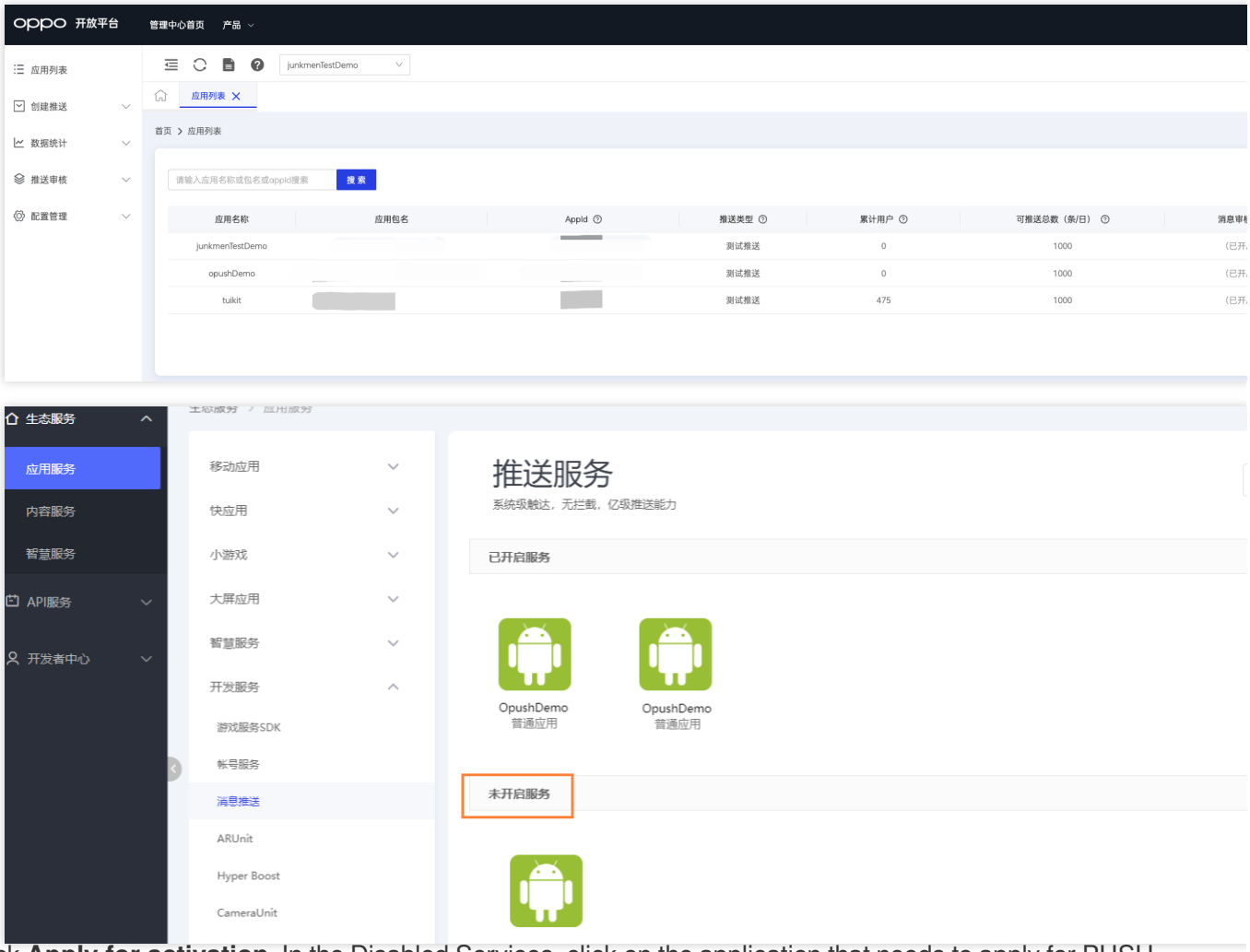


2. Click to enter the **Management Center** > **Application List** > **Apply for Push Services** interface, to apply for push permissions for applications not yet enabled.

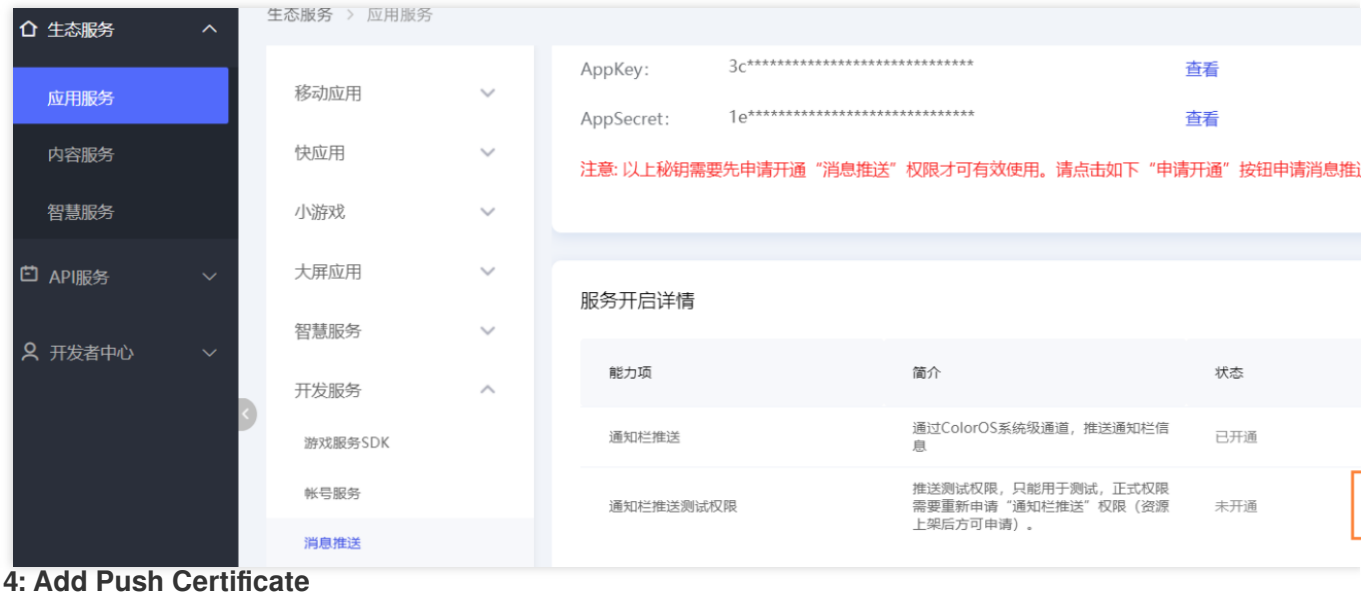
Note:

Enabled Services: Applications that have applied for PUSH permissions and have been approved.

Disabled Services: Applications that can apply for PUSH permissions.

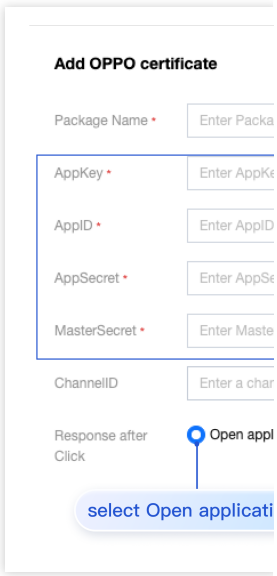


3. Click **Apply for activation**. In the Disabled Services, click on the application that needs to apply for PUSH permissions, enter the PUSH Service and click **Apply for activation**.



Step 4: Add Push Certificate

Log in to Tencent Cloud Chat Console, go to **Push** > **Access Settings** feature section to add vendor push certificates, and configure the parameters such as AppId, AppKey, AppSecret obtained from the vendor to the added push certificate.

Vendor Push Platform	Configuring in the IM console
	Note: Specify In-App Interface Link This configuration is for even configuration of in-app page 

Note:

If the application is not listed in the App Marketplace, push permissions are restricted. Official messages cannot be sent through the Vivo official website's Web Interface or API backend. However, test messages can be sent to set test devices through the API backend for testing.

Step 1: Register a Vivo Developer Account

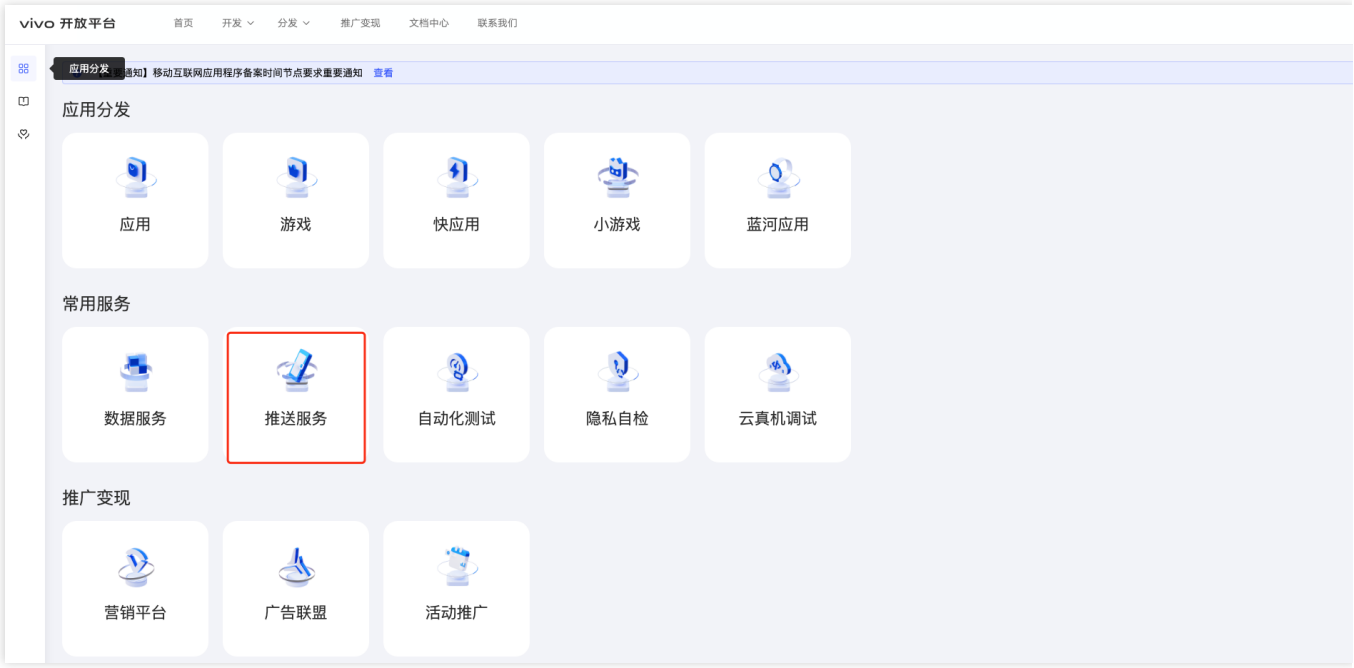
Enter Vivo Open Platform, register a Vivo Developer Account. For details, see Vivo Enterprise Developer Account Registration.

Step 2. Create an application

Enter **Vivo Open Platform**, click **Distribution** > **Application Distribution** > **App Store** > **Upload Application** to create your application.

Step 3:Enable Push Notification

Enter the Management Center and click **Push Service > Push Notification Application** to apply for enabling push for the newly created application.

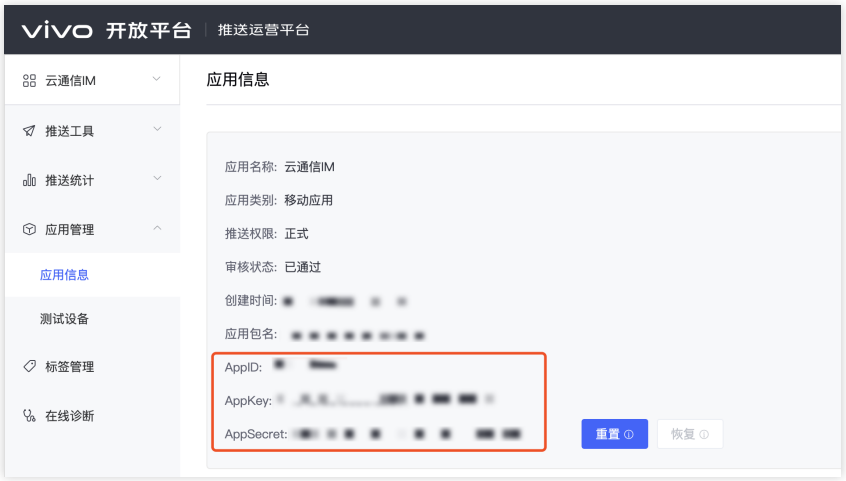
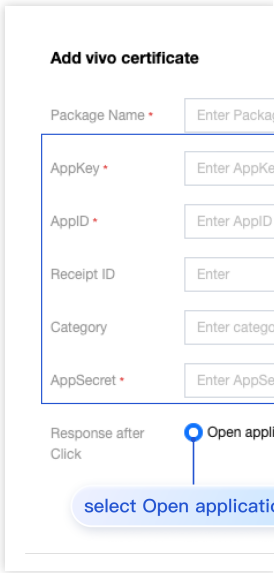


Step 4: Obtain Application Information

Go to the Push Operation Platform and click **Application Management > Application Information** to obtain application information.

Step 5: Add Push Certificate

Log in to Tencent Cloud Chat Console, click **Push** > **Access Settings** to add vendor push certificates, and configure the obtained vendor's AppId, AppKey, AppSecret, and other parameters to the added push certificate.

Vendor Push Platform	Configuring in the IM console
	**Note:** **Specified in-app page** link dispatching event listening fo configuration of in-app page r 

For receipt configuration, refer to: Message Delivery Statistics Configuration > VIVO

Step 1: Register a Meizu Developer Account

Register a Meizu developer account, see Developer Registration for details.

Step 2: Create an Application

1. Click **Console** >**Flyme Push**.

2. After entering the application information, create the application.

Note:

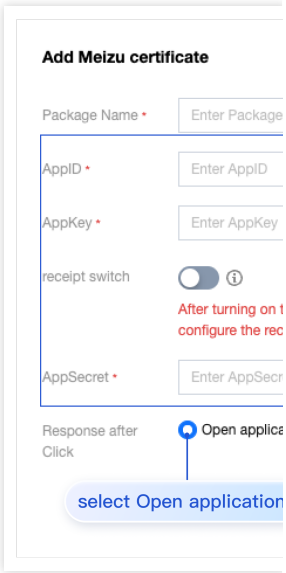
The application package name should be consistent with the plugin application package name.

Step 3: Obtain Application Information

In the application list, click **Open Application**. Enter the configuration management page to obtain application information.

Step 4: Add Push Certificate

Log in to Tencent Cloud Chat Console, click **Push** > **Access Settings** feature bar to add various manufacturer's push certificates, and configure the obtained parameters such as AppId, AppKey, AppSecret to the added push certificate.

Vendor Push Platform	Configuring in the IM console
	**Note:** **Specified in-app page** link c dispatching event listening for configuration of in-app page n 

For receipt configuration, refer to: Message Delivery Statistics Configuration > Meizu

Step 1. Register an HONOR Developer Account

Register an HONOR Developer Account, for details see Developer Registration.

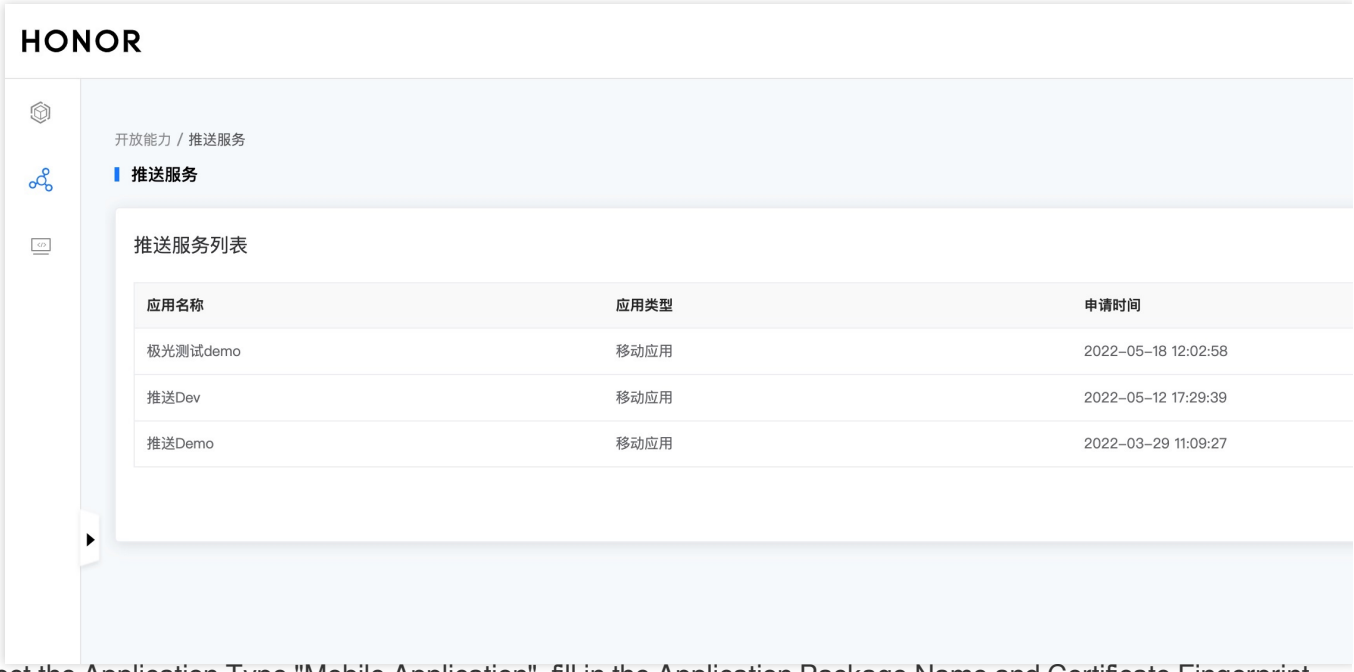
Step 2: Enter the Management Center Page.

Step 3: Enter the Push Service List

Click **Push Service** to access the Push Service list page.

Step 4: Create an Application

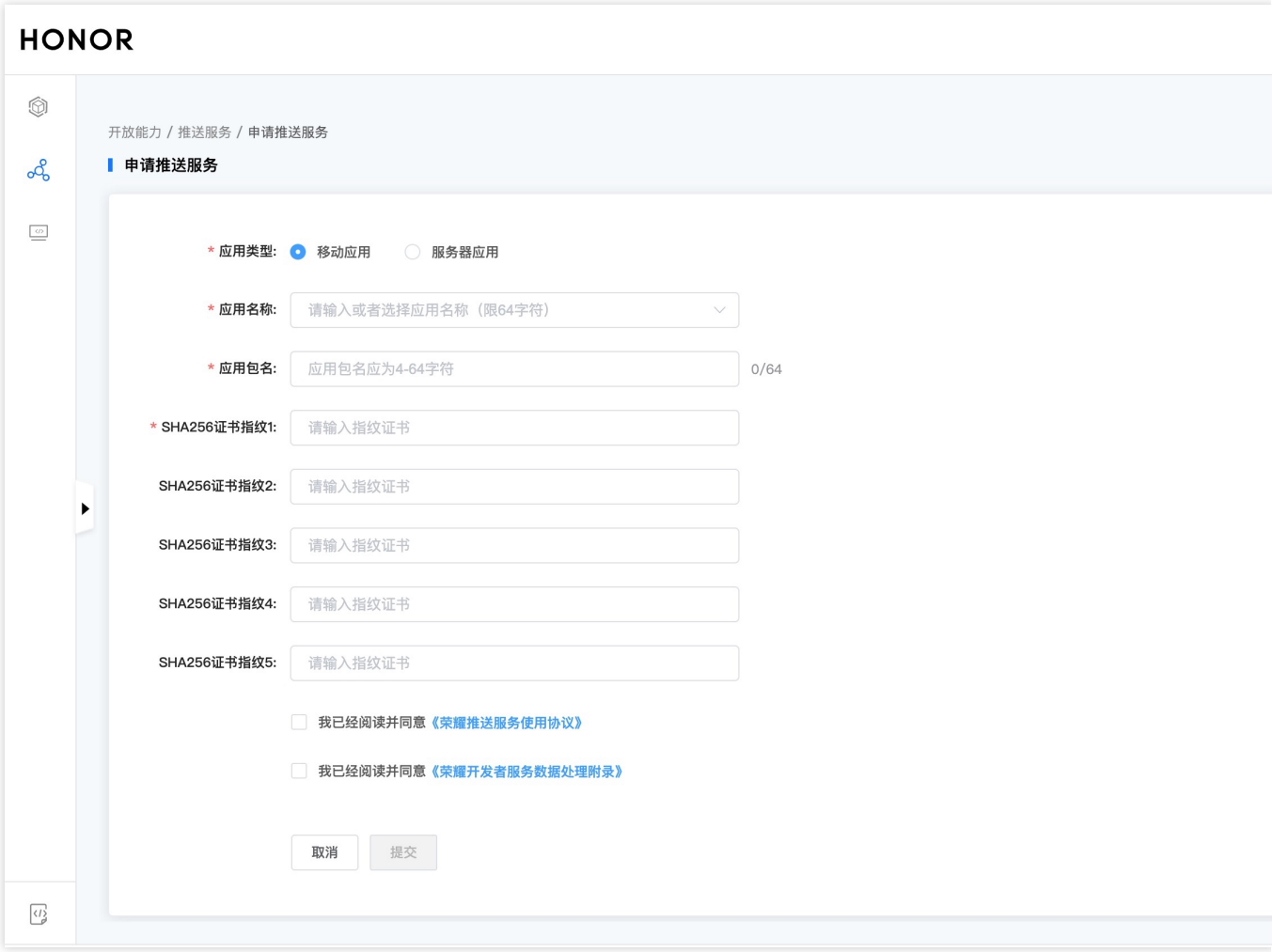
1. Click **Apply for Push Services** to enter the application submission page.



2. Select the Application Type "Mobile Application", fill in the Application Package Name and Certificate Fingerprint, agree to the Push Service Agreement and Data Processing Appendix, then click **Submit**.

Note:

You must add the packaged SHA256 Certificate Fingerprint, which should match the SHA256 Certificate Fingerprint of your own packaging certificate.

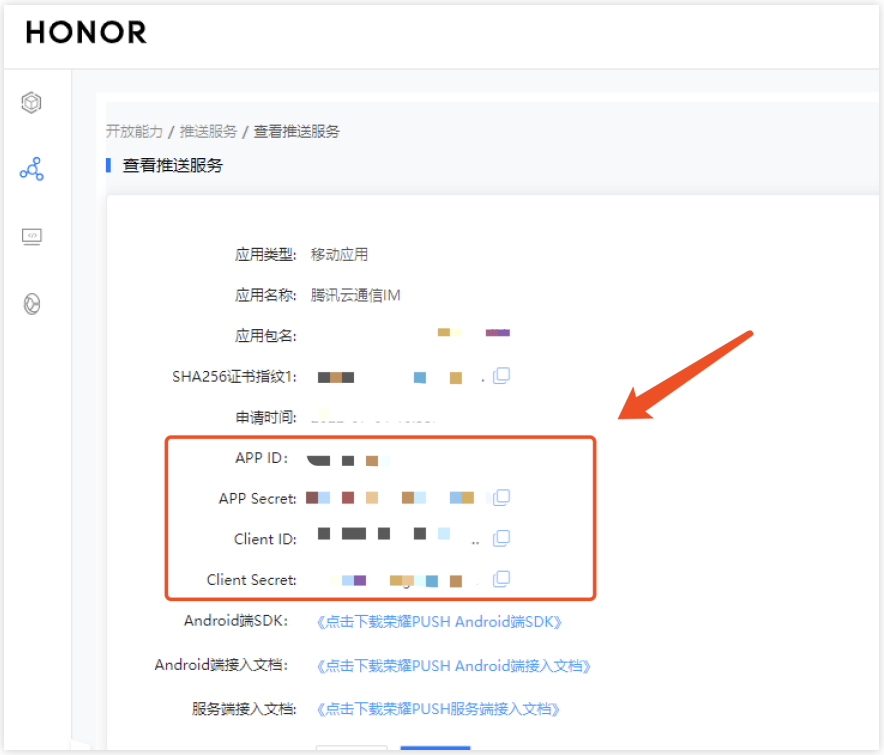
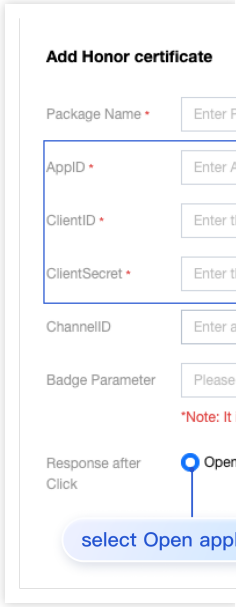


Step 5: Obtain Application Information

In the **Push Service** list, click **View** to obtain application information.

Step 6: Add Push Certificate

Log in to Tencent Cloud Chat Console, click **Push** > **Access Settings**, add vendor push certificates, and configure the obtained vendor's AppID, AppKey, AppSecret, and other parameters to the added push certificate.

Vendor Push Platform	Configuring in the IM cons
	**Note:** **Specified in-app page** li dispatching event listenin configuration of in-app pa 

Google FCM is currently under development, please stay tuned~~

Before integrating the TIMPush component, you need to first apply for an APNs push certificate from Apple, then upload the push certificate to the IM Console. Afterwards, you can follow the Quick Integration steps to integrate.

There are currently two mainstream types of certificates for Apple, the p12 certificate and the p8 certificate. Each type of certificate has its advantages and disadvantages, and you can choose one of them according to your needs.

	Certificate Type	Validity Period and Management	Security	Dynamic Island
P12 Certificate	A P12 certificate is a binary file that contains both a public and a private key, used for certificate-based authentication. It	P12 certificates usually have a validity period of one year, after which they need to be regenerated and deployed. Each application requires a	Certificate: The P12 certificate uses certificate-based authentication, requiring the private key to be stored on the server. This may increase the security risk, as the	Not supported

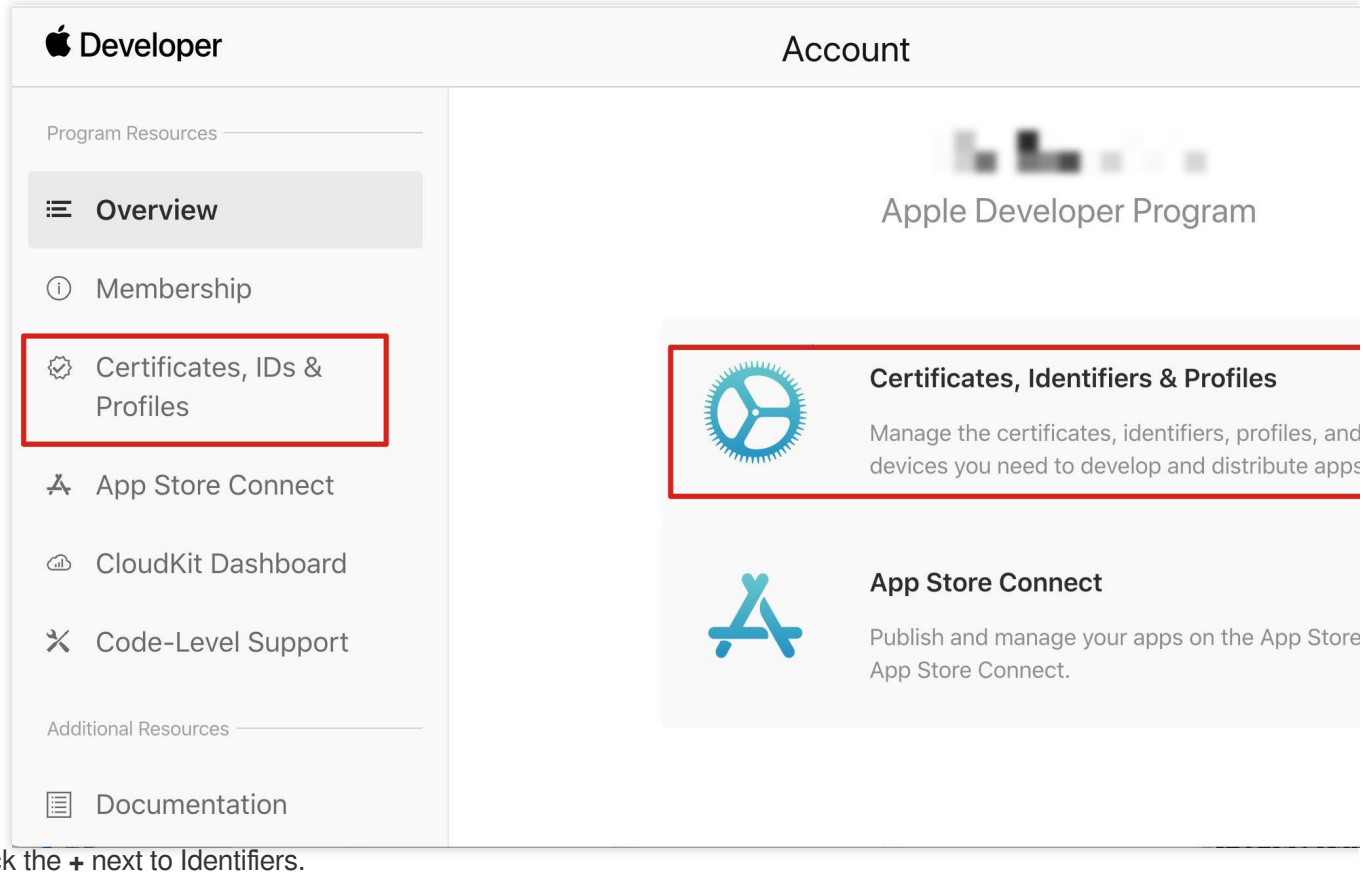
	bundles the public key certificate and the private key into one file, with an extension of .p12 or .pfx.	separate P12 certificate to handle push notifications.	private key could be accessed by unauthorized users.	
P8 Certificate	A P8 certificate is an Authorization Key, used for token-based authentication. It is a text file containing a private key, with an extension of .p8.	P8 certificates do not have an expiration date, so you do not need to worry about certificate expiration. Moreover, using a P8 certificate can simplify certificate management, as you can use one P8 certificate to provide push notification services for multiple applications.	P8 certificates use token-based authentication, meaning your server periodically generates a JSON Web Token (JWT) to establish a connection with APNs. This method is more secure, as it does not require storing a private key on the server.	Supports Dynamic Island Push

1. Using a p12 certificate (traditional push certificate)

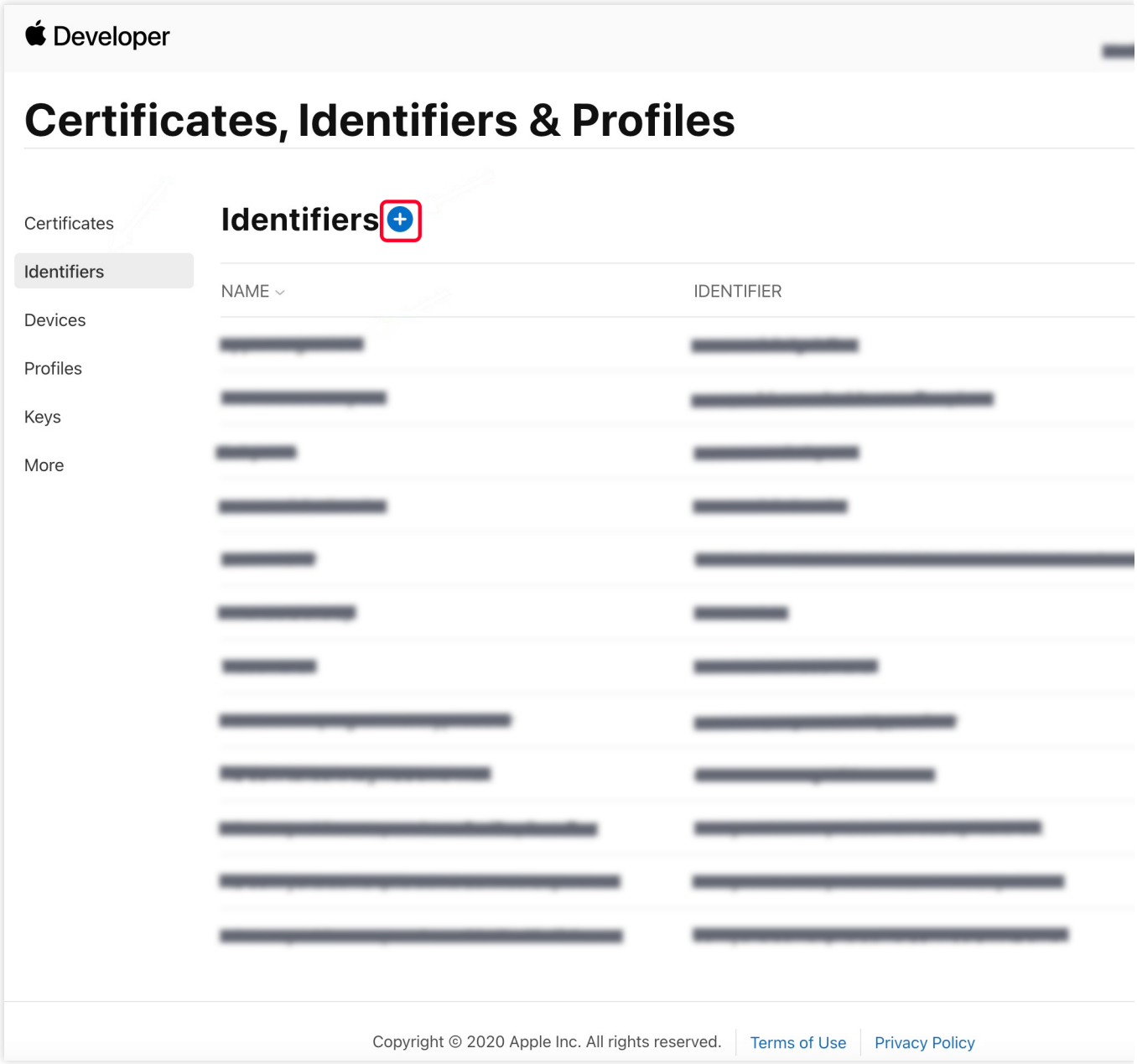
Step 1: Apply for an APNs certificate

Enable remote push for the app

1. log in to Apple Developer Center website, click **Certificates, Identifiers & Profiles** or the sidebar's **Certificates, IDs & Profiles**, enter the **Certificates, IDS & Profiles** page.



2. click the + next to Identifiers.

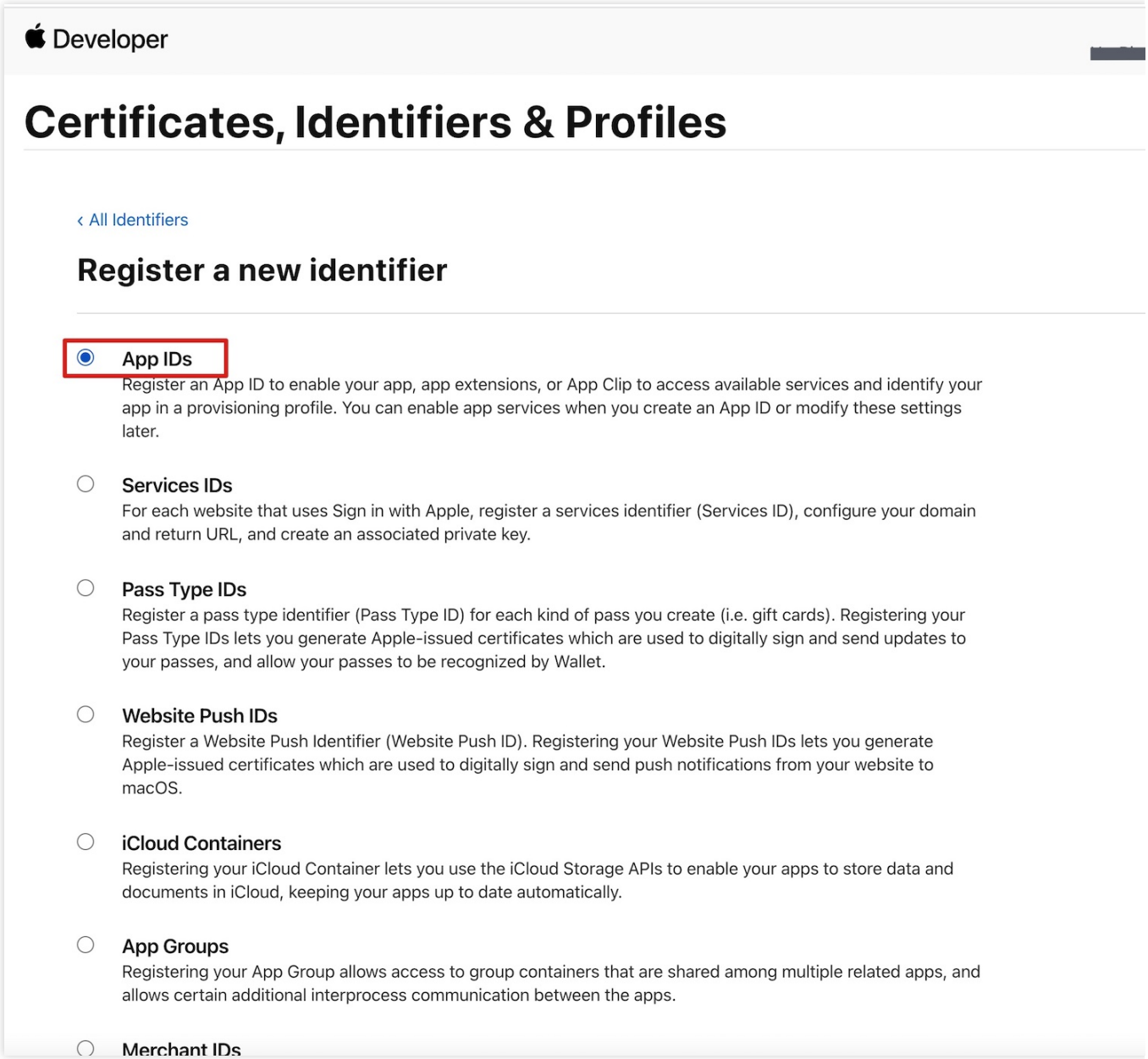


3. You can follow the steps below to create a new AppID or add a `Push Notification Service` to your existing AppID.

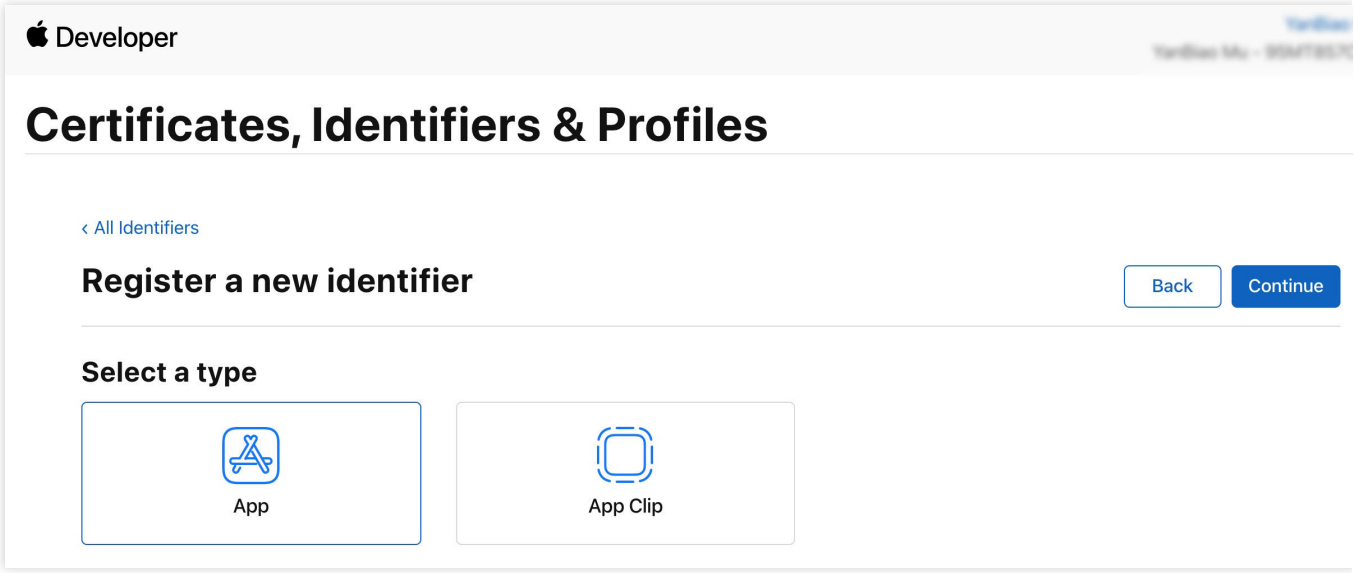
Note:

Your App's `Bundle ID` cannot use the wildcard `*`, otherwise, the remote push service cannot be used.

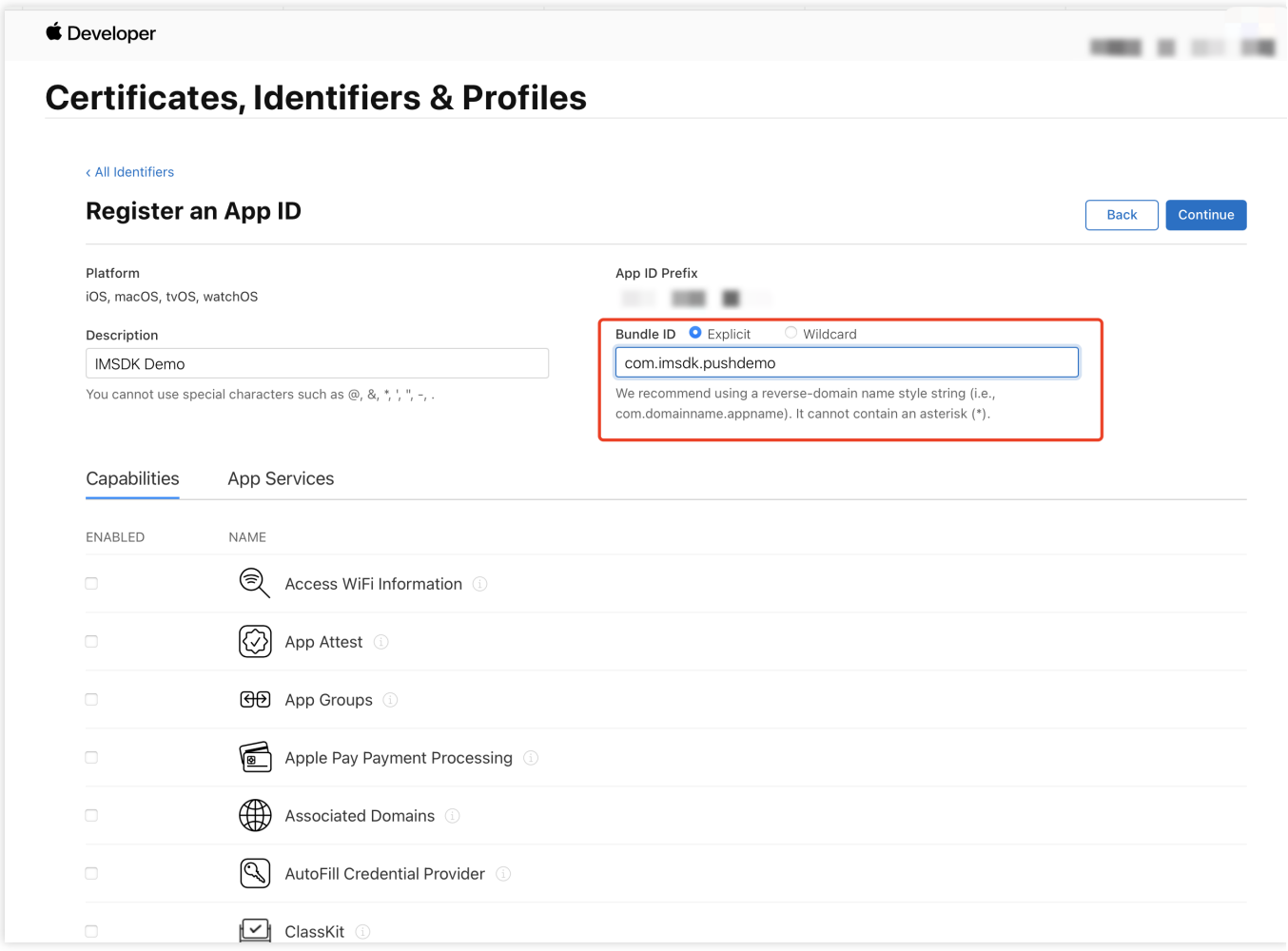
4. Check the **App IDs** box, click **Continue** to proceed to the next step.



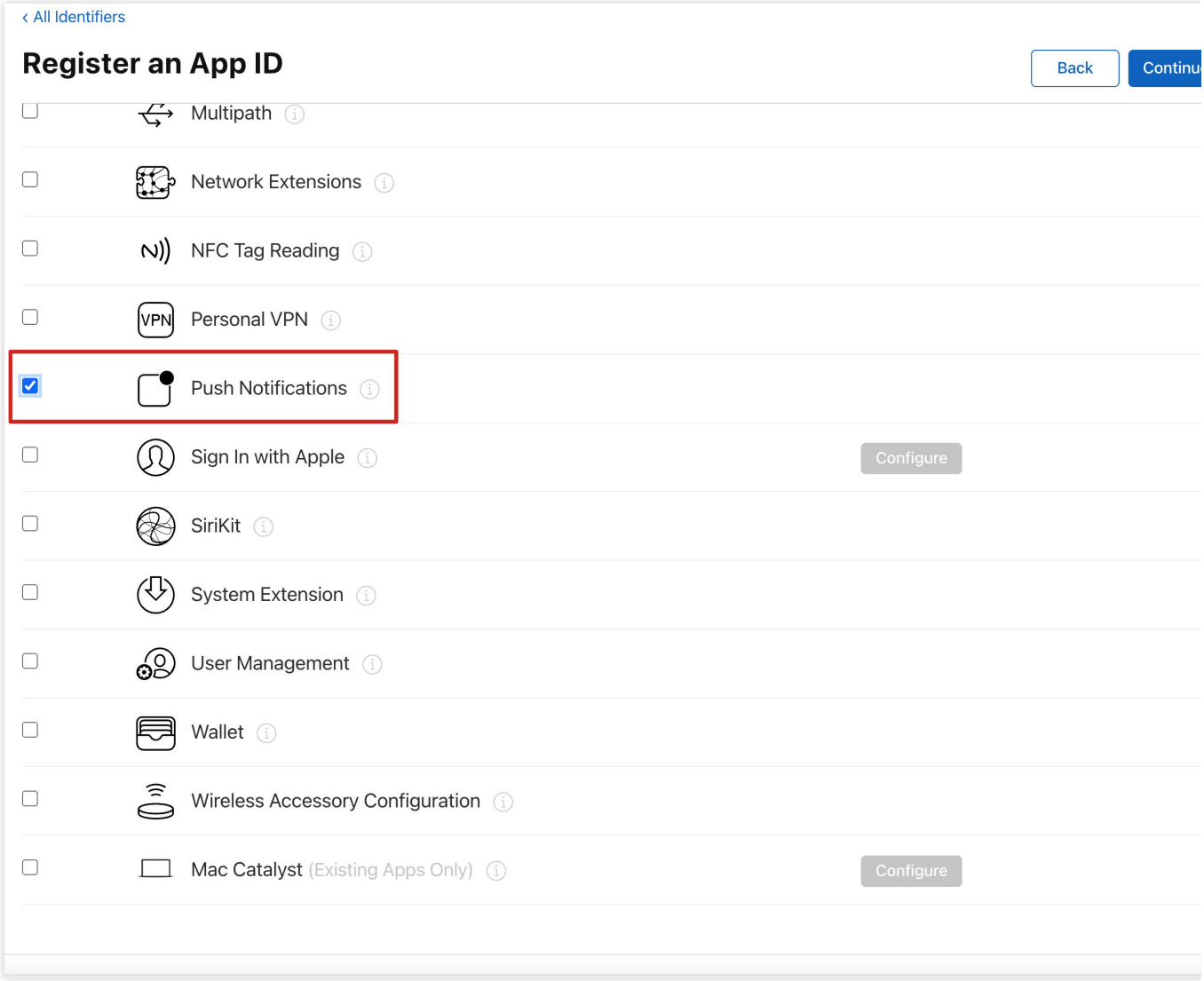
5. Select **App**, click **Continue** to proceed to the next step.



6. Configure the `Bundle ID` and other information, click **Continue** to proceed to the next step.

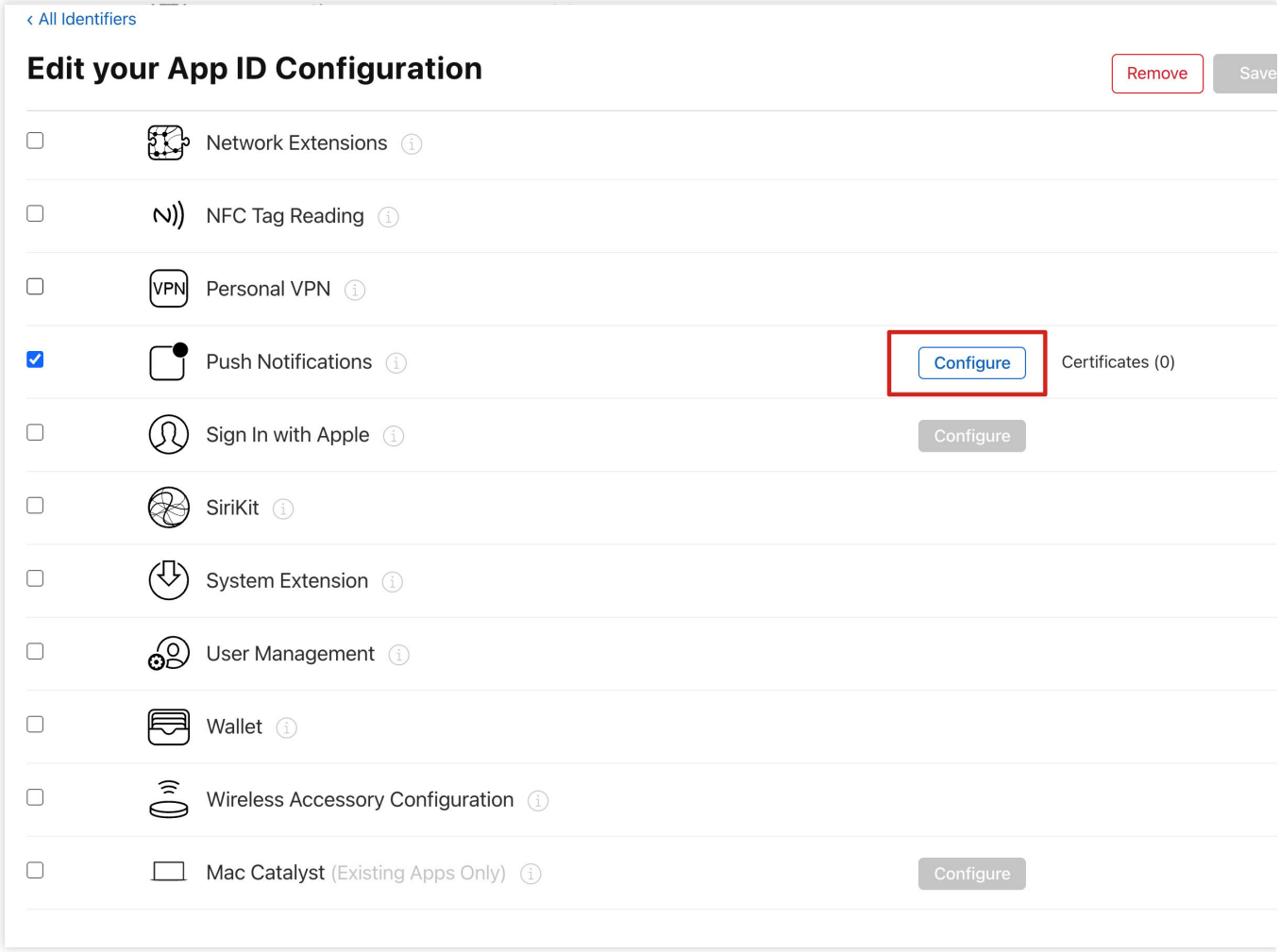


7. Check the **Push Notifications** box to enable the remote push service.

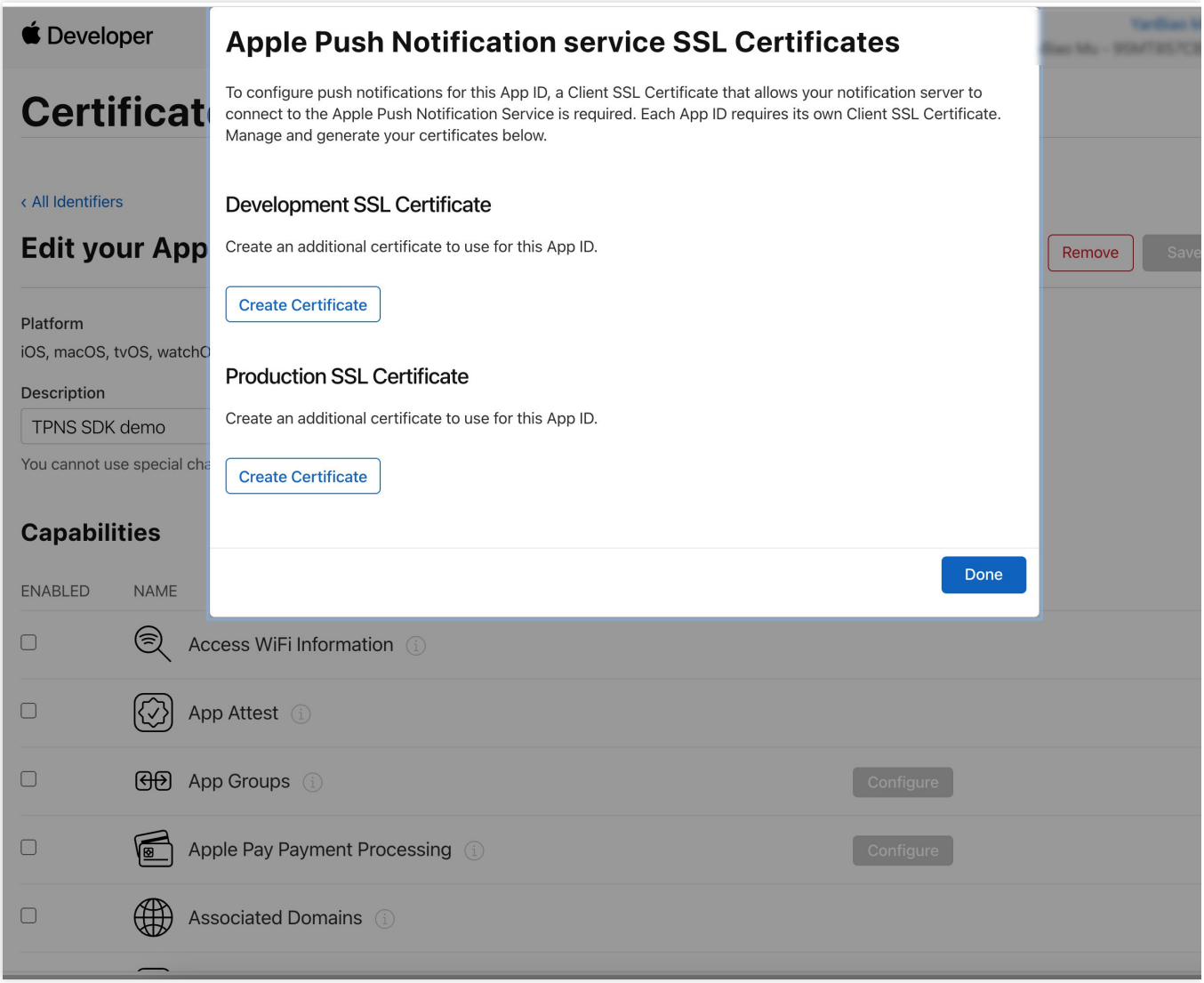


Certificate Generation

1. Select your AppID and choose **Configure**.



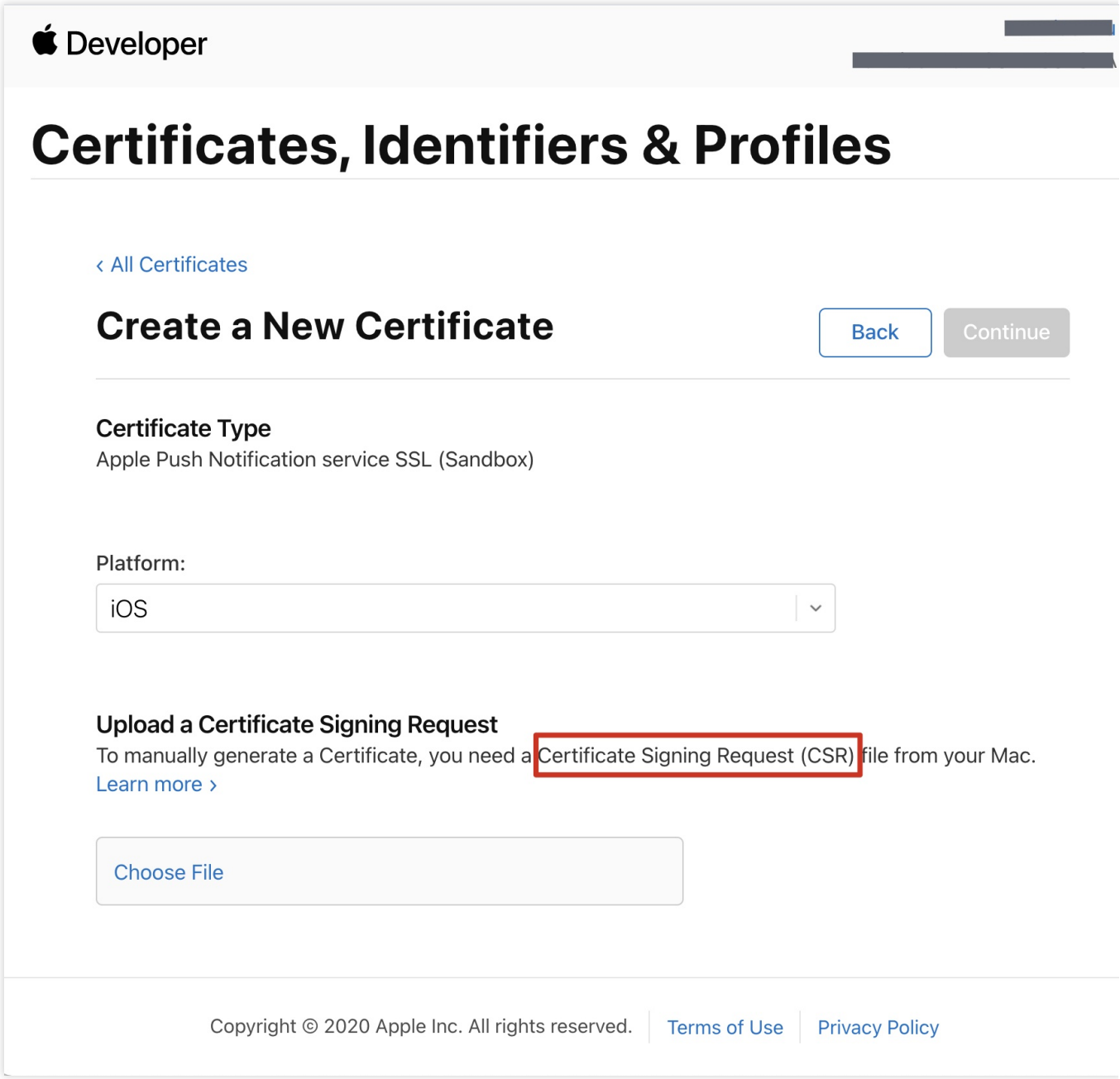
2. In the **Apple Push Notification service SSL Certificates** window, there are two `SSL Certificates` for the development environment (Development) and the production environment (Production), as shown below:



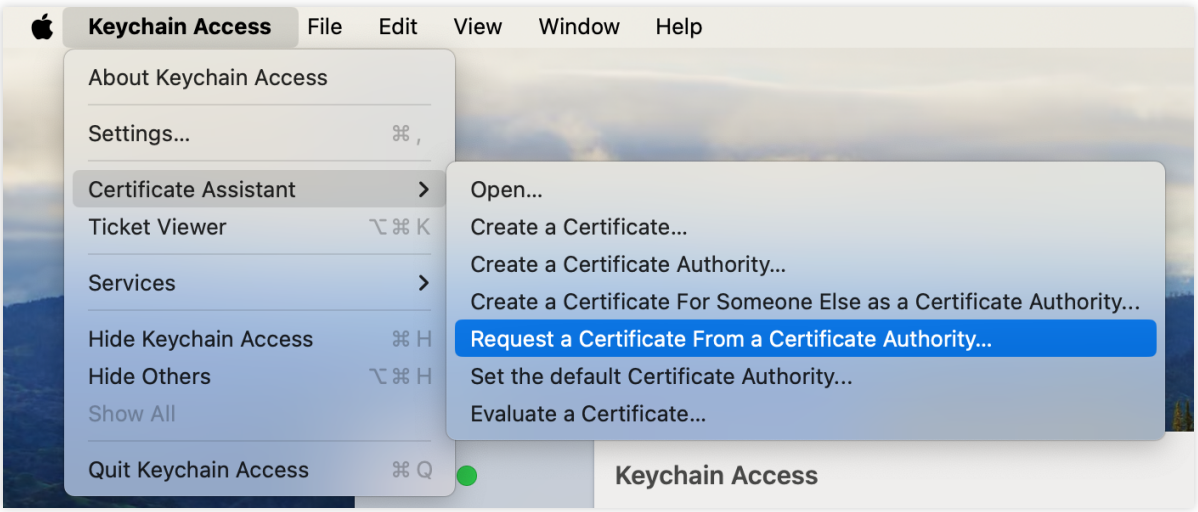
3.

We

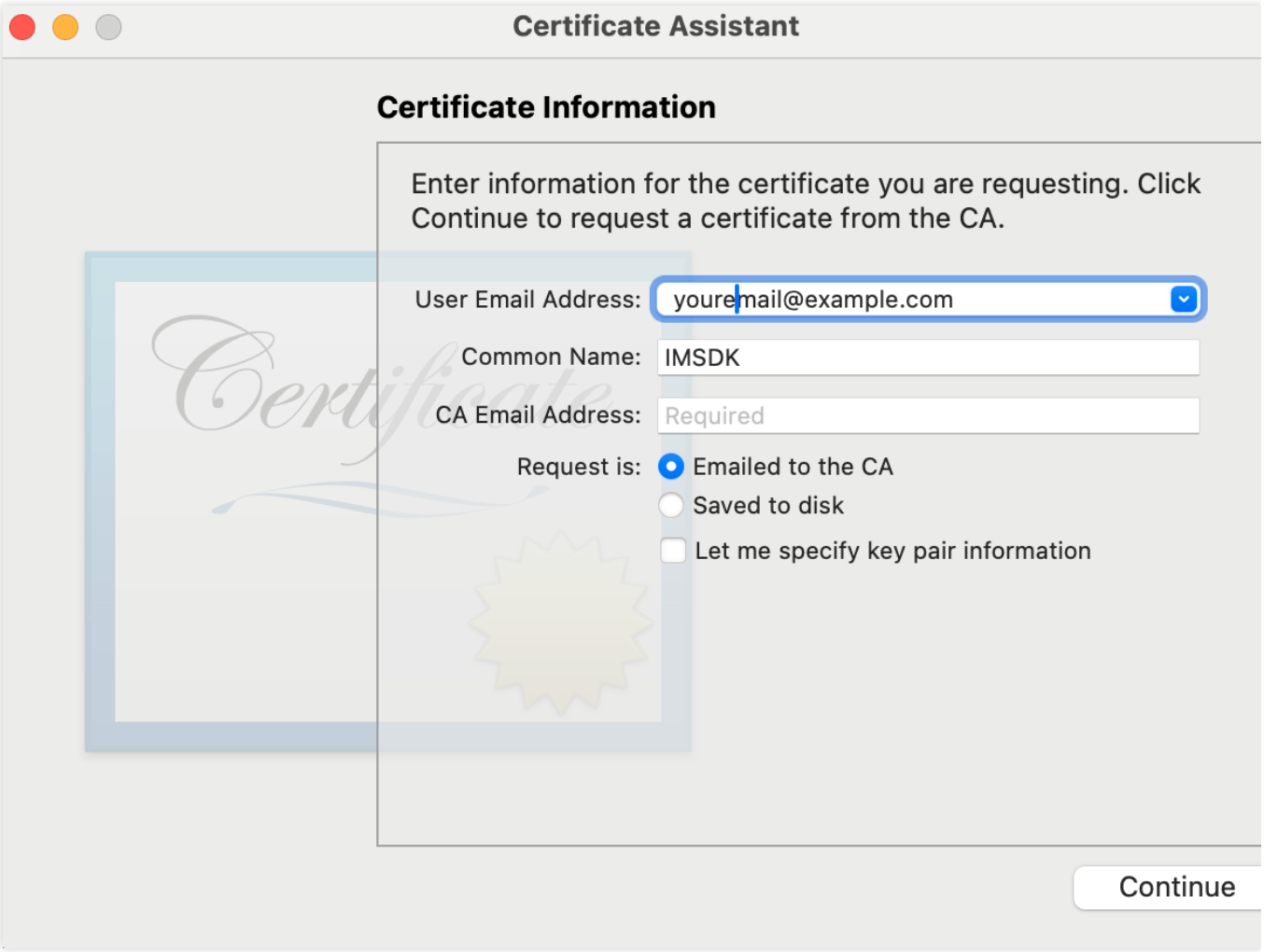
first select the **Create Certificate** for the Development environment, the system will prompt us that we need a Certificate Signing Request (CSR).



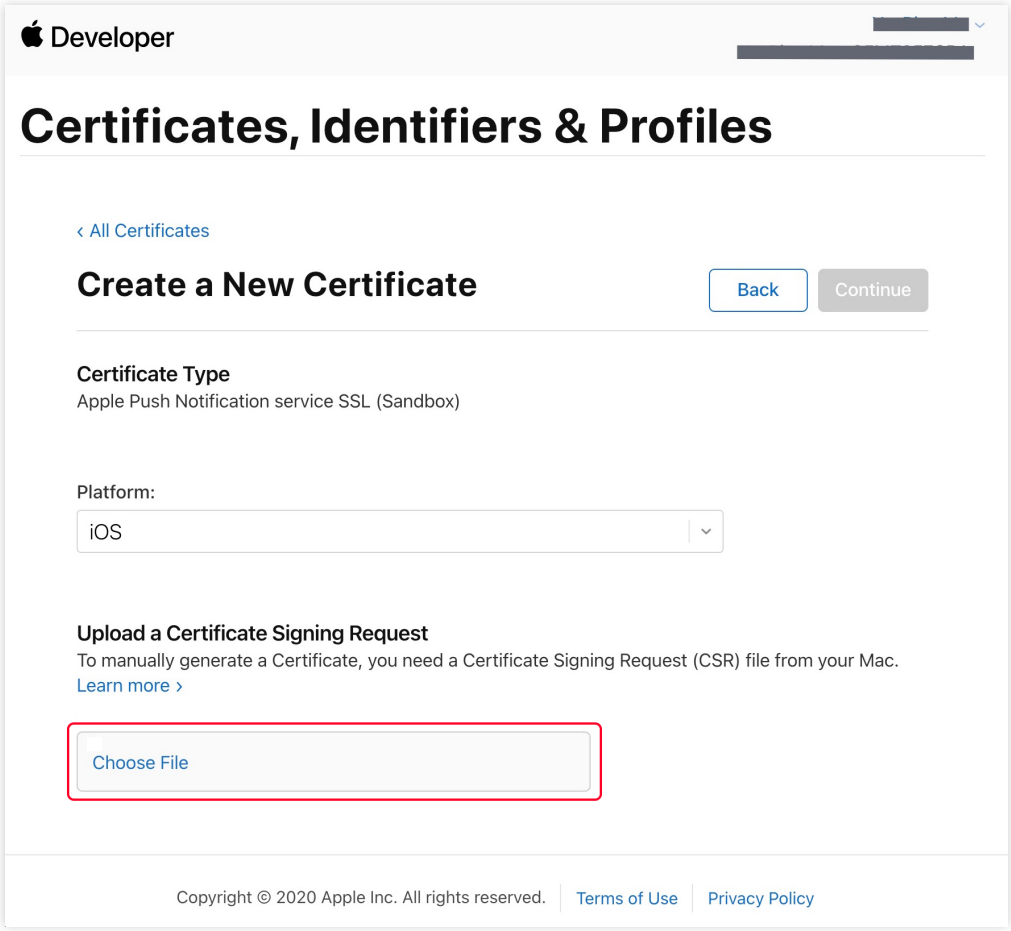
4. On a Mac, open **Keychain Access tool**, in the menu select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority** (`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`).



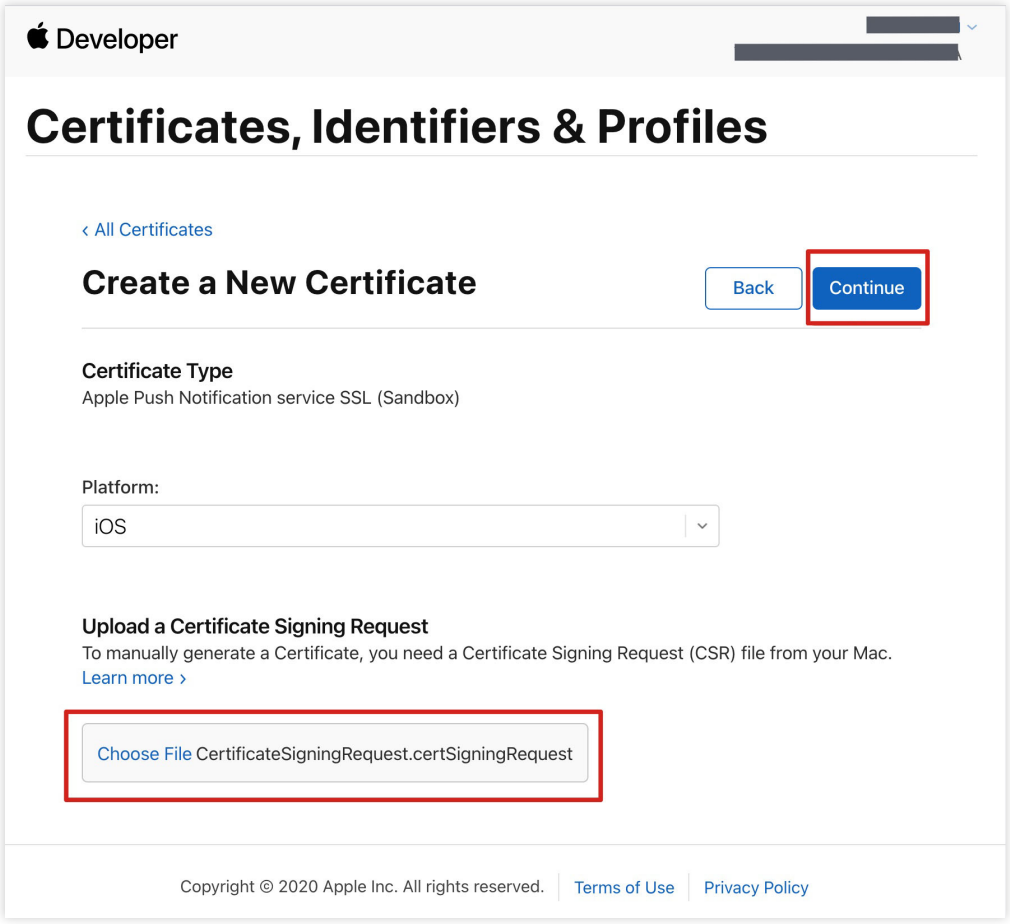
5. Enter your email address, Common Name (your name or company name), select **Save to disk**, click **continue**, the system will generate a `*.certSigningRequest` file.



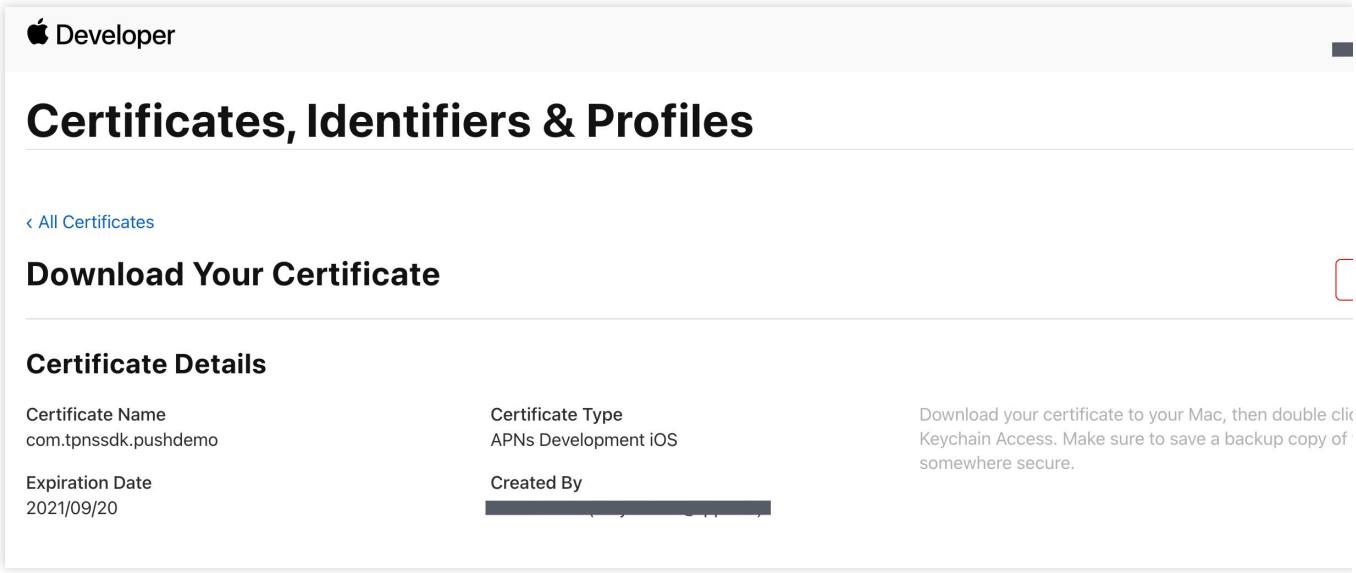
6. Go back to the page on the Apple Developer website mentioned in Step 3, click **Choose File** to upload the generated `*.certSigningRequest` file.



7. Click **Continue** to generate the push certificate.



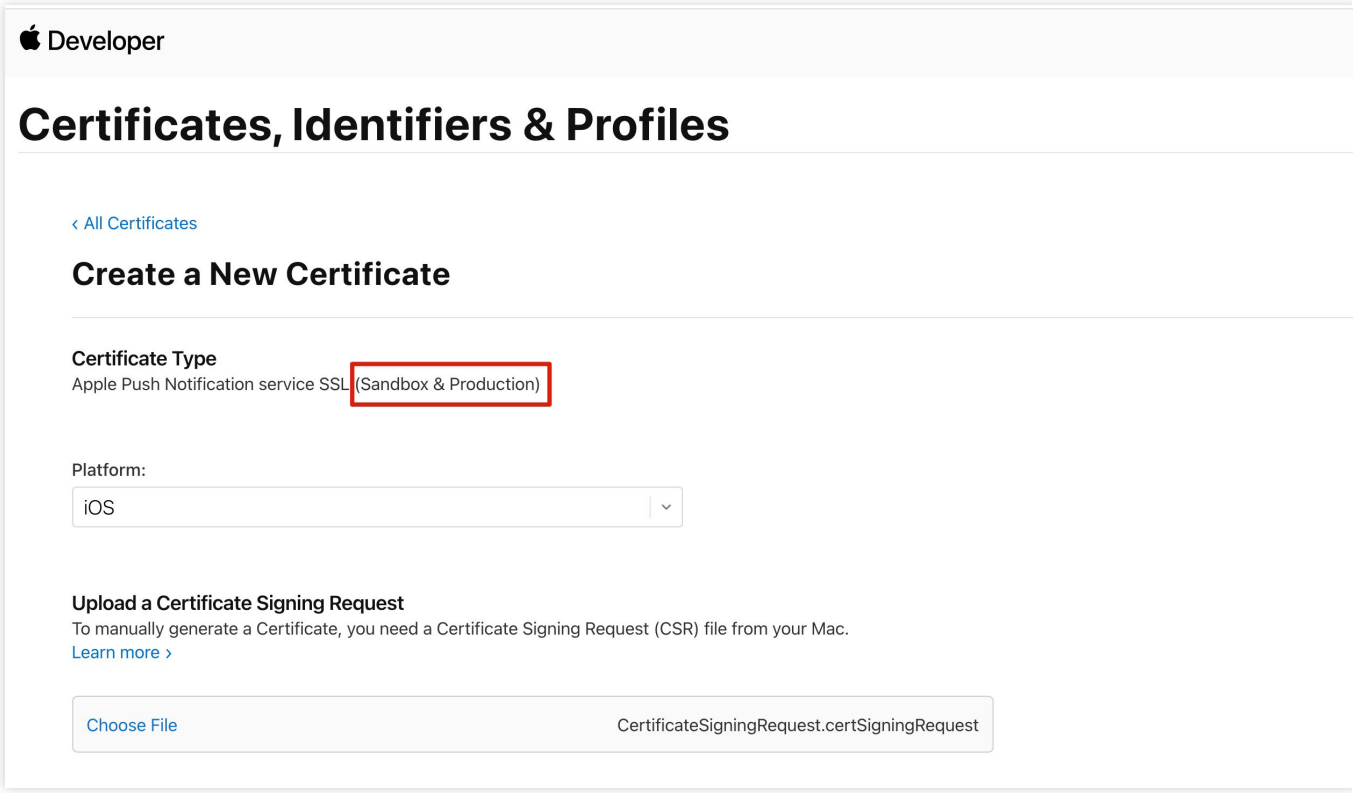
8. click **Download** to download the `Development SSL Certificate` to your local environment.

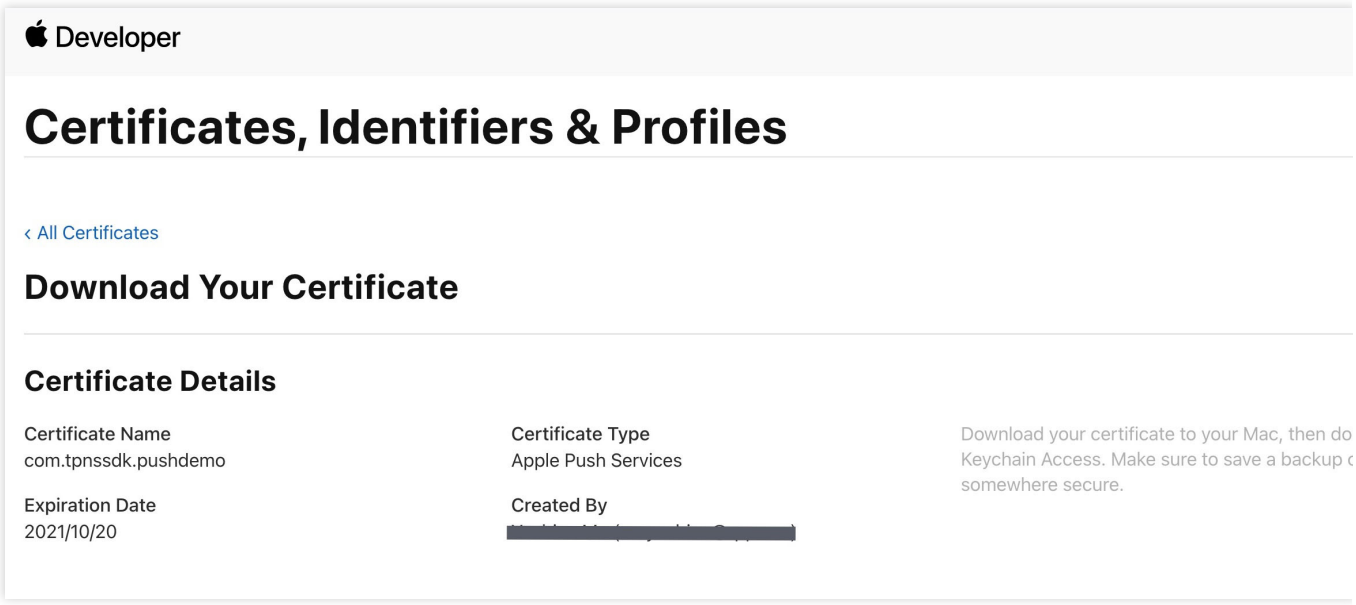


9. Repeat steps 1 - 8 above to download the `Production SSL Certificate` for the production environment to your local machine.

Note

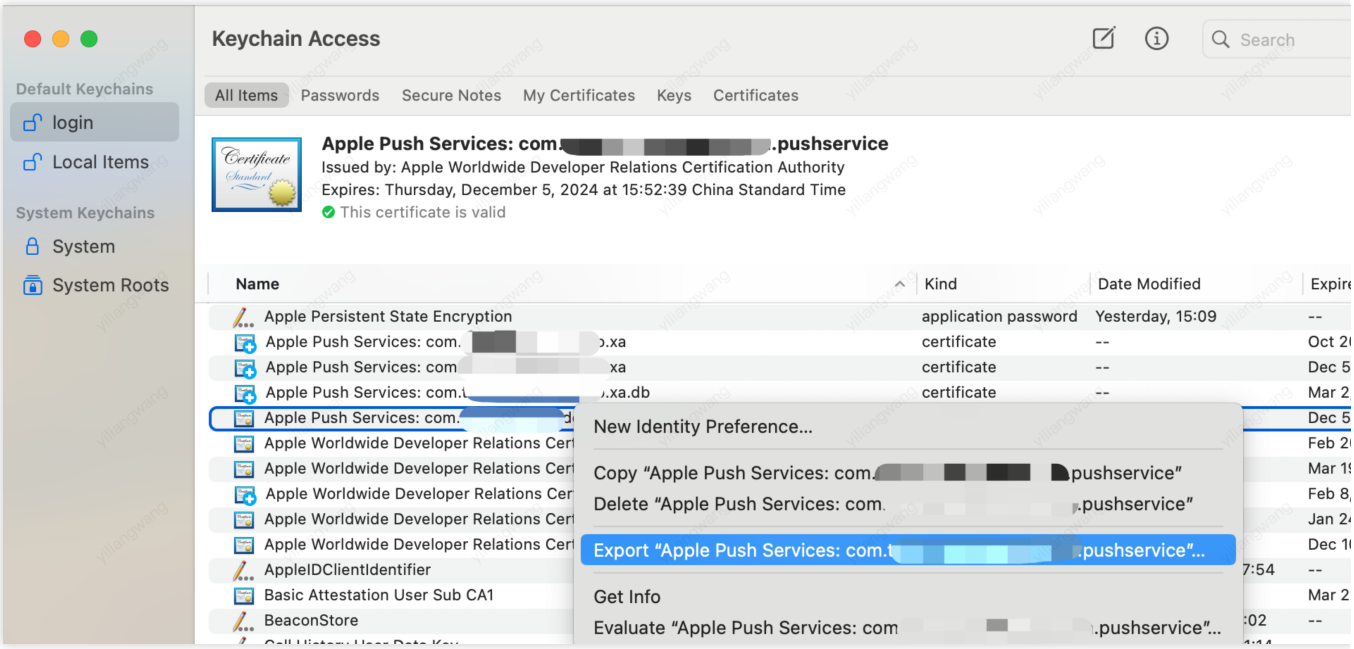
The certificate for the production environment is actually a combined certificate of Development (Sandbox) + Production, and it can be used as a certificate for both the development and production environments.





10. Double-click the downloaded `SSL Certificate` for the development and production environments. The system will import it into the keychain.

11. Open the Keychain App, go to **log in to** > **My Certificates**, right-click to export the newly created `Apple Development IOS Push Services` and `Apple Push Services` for the development and production environments as `p12` files respectively.



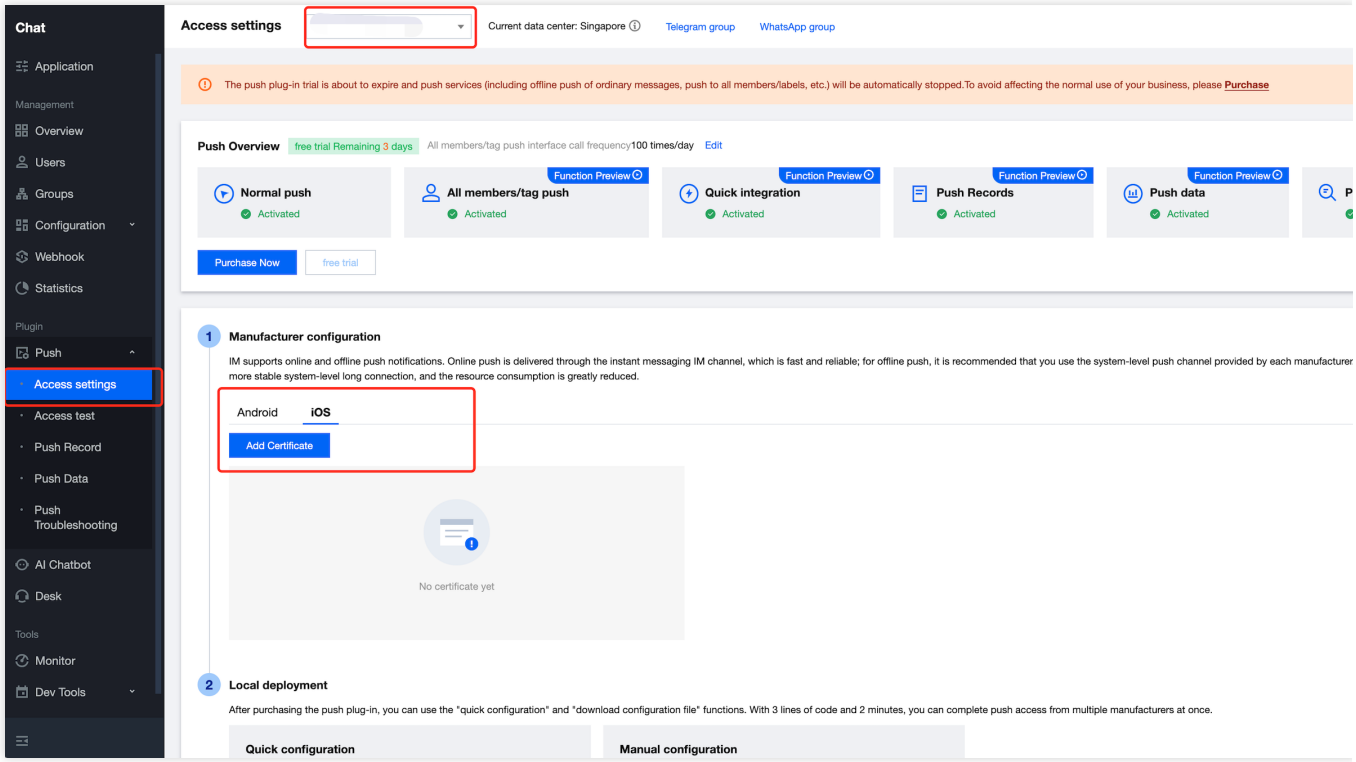
Note

Set a password when saving the `.p12` file.

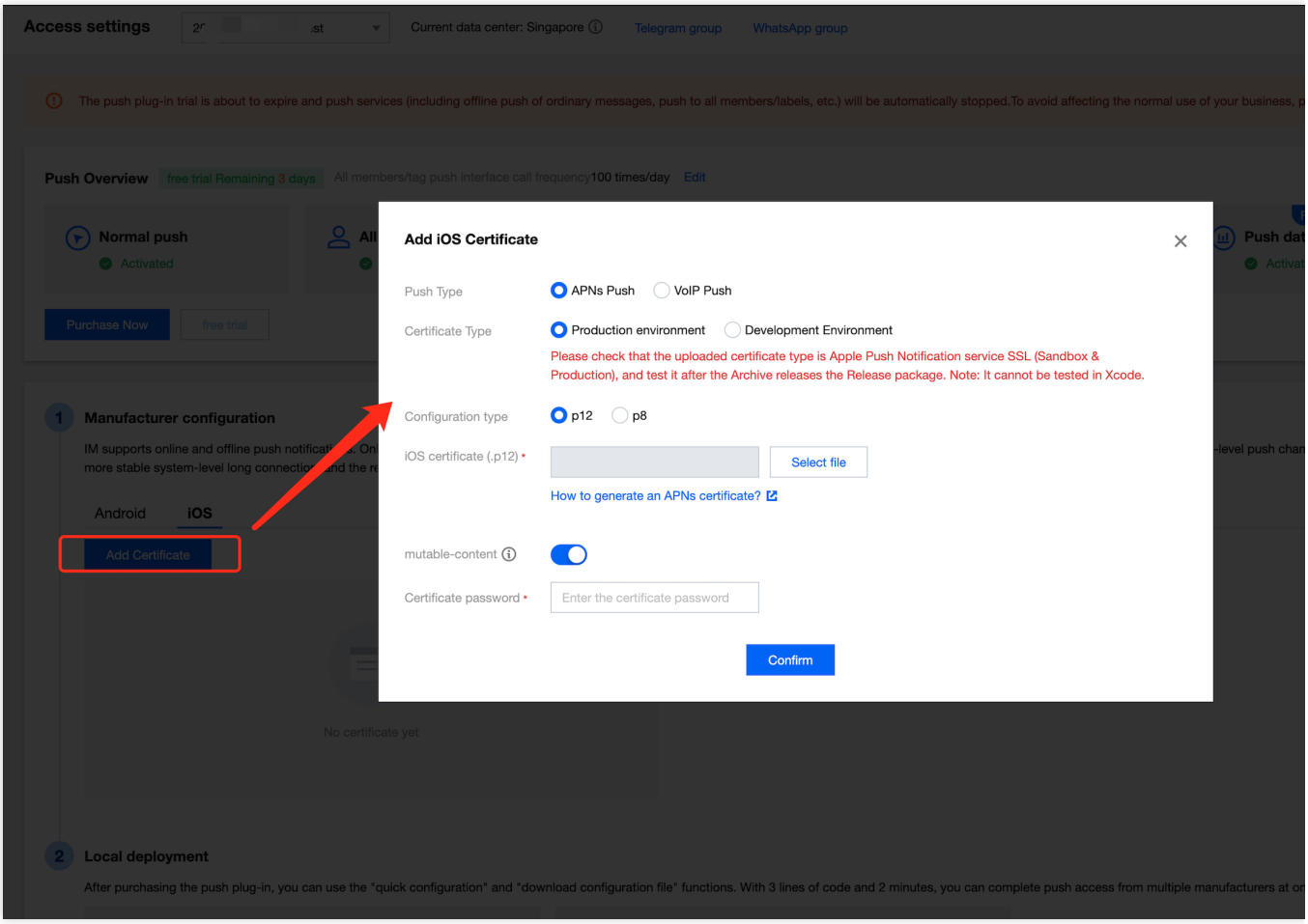
Step 2: Upload the certificate to the console

1. Log in to the Chat Console.

2. Click Plugin Service-Push-Access Settings to enter the access settings page



3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the certificate type, upload the iOS Certificate (.p12), set the certificate password, and click **Confirm**.



Note:

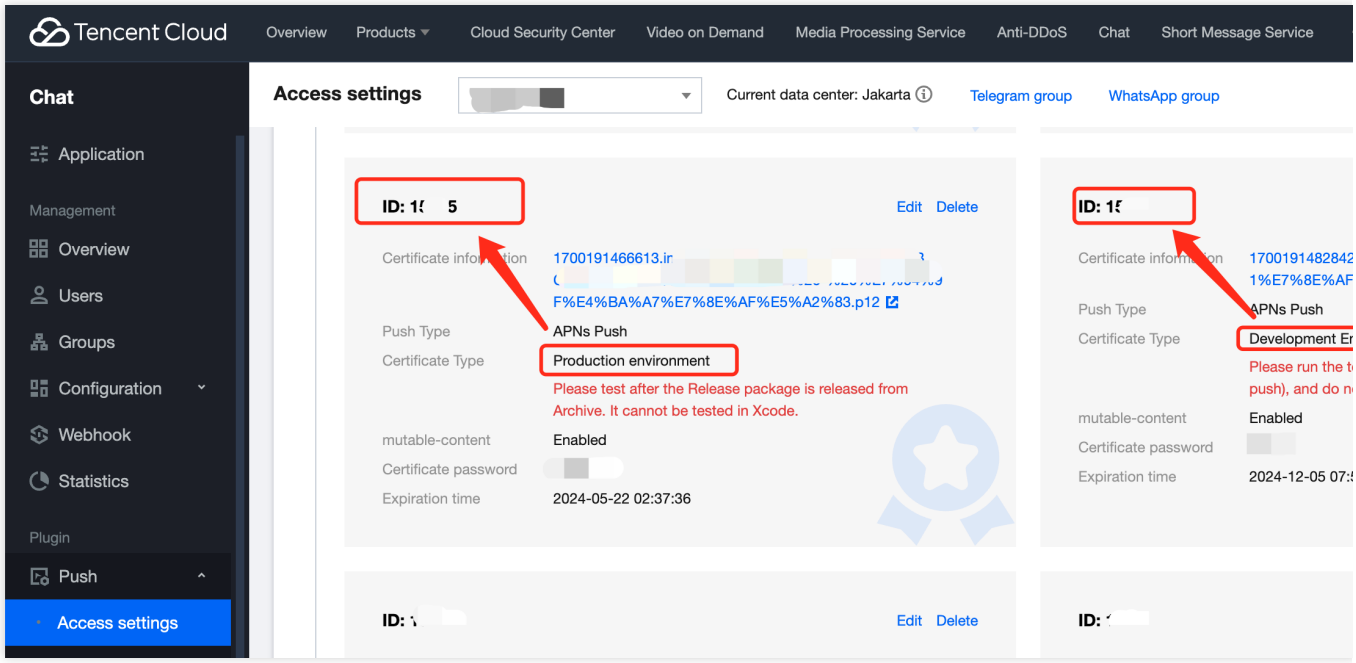
We recommend naming the uploaded certificate in English (special characters such as brackets are not allowed).

You need to set a password for the uploaded certificate. Without a password, push notifications cannot be received.

For an app published on App Store, the environment of the certificate must be the production environment. Otherwise, push notifications cannot be received.

The uploaded .p12 certificate must be your own authentic and valid certificate.

5. After the pending certificate information is generated, record the certificate's ID.

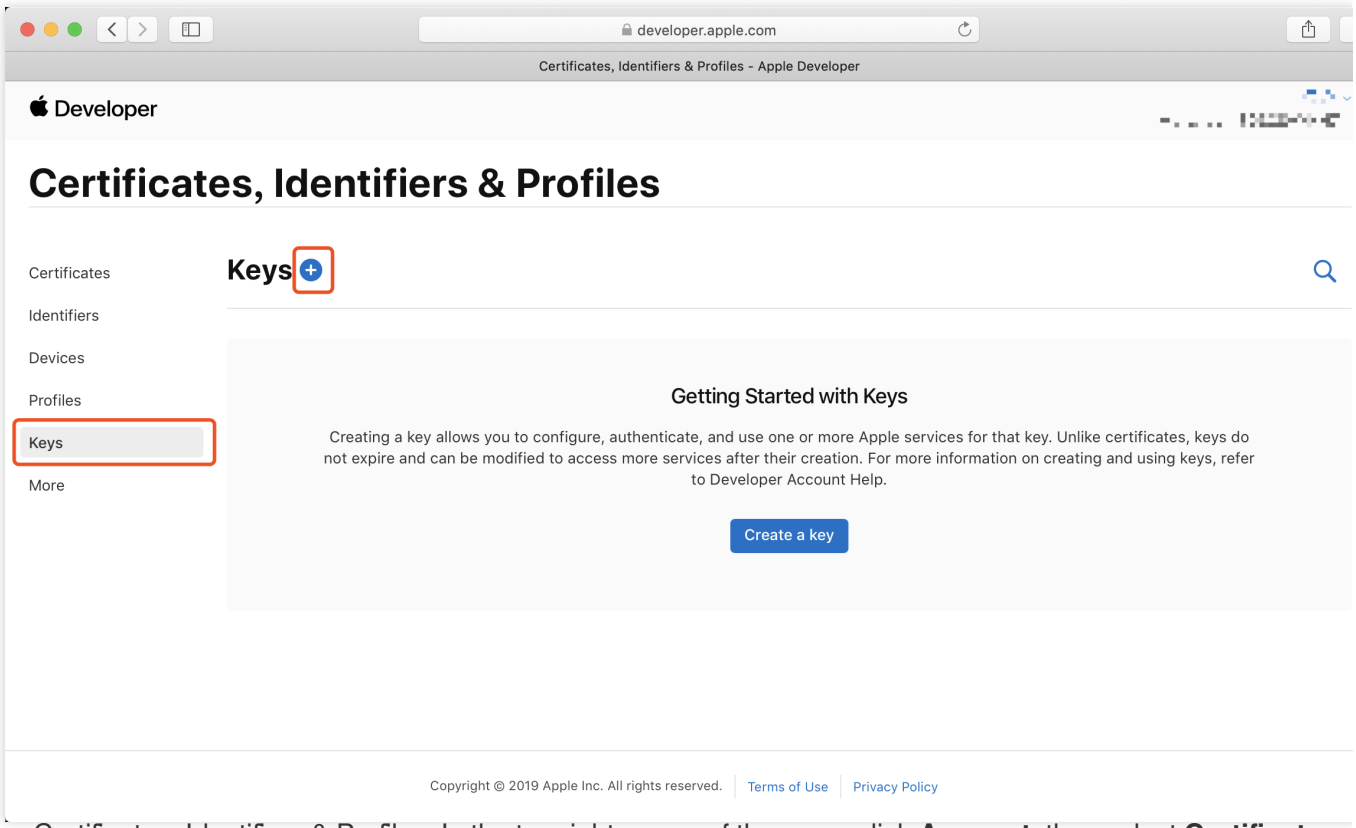


2. Using a p8 certificate (supports Dynamic Island push notifications)

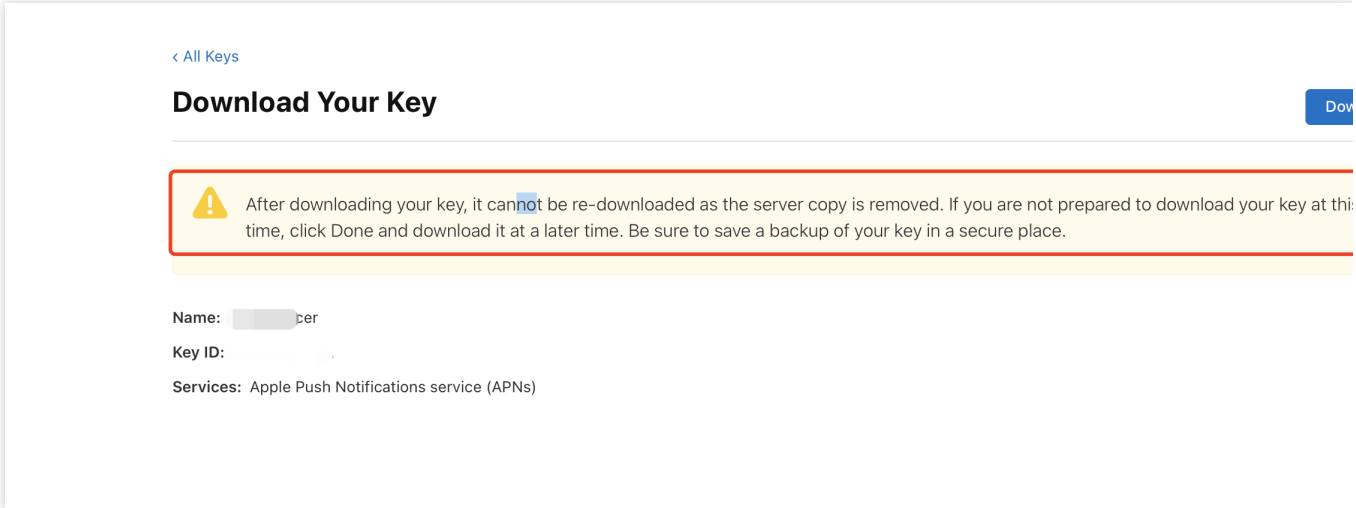
p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a p8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications. In addition, p8 certificates support Dynamic Island push notifications.

Step 1: Apply for an APNs certificate

To create a p8 certificate file, first log in to Apple Developer Center.



1. Enter Certificates, Identifiers & Profiles: In the top right corner of the page, click **Account**, then select **Certificates, Identifiers & Profiles** from the dropdown menu.
2. To create a new App ID: in the left-hand menu, click **Identifiers**, then click the **+** on the right to create a new App ID. Fill in the relevant information and click **Continue**.
3. To create a new key: in the left-hand menu, click **Keys**, then click the **+** on the right to create a new key. Enter the name of the key, then check **Apple Push Notifications service (APNs)** and click **Continue**.



Confirm and generate the key: On the confirmation page, verify your key information, then click **Register**. Next, you'll see a page prompting you to download the key. Click **Download** and save the generated .p8 file to your computer.

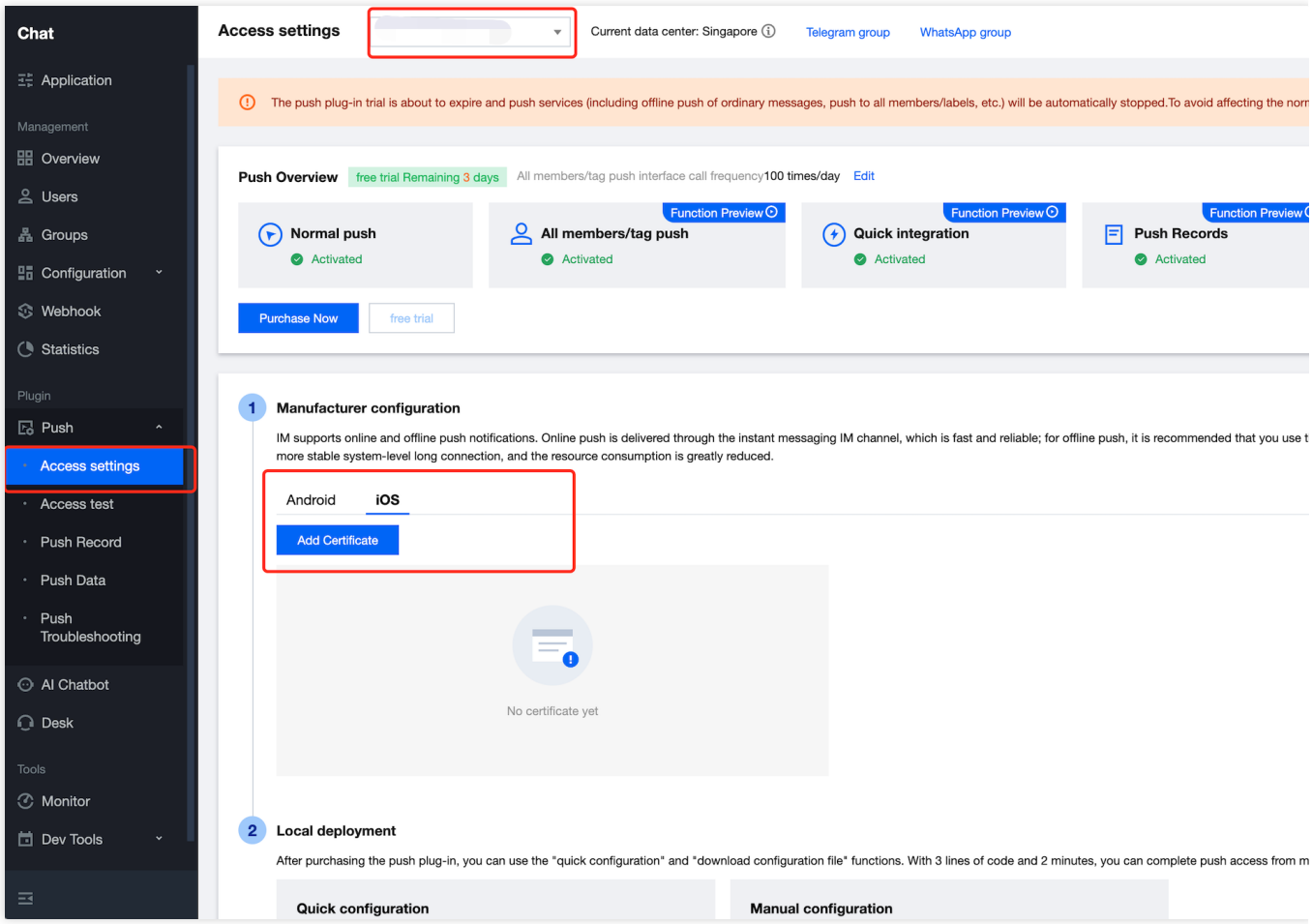
Note:

The p8 certificate can only be downloaded once; please save it properly.

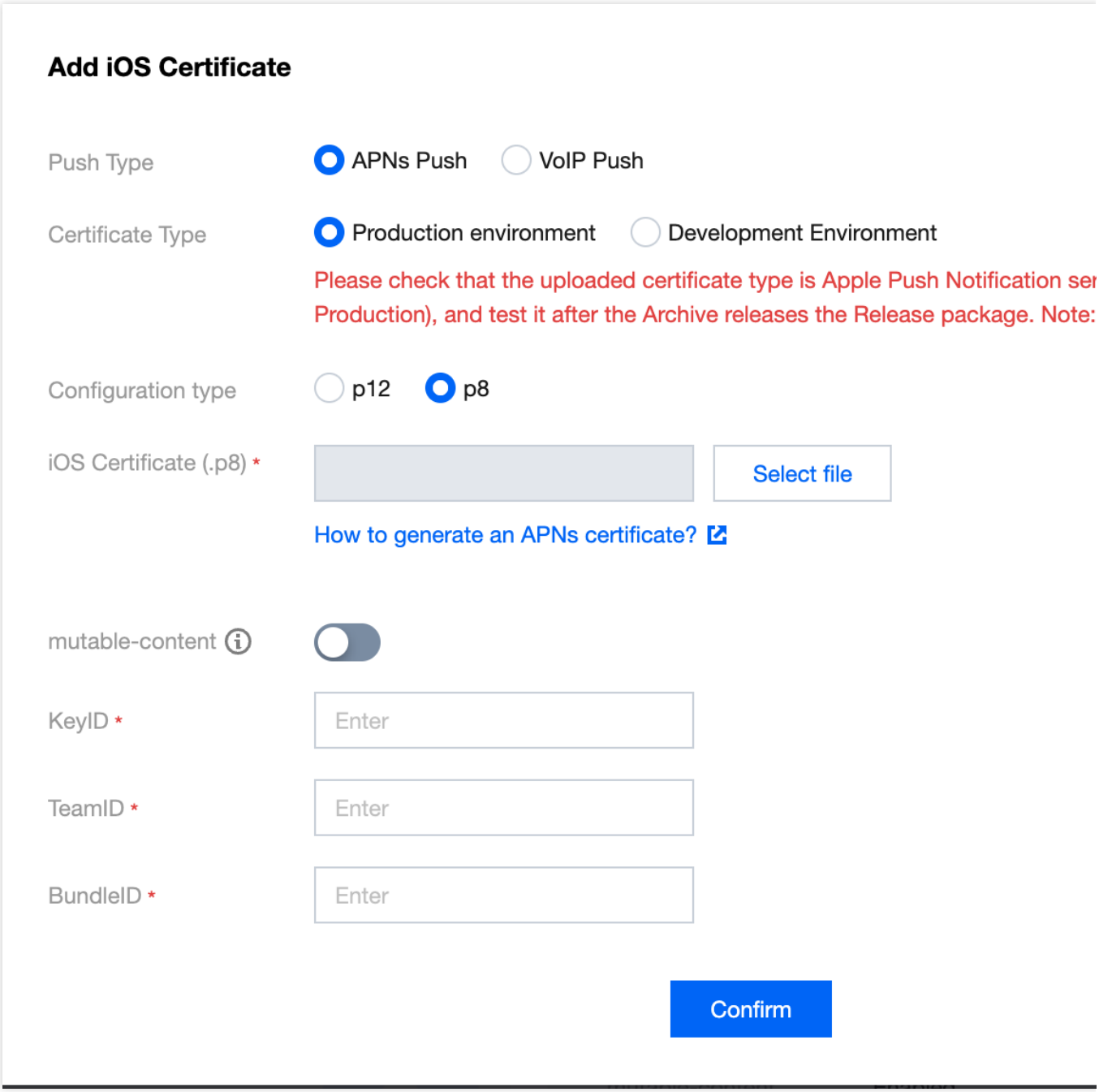
Please safeguard the downloaded p8 file, as you will not be able to download it again. You can use this p8 certificate to configure your iOS applications to receive push notifications.

Step 2: Upload the p8 certificate to the IM console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page



3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the .p8 certificate



Note:

Key ID: This is the unique identifier for your APNs Auth Key. When you create a new APNs Auth Key in the Apple Developer Center, a Key ID will be generated for you. You can find it in the "Certificates, Identifiers & Profiles" section under "Keys".

Team ID: This is the unique identifier for your developer account. You can find it on the account details page of the Apple Developer Center. Click "Membership" in the upper right corner, and you can find your Team ID in the "Membership Details" section.

Bundle ID: This is the unique identifier for your application, also known as the app ID. You can find it in the "Certificates, Identifiers & Profiles" section of the Apple Developer Center. Select "Identifiers", then find the corresponding Bundle ID in your list of applications.

Flutter

Last updated：2024-06-13 11:00:50

The current messaging push plugin, when utilized within Flutter, exclusively supports dispatching notifications to Android devices (including channels from various manufacturers) and iOS devices.

iOS

Android

Before integrating the TIMPush component, you need to apply for an APNs Push Certificate from Apple and then upload the Push Certificate to the IM console. After that, you can proceed with the quick access steps.

There are currently two mainstream types of certificates for Apple Manufacturer Configuration: p12 certificates and p8 certificates. Each type of certificate has its advantages and drawbacks, and you can choose one according to your needs.

Certificate Type:

p12 Certificate: A p12 certificate is a binary file containing both a public key and a private key, used for certificate-based authentication. It bundles the public key certificate and the private key into one file, with extensions .p12 or .pfx.

p8 Certificate: A p8 certificate is an Auth Key, used for token-based authentication. It is a text file containing a private key, with an extension of .p8.

Validity and Management:

p12 Certificate: A p12 certificate typically has a one-year validity period, after which it needs to be regenerated and deployed. Each application requires a separate P12 certificate to handle push notifications.

p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a P8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications.

Security:

p12 Certificate: A p12 certificate uses certificate-based authentication and requires the private key to be stored on the server. This could increase security risks, as the private key could be accessed by unauthorized users.

p8 Certificate: A p8 certificate uses token-based authentication, which means your server periodically generates a JSON Web Token (JWT) to establish a connection with APNs. This method is more secure, as it doesn't require storing a private key on the server.

Dynamic Island:

p12 Certificate: Not supported.

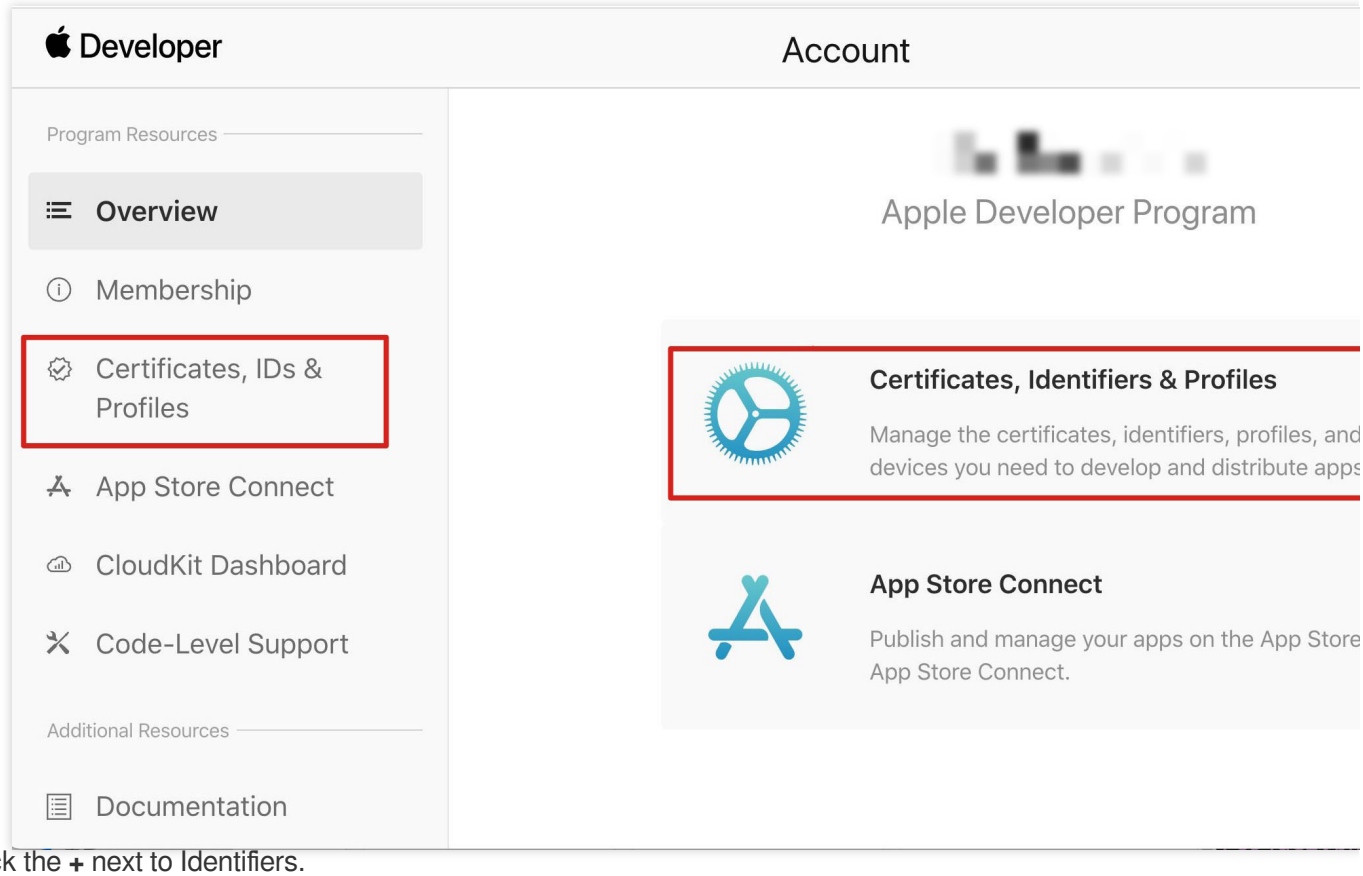
p8 Certificate: Supports Dynamic Island push notifications.

1. Using a p12 certificate (traditional push certificate)

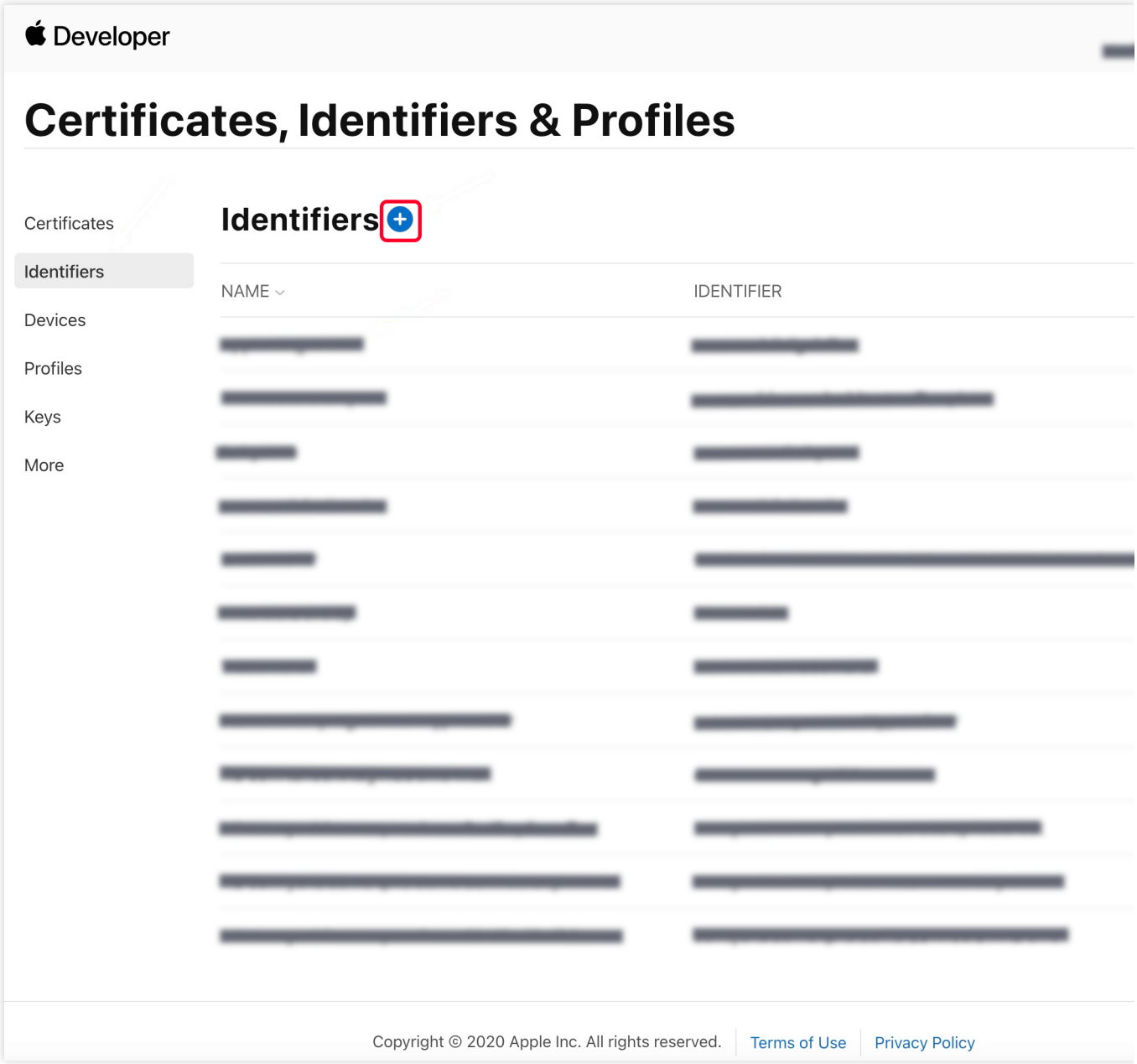
Step 1: Apply for an APNs certificate

Enable remote push for the app

1. log in to Apple Developer Center website, click **Certificates, Identifiers & Profiles** or the sidebar's **Certificates, IDs & Profiles**, enter the **Certificates, IDS & Profiles** page.



2. click the + next to Identifiers.

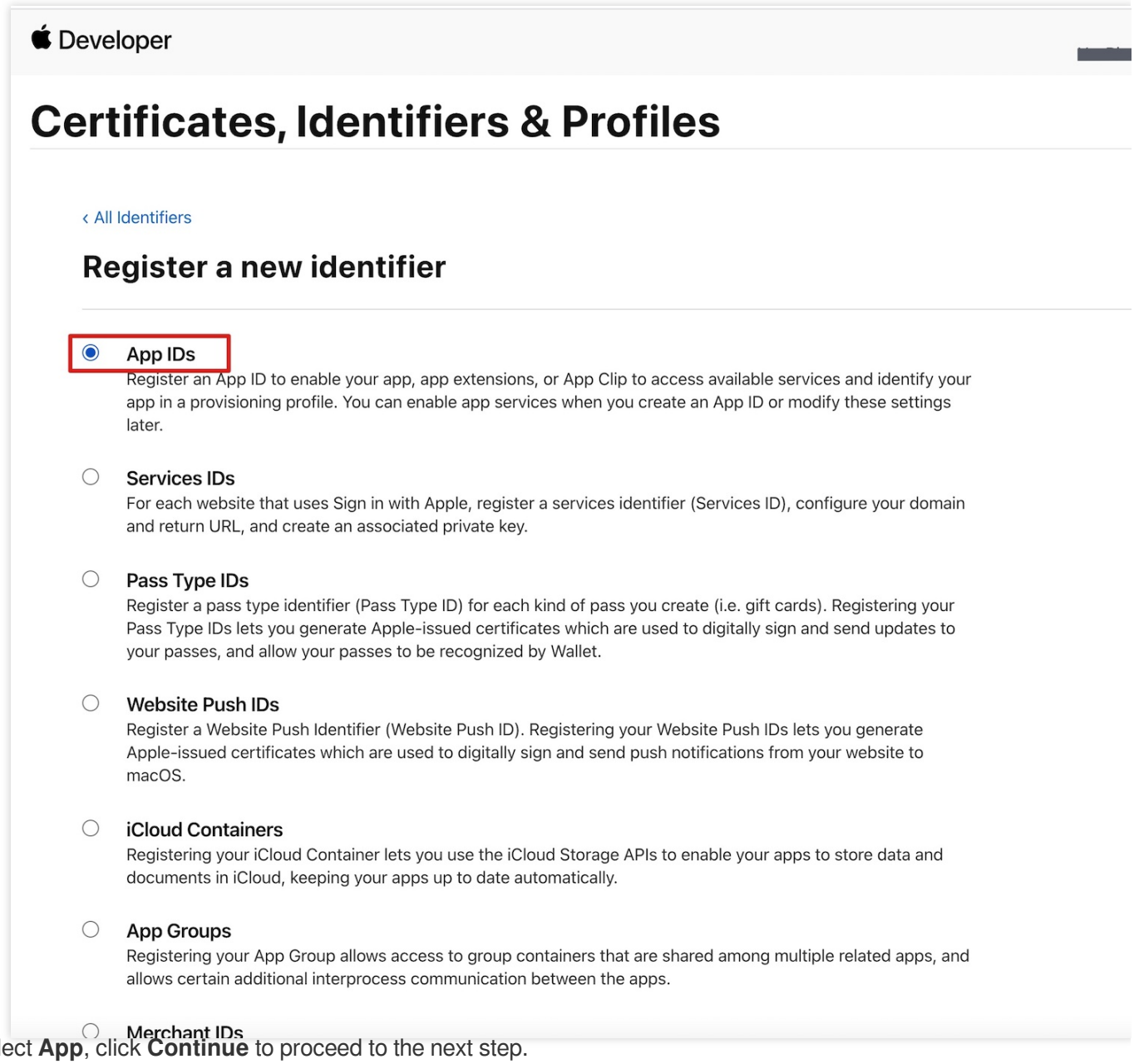


3. You can follow the steps below to create a new AppID or add a `Push Notification Service` to your existing AppID.

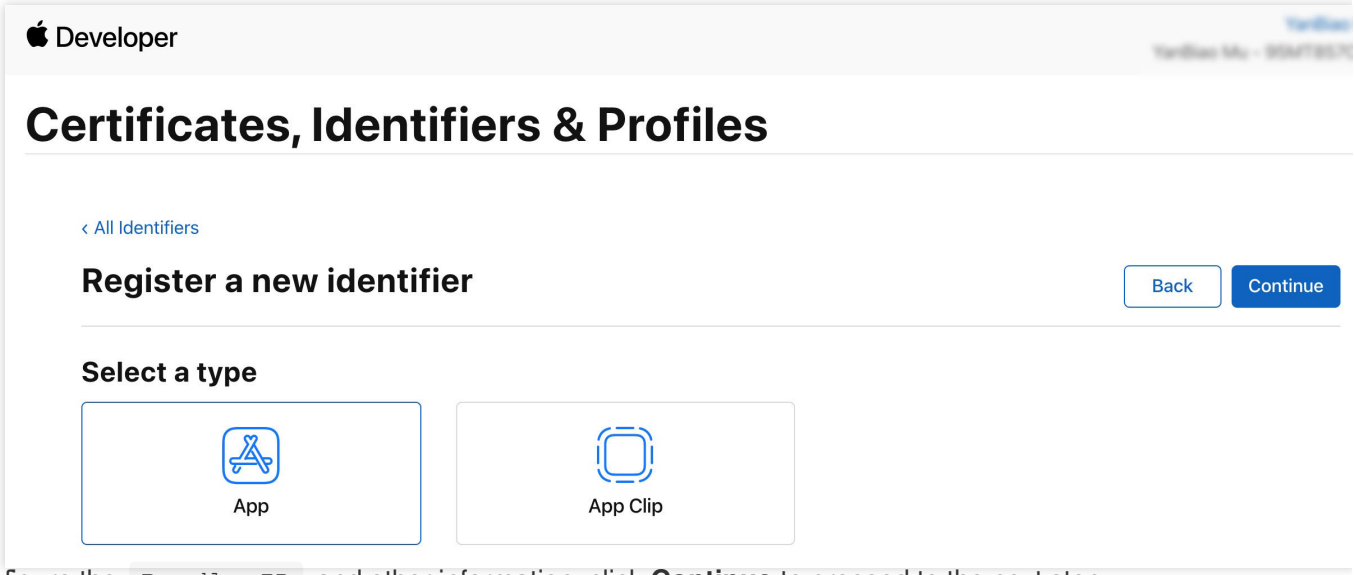
Note:

Your App's `Bundle ID` cannot use the wildcard `*`, otherwise, the remote push service cannot be used.

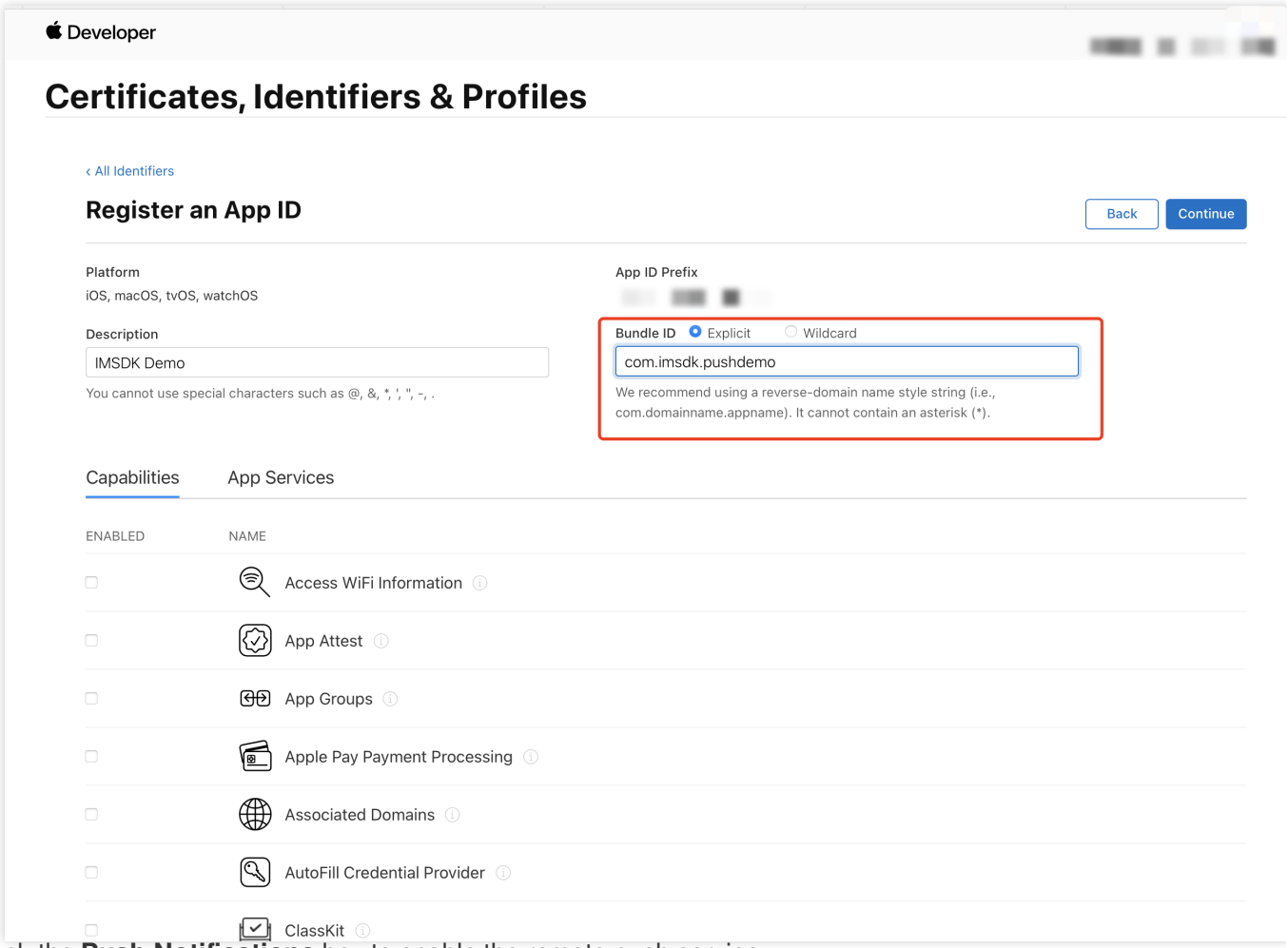
4. Check the **App IDs** box, click **Continue** to proceed to the next step.



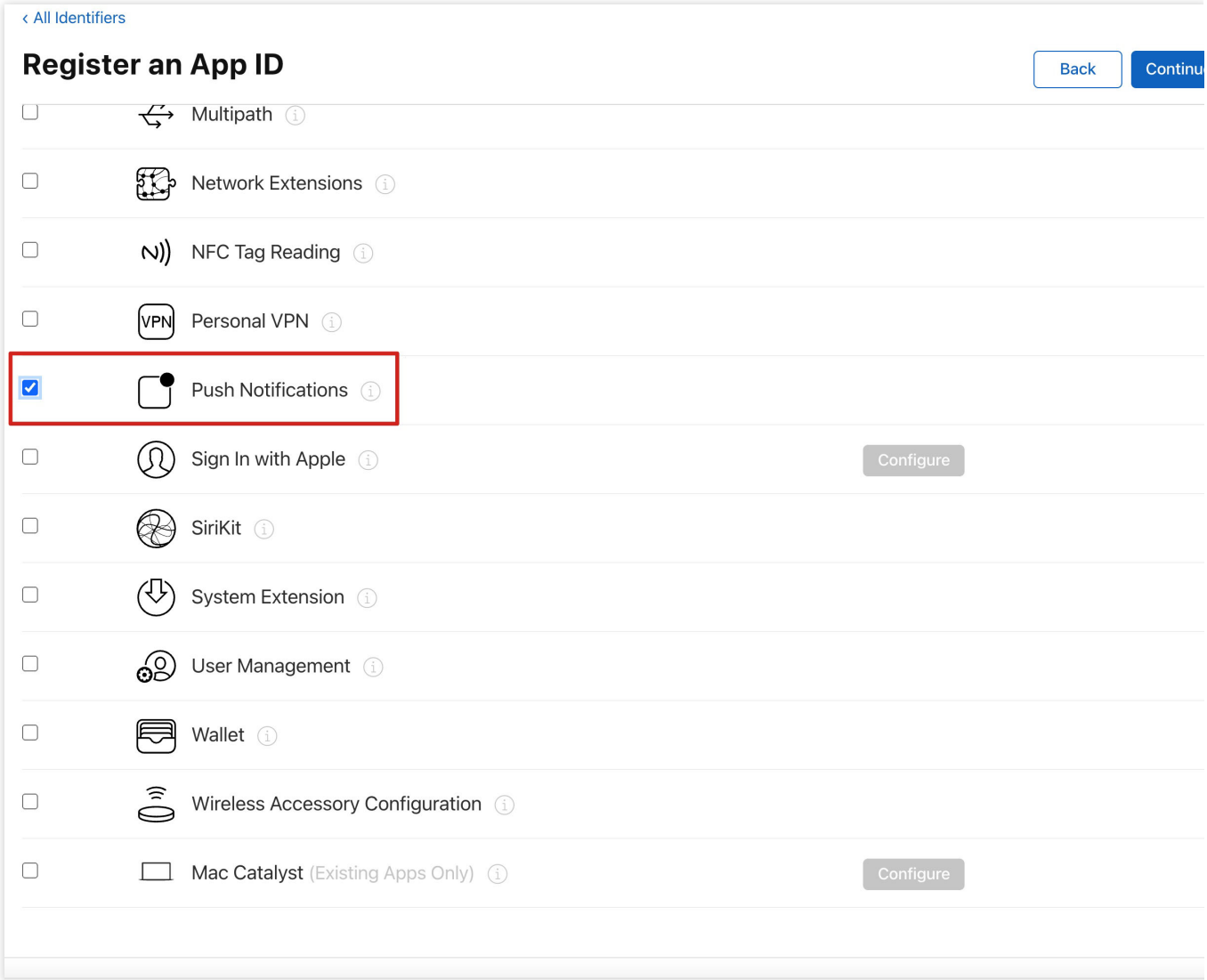
5. Select **App**, click **Continue** to proceed to the next step.



6. Configure the `Bundle ID` and other information, click **Continue** to proceed to the next step.

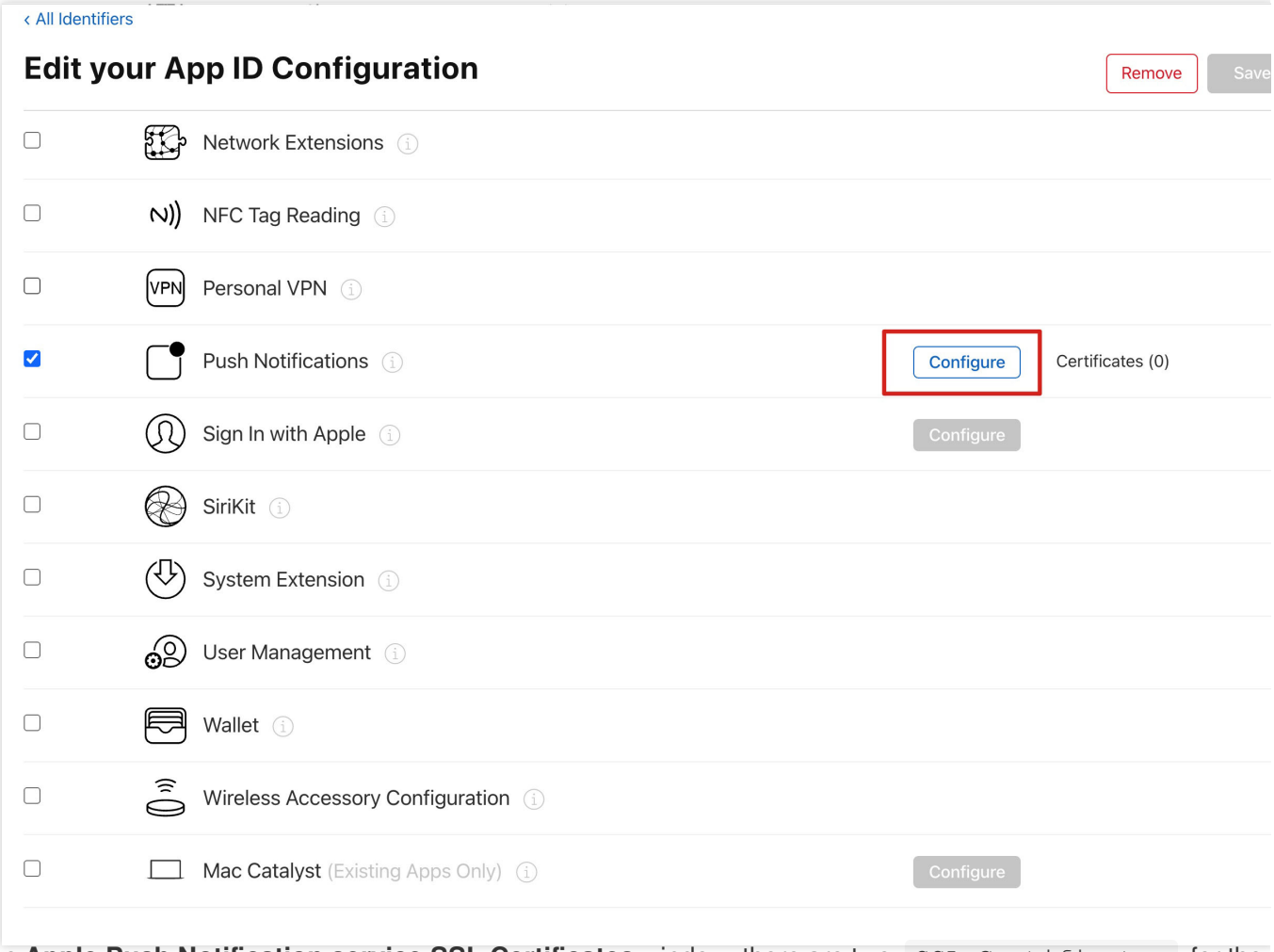


7. Check the **Push Notifications** box to enable the remote push service.

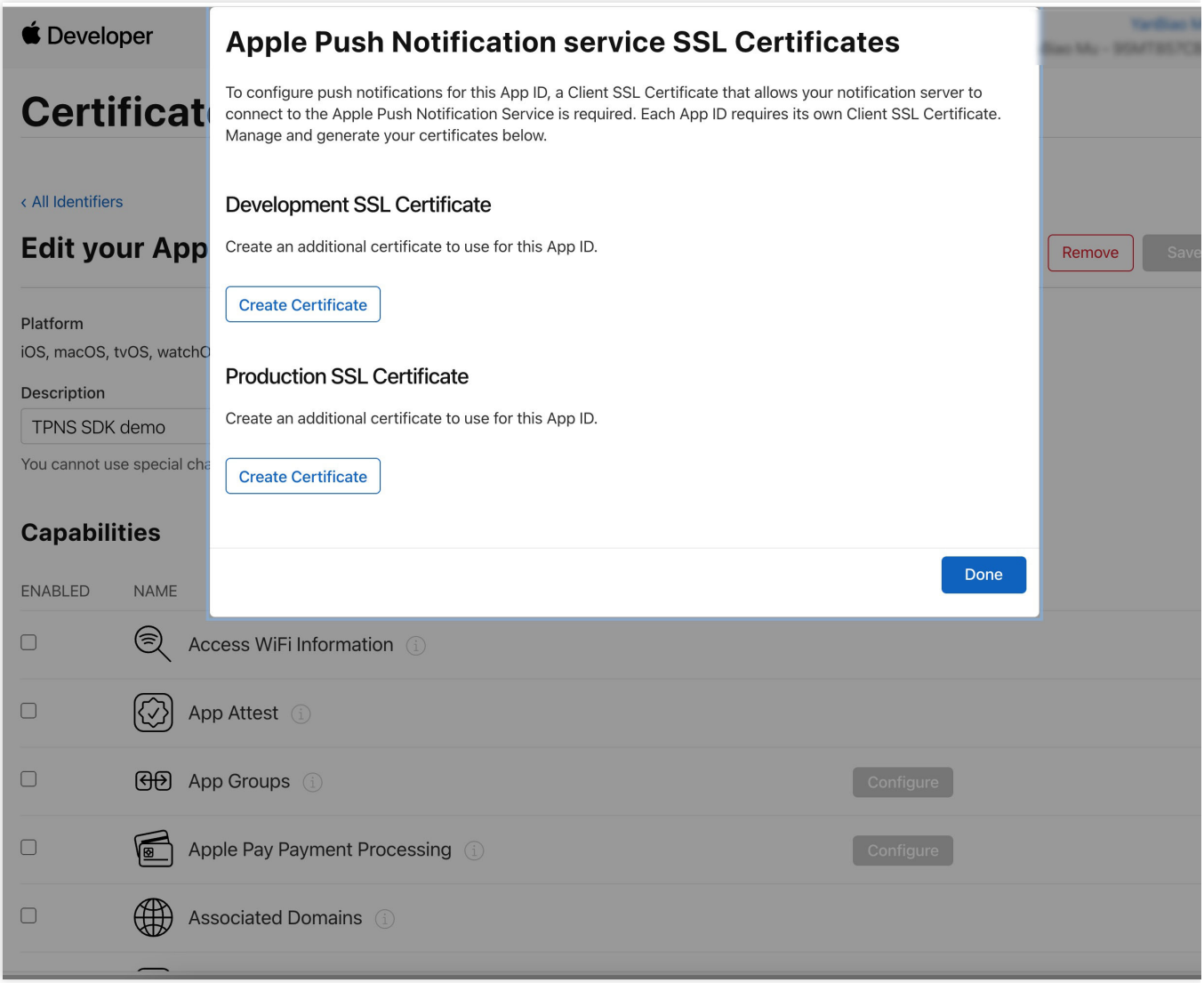


Certificate Generation

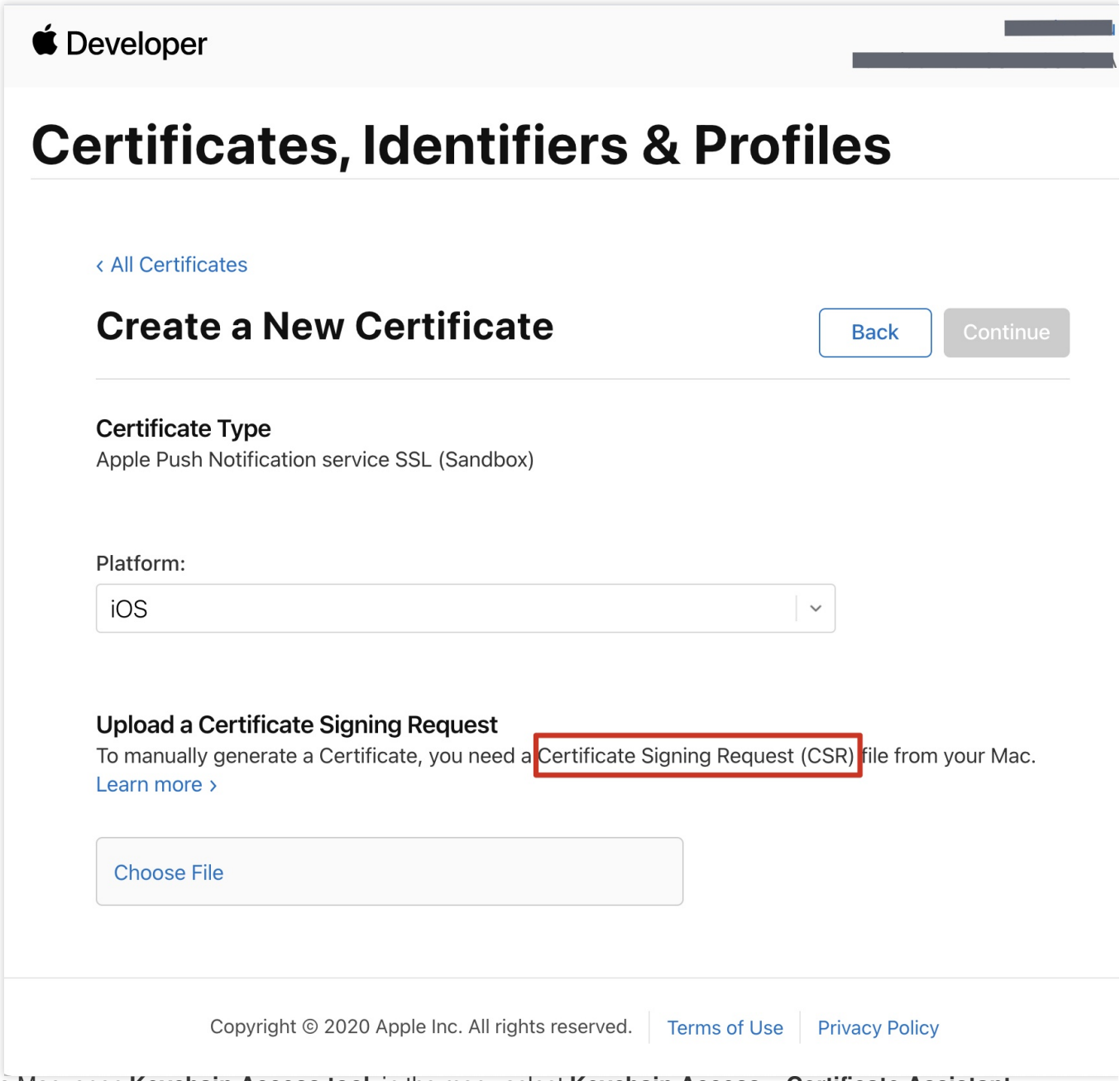
1. Select your AppID and choose **Configure**.



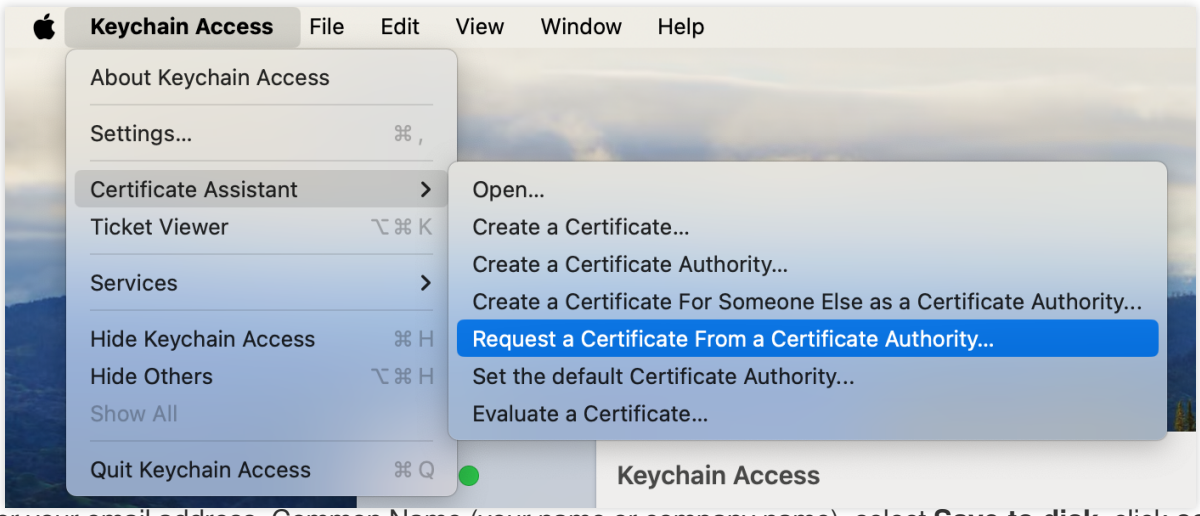
2. In the **Apple Push Notification service SSL Certificates** window, there are two `SSL Certificates` for the development environment (Development) and the production environment (Production), as shown below:



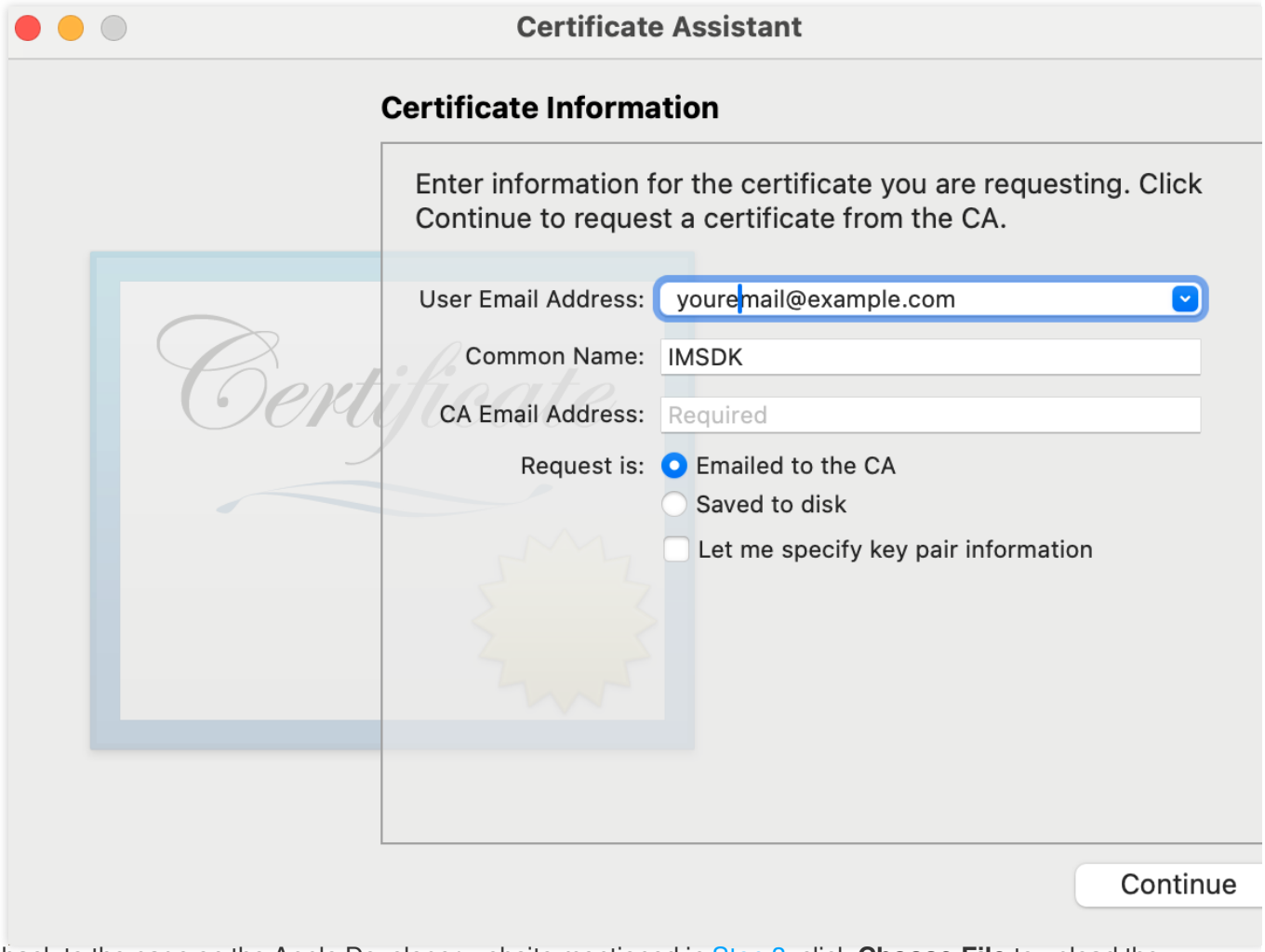
3. We first select the **Create Certificate** for the Development environment, the system will prompt us that we need a Certificate Signing Request (CSR).



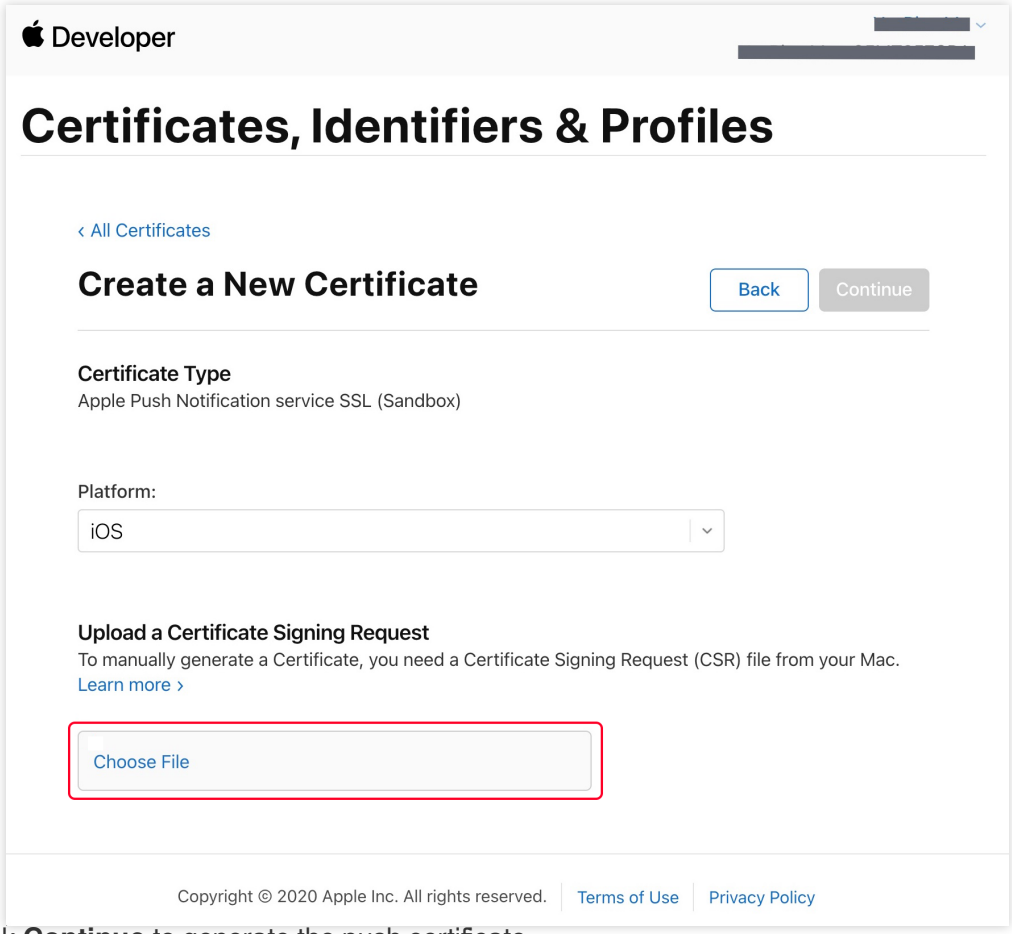
4. On a Mac, open **Keychain Access tool**, in the menu select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority** (`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`).



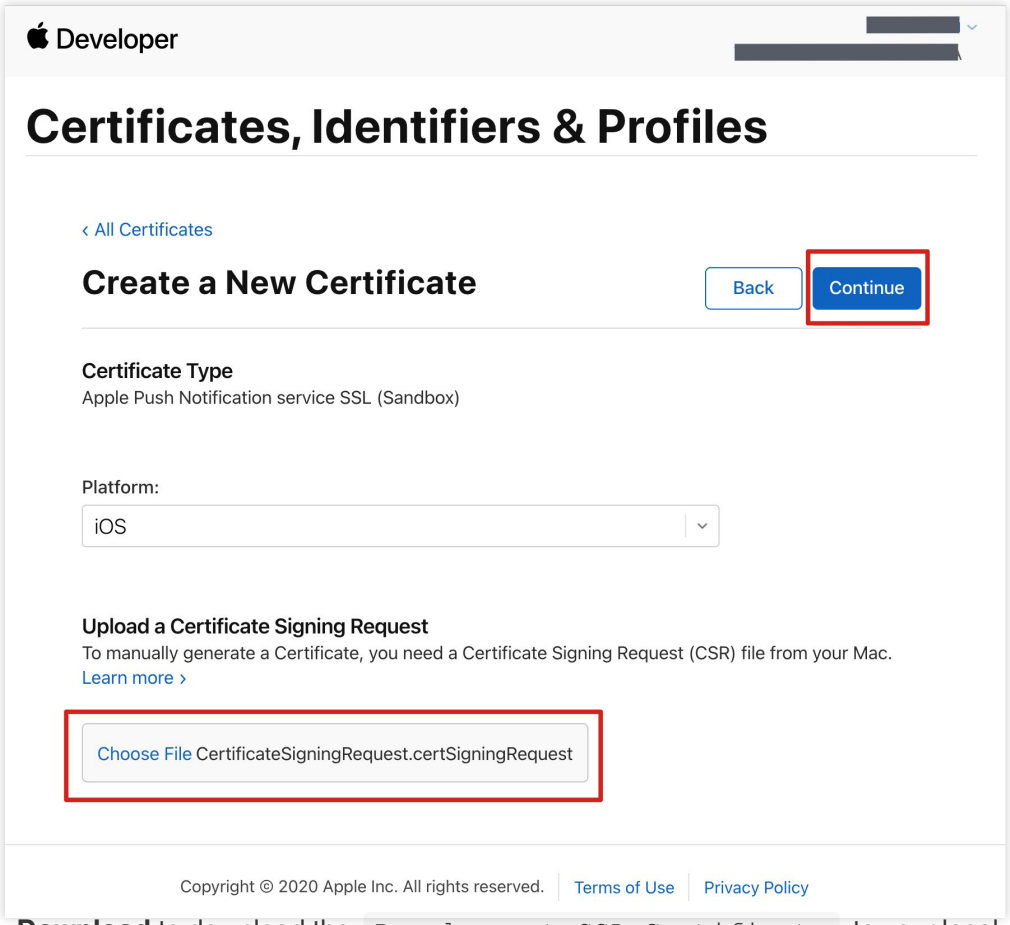
5. Enter your email address, Common Name (your name or company name), select **Save to disk**, click **continue**, the system will generate a `*.certSigningRequest` file.



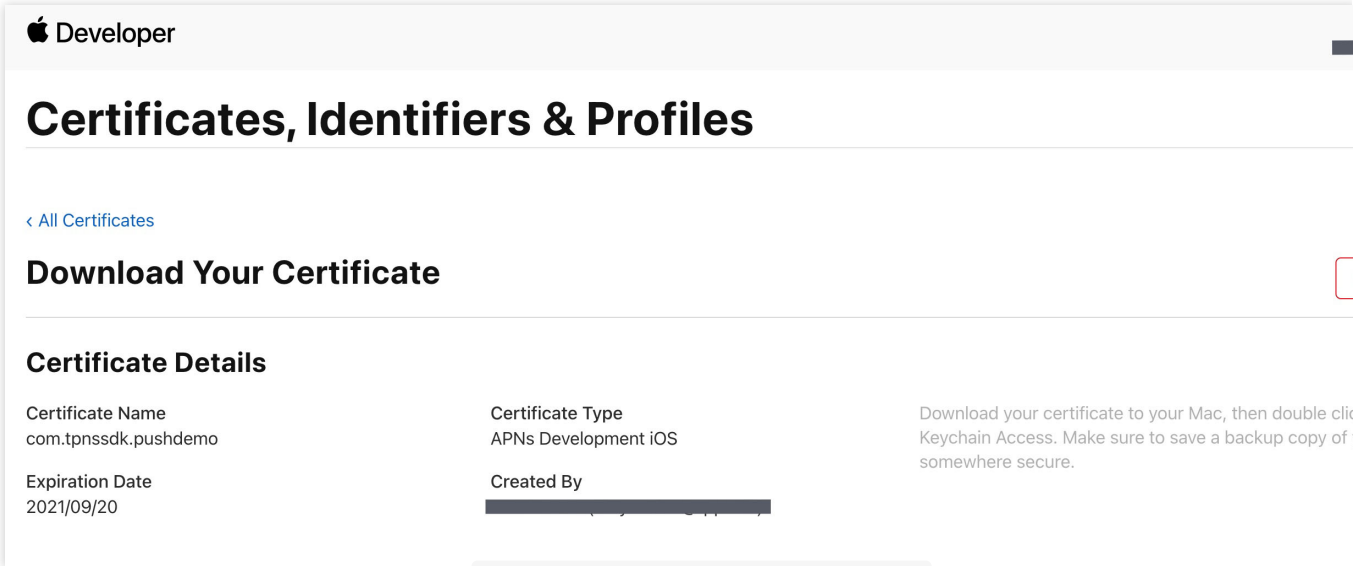
6. Go back to the page on the Apple Developer website mentioned in Step 3, click **Choose File** to upload the generated `*.certSigningRequest` file.



7. Click **Continue** to generate the push certificate.



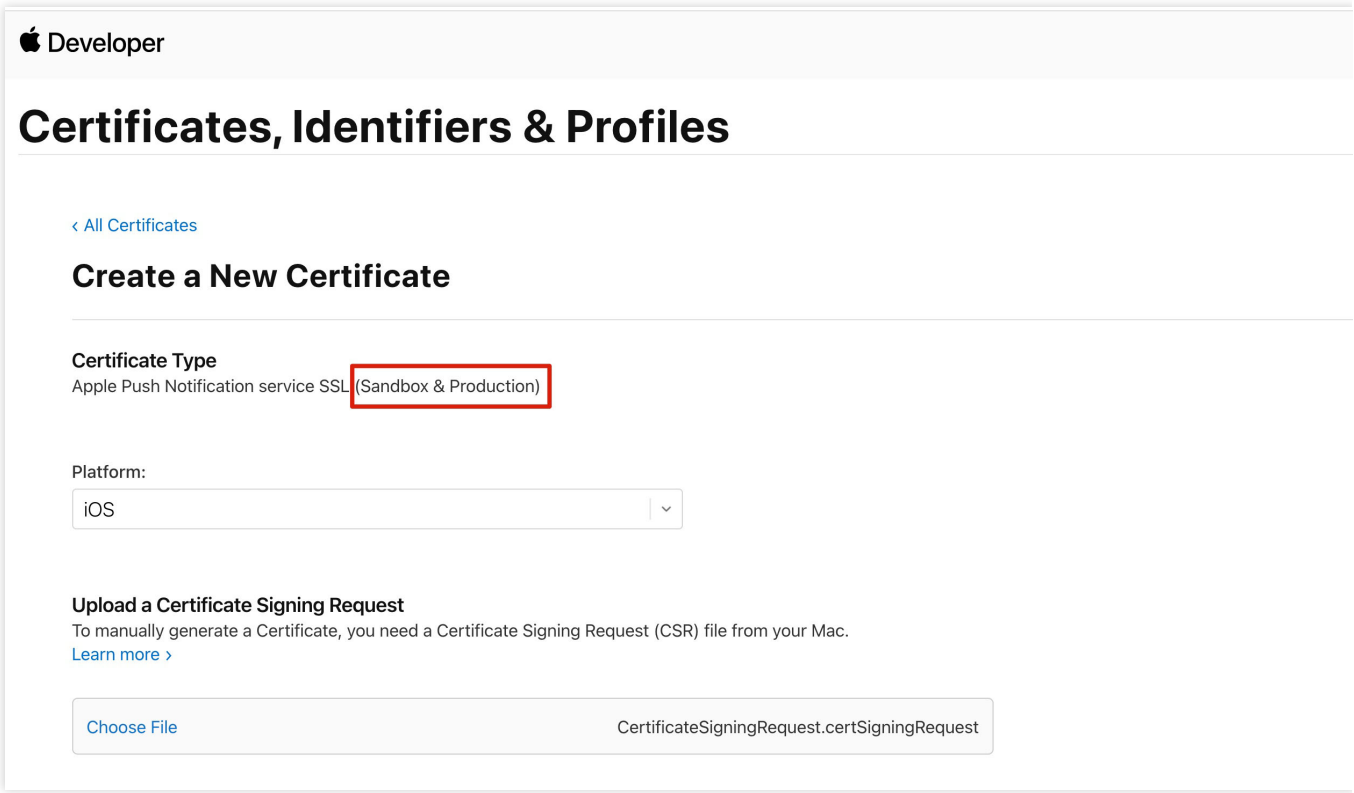
8. click **Download** to download the `Development SSL Certificate` to your local environment.

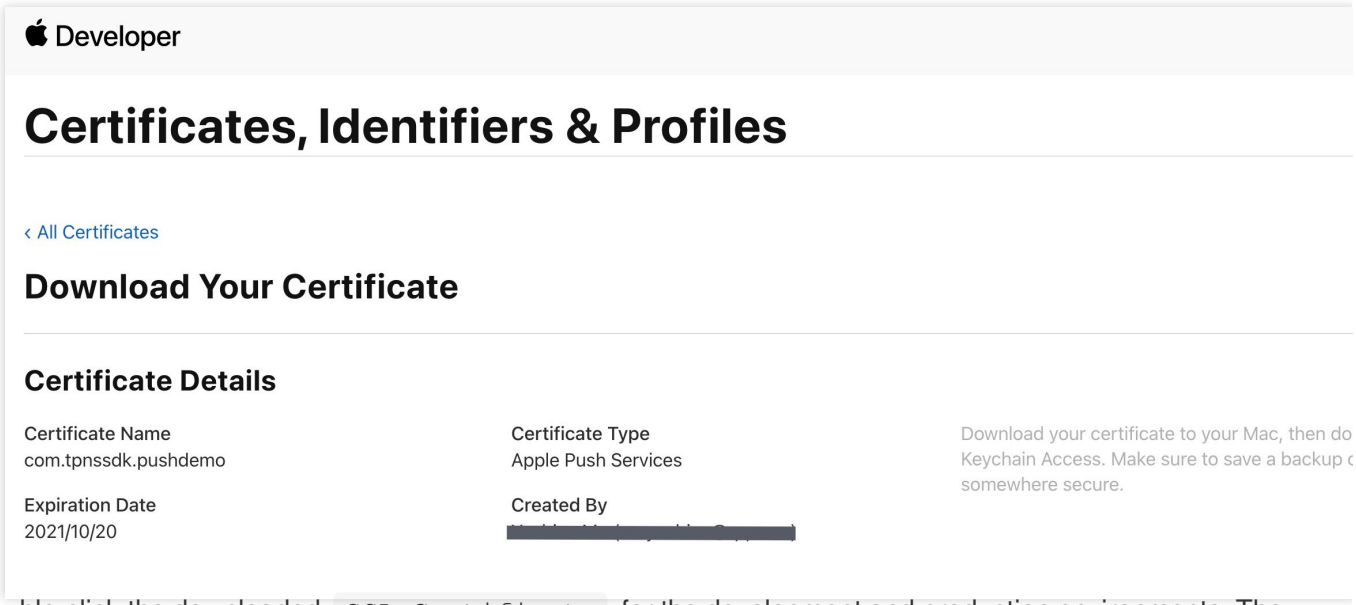


9. Repeat steps 1 - 8 above to download the `Production SSL Certificate` for the production environment to your local machine.

Note:

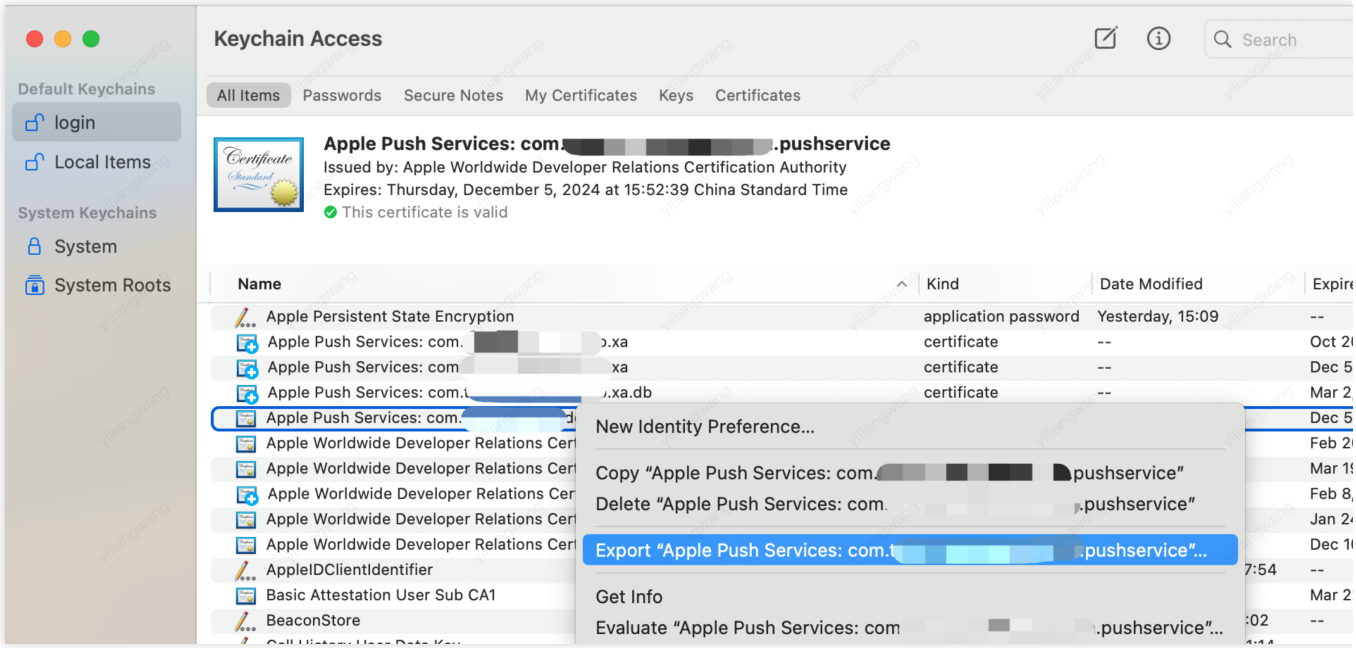
The certificate for the production environment is actually a combined certificate of Development (Sandbox) + Production, and it can be used as a certificate for both the development and production environments.





10. Double-click the downloaded `SSL Certificate` for the development and production environments. The system will import it into the keychain.

11. Open the Keychain App, go to **log in to** > **My Certificates**, right-click to export the newly created `Apple Development IOS Push Services` and `Apple Push Services` for the development and production environments as `p12` files respectively.

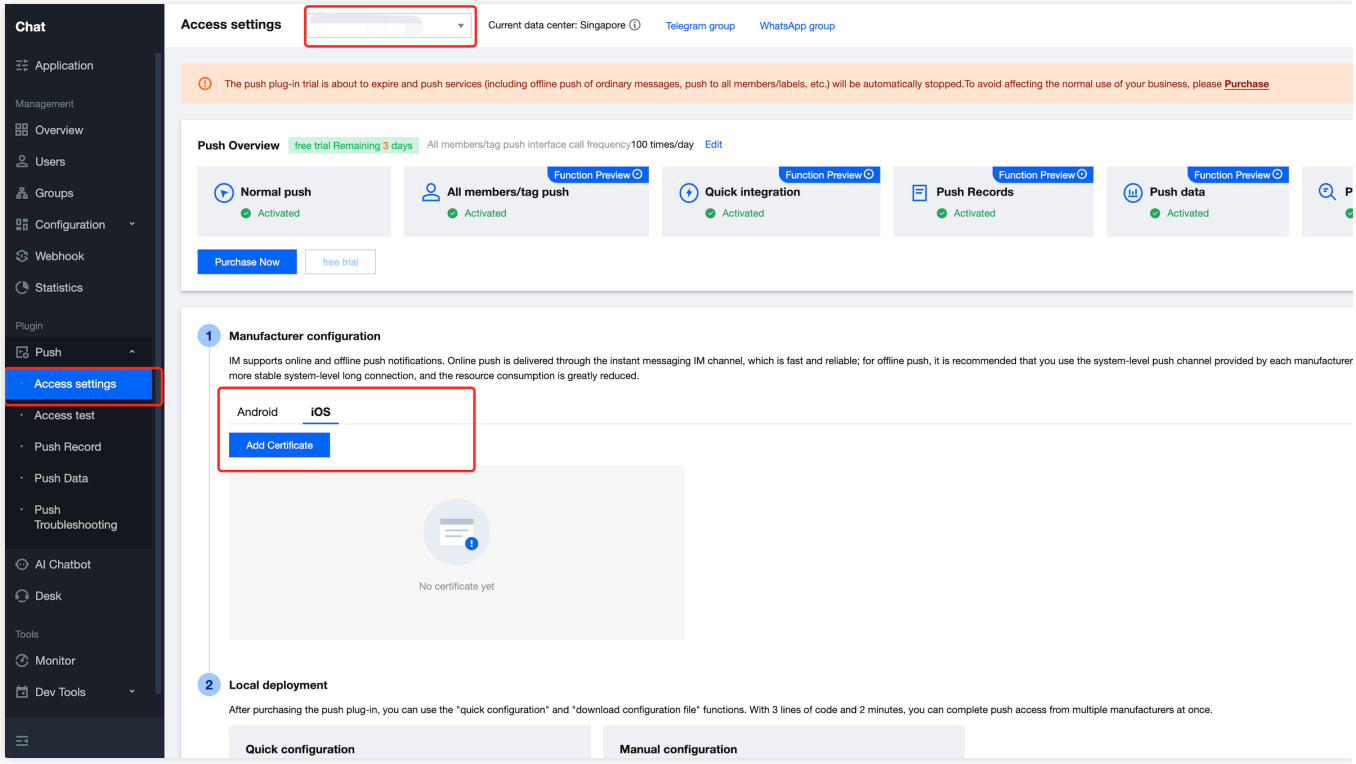


Note

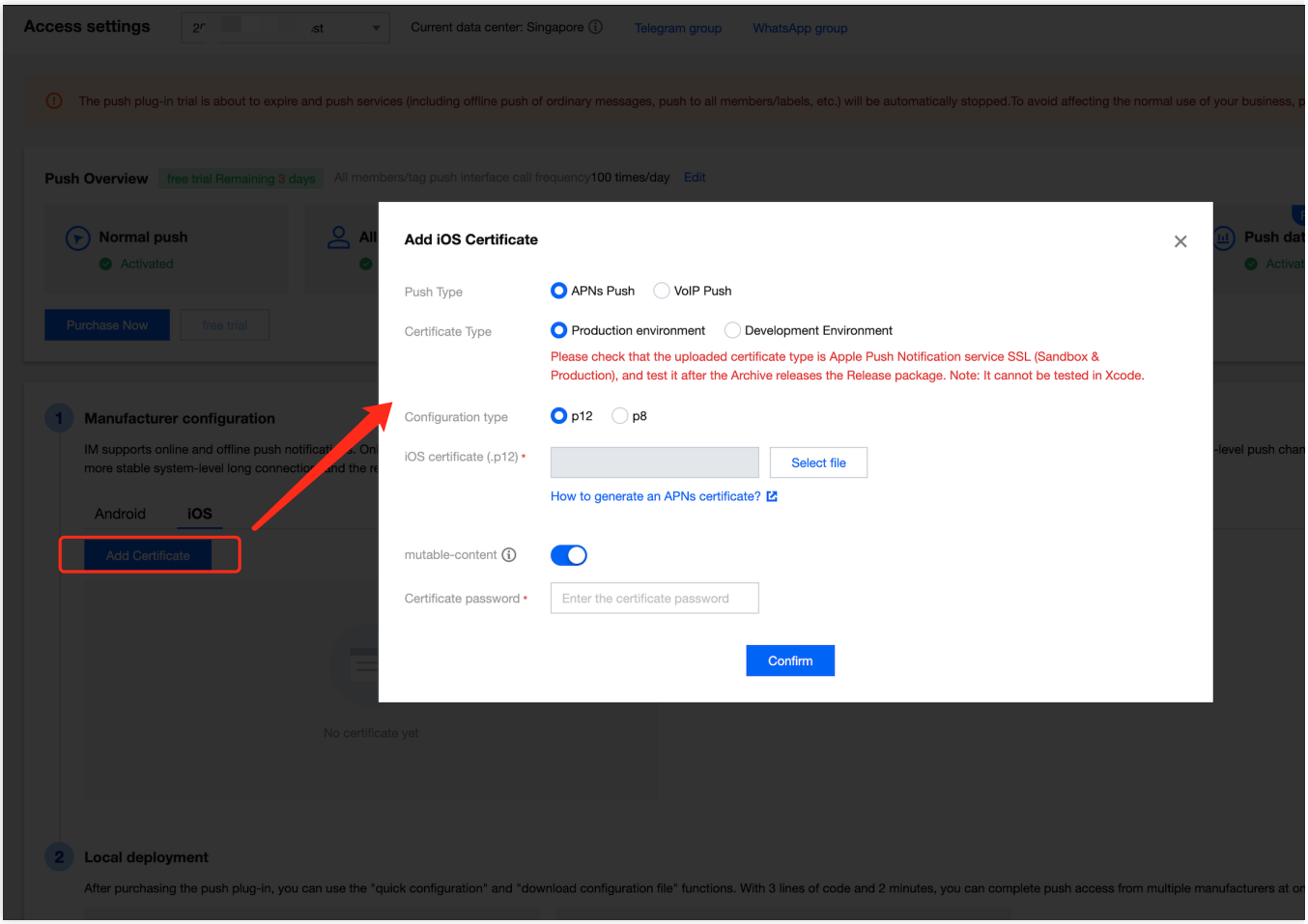
Set a password when saving the `.p12` file.

Step 2: Upload the certificate to the console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page



3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the certificate type, upload the iOS Certificate (.p12), set the certificate password, and click **Confirm**.



Note:

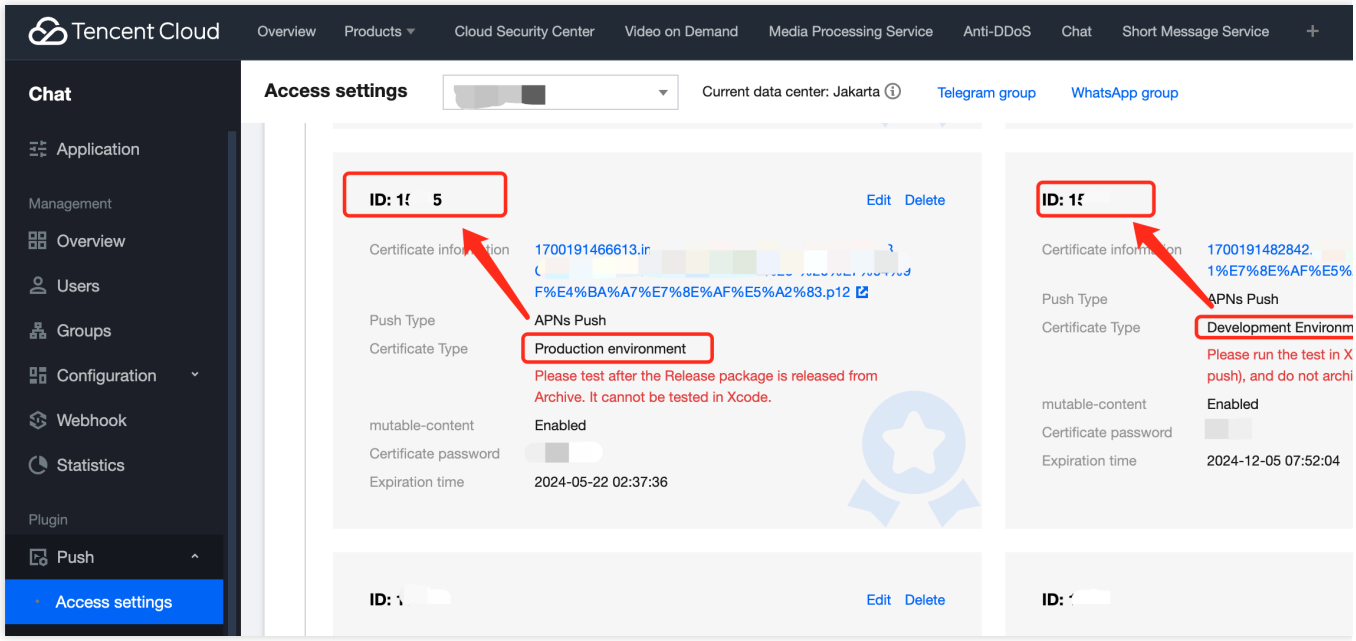
We recommend naming the uploaded certificate in English (special characters such as brackets are not allowed).

You need to set a password for the uploaded certificate. Without a password, push notifications cannot be received.

For an app published on App Store, the environment of the certificate must be the production environment. Otherwise, push notifications cannot be received.

The uploaded .p12 certificate must be your own authentic and valid certificate.

5. After the pending certificate information is generated, record the certificate's ID.

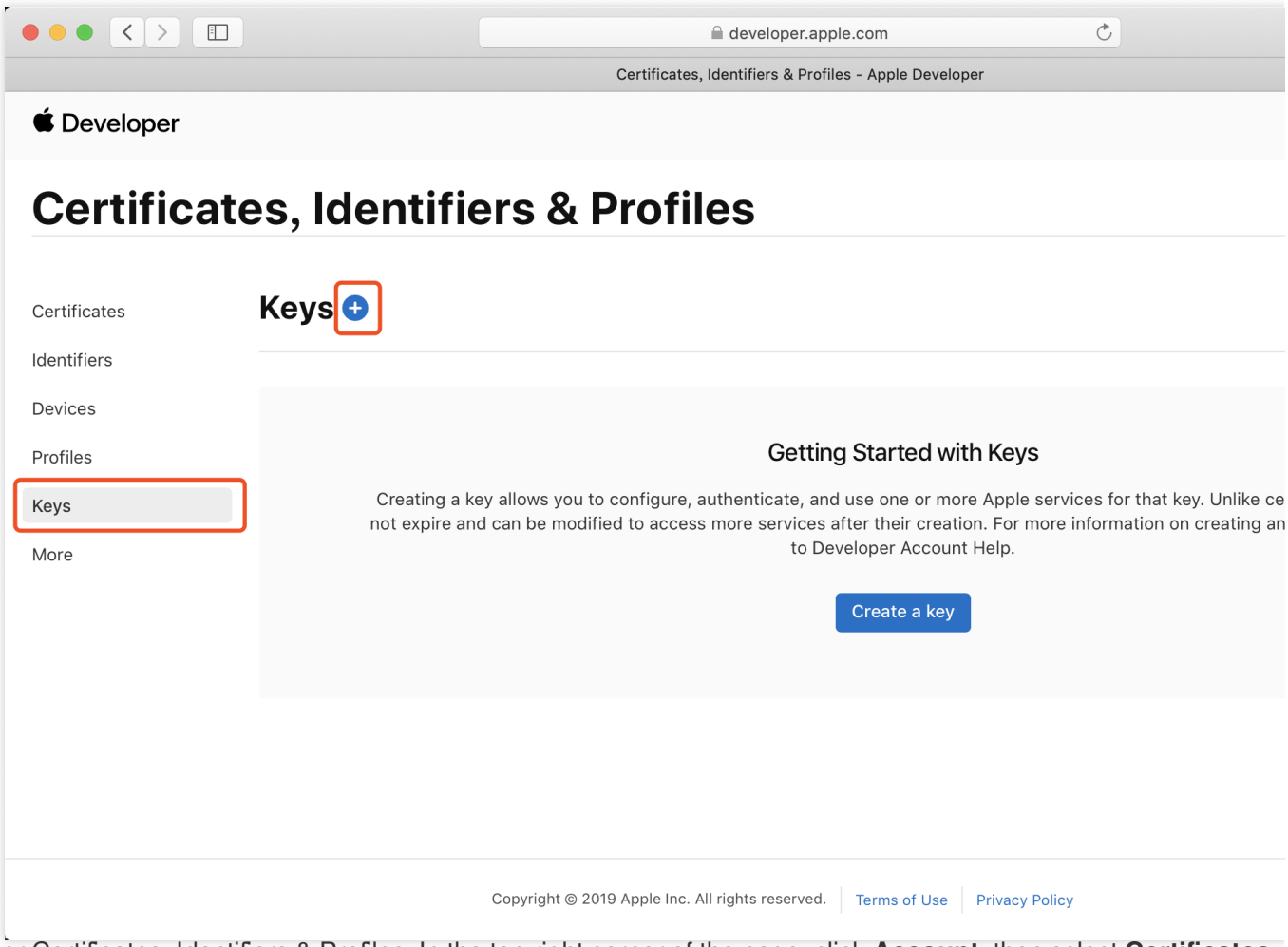


Second, using a p8 certificate (supports Dynamic Island push notifications)

p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a p8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications. In addition, p8 certificates support Dynamic Island push notifications.

Step 1: Apply for an APNs certificate

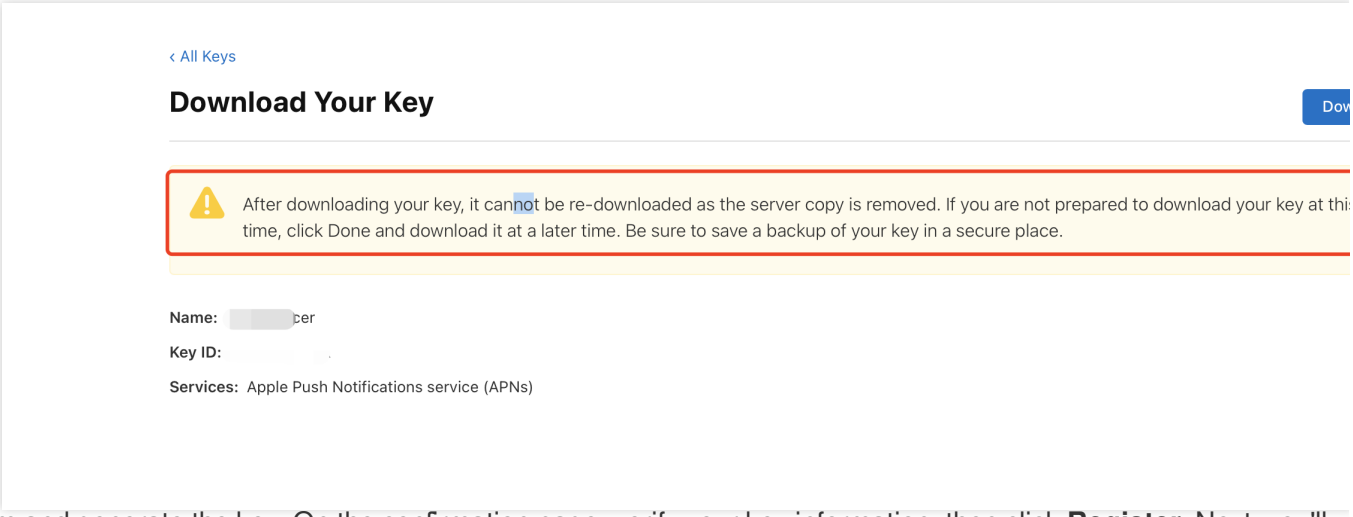
To create a p8 certificate file, first log in to Apple Developer Center.



1. Enter Certificates, Identifiers & Profiles: In the top right corner of the page, click **Account**, then select **Certificates, Identifiers & Profiles** from the dropdown menu.

2. To create a new App ID: in the left-hand menu, click **Identifiers**, then click the **+** on the right to create a new App ID. Fill in the relevant information and click **Continue**.

3. To create a new key: in the left-hand menu, click **Keys**, then click the **+** on the right to create a new key. Enter the name of the key, then check **Apple Push Notifications service (APNs)** and click **Continue**.



Confirm and generate the key: On the confirmation page, verify your key information, then click **Register**. Next, you'll see a page prompting you to download the key. Click **Download** and save the generated .p8 file to your computer.

Note:

The p8 certificate can only be downloaded once; please save it properly.

Please safeguard the downloaded p8 file, as you will not be able to download it again. You can use this p8 certificate to configure your iOS applications to receive push notifications.

Step 2: Upload the p8 certificate to the IM console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page
3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the .p8 certificate

Operation step

Step 1. Register your app with vendor push platforms

To utilize the offline push feature, you need to register your app on each vendor's push platform to obtain parameters such as AppID and AppKey. Currently, the mobile manufacturers supported in China include: Mi, Huawei, Honor, OPPO, Vivo, Meizu, and internationally Google FCM is supported.

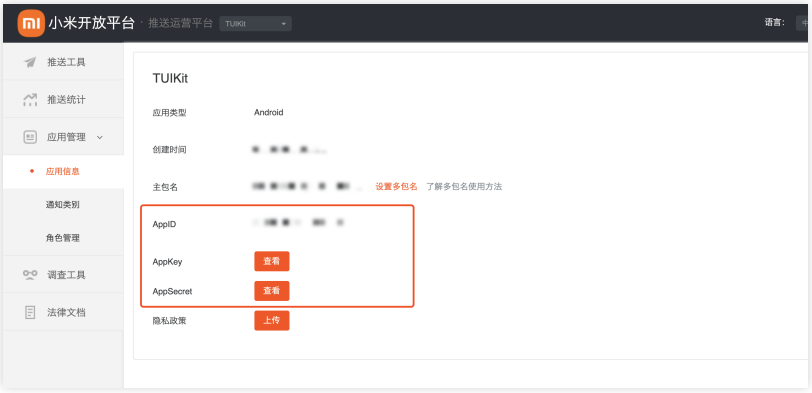
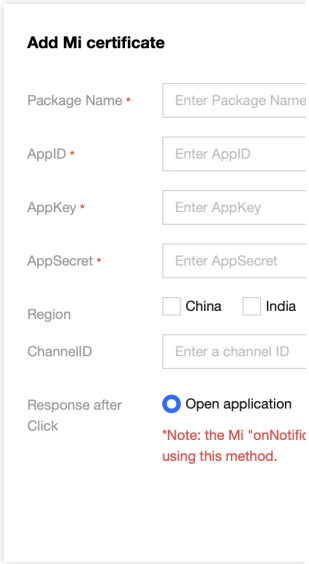
Step 2. Create resources in the IM console

Log in to Tencent Cloud Chat Console, then in the **Push Management** > **Access Settings** feature section, add each vendor's push certificate, and configure the AppID, AppKey, AppSecret, and other parameters obtained in Step 1 to the added push certificate.

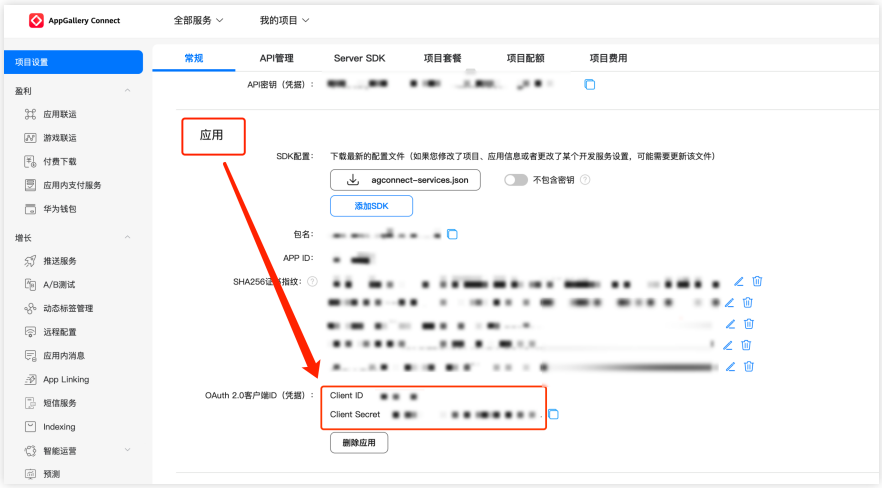
Explanation of the **Subsequent Actions** option:

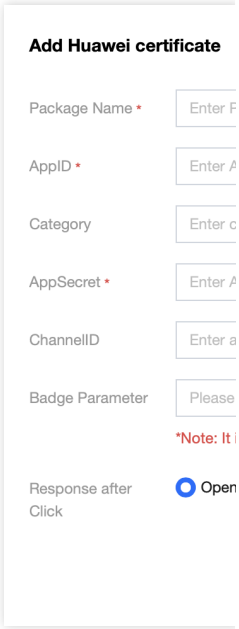
Open Application: Clicking the notification bar launches the app, by default starting the app's Launcher interface;

Open Web Page: Clicking the notification bar will redirect to the configured web link;
Open the specified interface within the app: clicking the notification bar will redirect the interface based on the configured self Definition, see Custom Redirect on Click.
Mi
Huawei
OPPO
vivo
Meizu
HONOR
Google FCM

Vendor Push Platform	Configuring in the IM console
	

Vendor Push Platform	Configuring in the IM cons

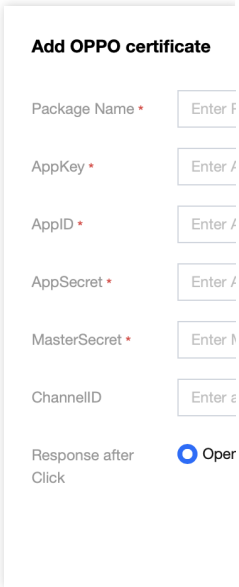




Note:
Client ID corresponds to A

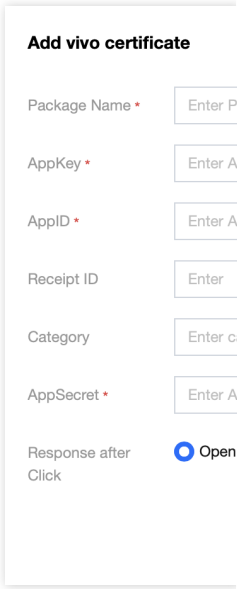
Vendor Push Platform

Configuring in the IM cons



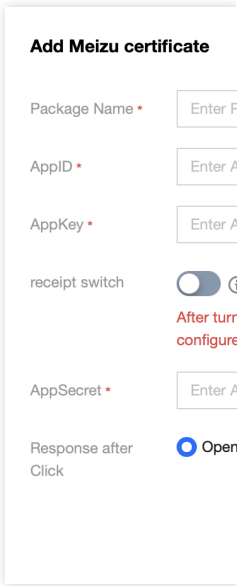
Vendor Push Platform

Configuring in the IM cons

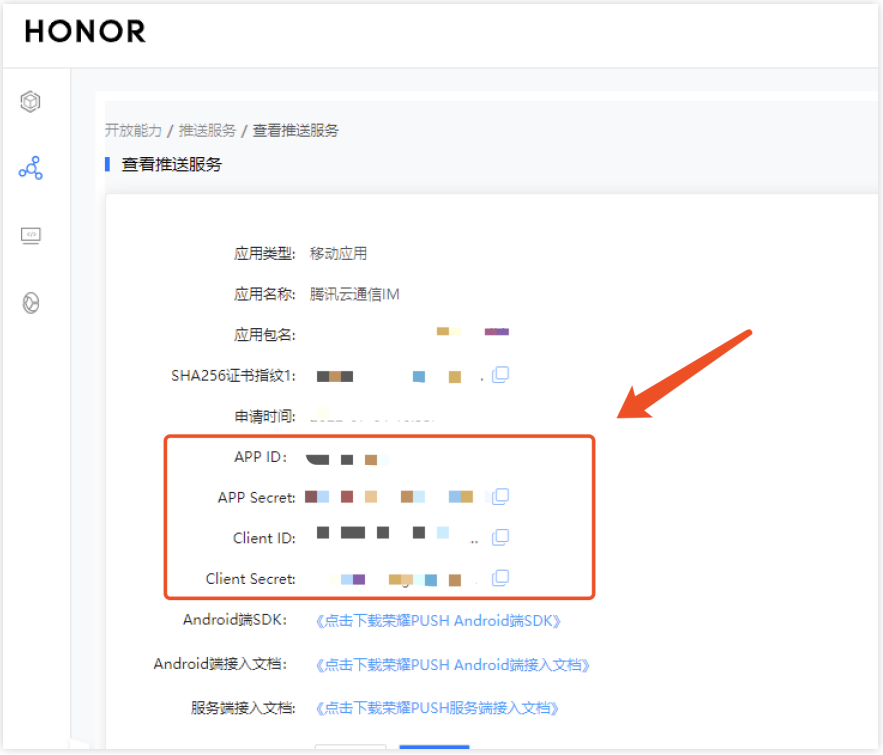
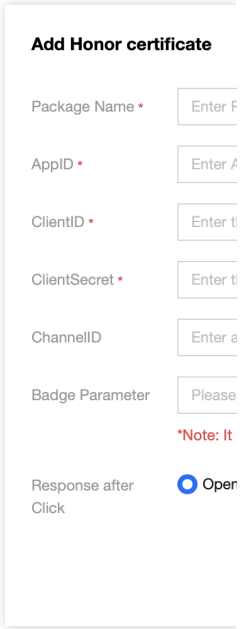


For receipt configuration, please refer to: Message reach statistics configuration - vivo

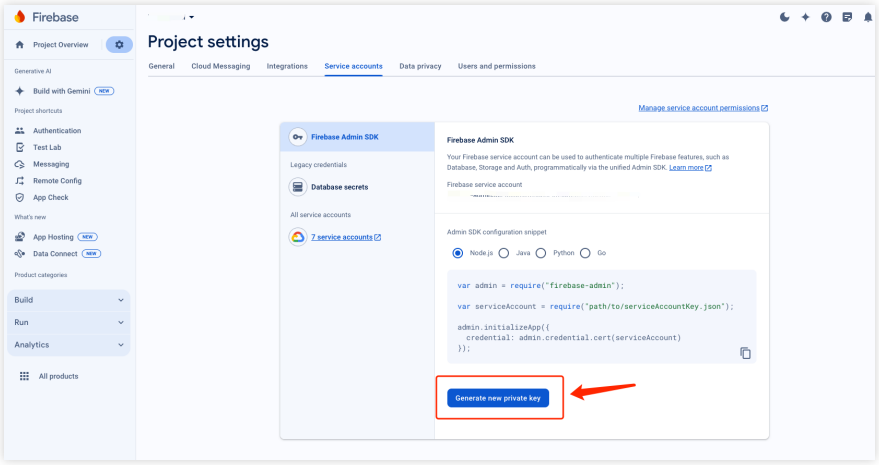
Vendor Push Platform	Configuring in the IM cons

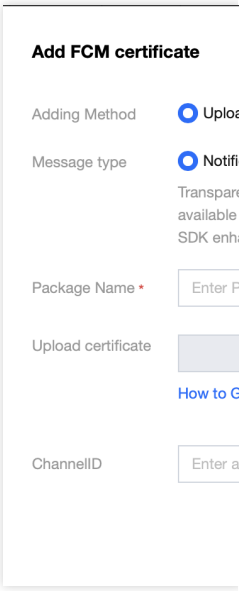


For receipt configuration, please refer to: Message reach statistics configuration - Meizu

Vendor Push Platform	Configuring in the IM cons
	

Vendor Push Platform	Configuring in the IM cons





Note:

Regarding **Click for Subsequent Actions** supports the Report Statistics feature:

1. If you choose to open an app or a web page, purchasing the plugin will by default support reporting statistics.

2. If you choose to open a specified interface within the application:

For new certificate status, please directly use the auto-fill default value to support click statistics reporting.

If there was a certificate configured previously, continue using the old certificate and modify it to the default value to support reporting statistics, or regenerate a new certificate.

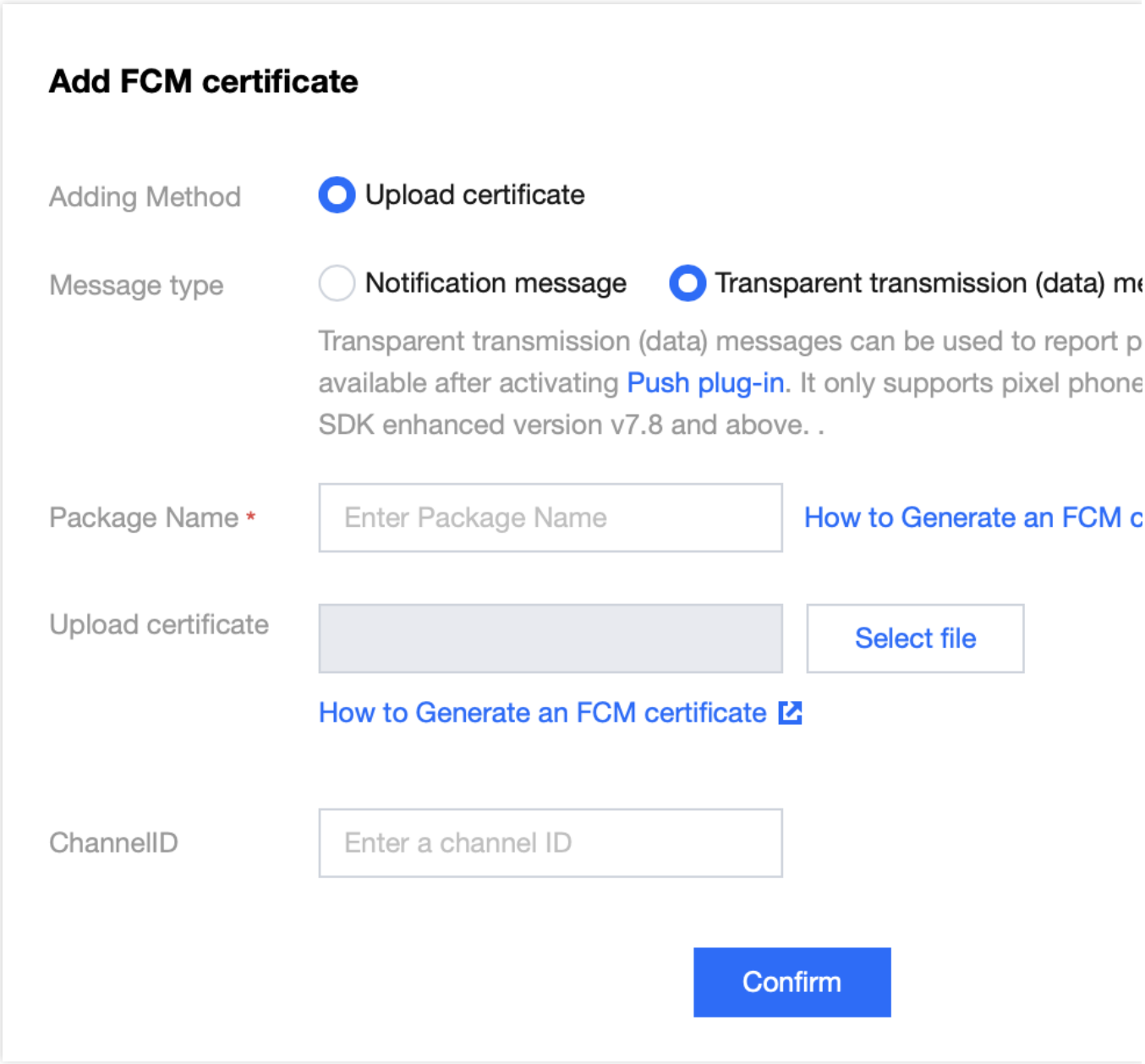
About FCM Data Messaging

FCM provides two push methods: Notification Message and Data Messaging.

Notification Message, with a simple style not differentiated by device, can support offline push once successfully integrated;

Data Messaging, offering rich customization for specific devices, supports reach and click reporting, and requires testing on the device before going live after integration.

The console defaults to Notification Message, and switching between modes can be done in the IM Console:



Note:

FCM Data Messaging capability is only supported on Pixel phones with TIMPush 7.8 and above, other manufacturers' devices need to be self-tested for support;

React-Native

Last updated：2024-06-24 15:45:32

The current messaging push plugin, when utilized within React Native, exclusively supports dispatching notifications to Android devices (including channels from various manufacturers) and iOS devices.

iOS

Android

Before integrating the TIMPush component, you need to apply for an APNs Push Certificate from Apple and then upload the Push Certificate to the IM console. After that, you can proceed with the quick access steps.

There are currently two mainstream types of certificates for Apple Manufacturer Configuration: p12 certificates and p8 certificates. Each type of certificate has its advantages and drawbacks, and you can choose one according to your needs.

Certificate Type:

p12 Certificate: A p12 certificate is a binary file containing both a public key and a private key, used for certificate-based authentication. It bundles the public key certificate and the private key into one file, with extensions .p12 or .pfx.

p8 Certificate: A p8 certificate is an Auth Key, used for token-based authentication. It is a text file containing a private key, with an extension of .p8.

Validity and Management:

p12 Certificate: A p12 certificate typically has a one-year validity period, after which it needs to be regenerated and deployed. Each application requires a separate P12 certificate to handle push notifications.

p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a P8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications.

Security:

p12 Certificate: A p12 certificate uses certificate-based authentication and requires the private key to be stored on the server. This could increase security risks, as the private key could be accessed by unauthorized users.

p8 Certificate: A p8 certificate uses token-based authentication, which means your server periodically generates a JSON Web Token (JWT) to establish a connection with APNs. This method is more secure, as it doesn't require storing a private key on the server.

Dynamic Island:

p12 Certificate: Not supported.

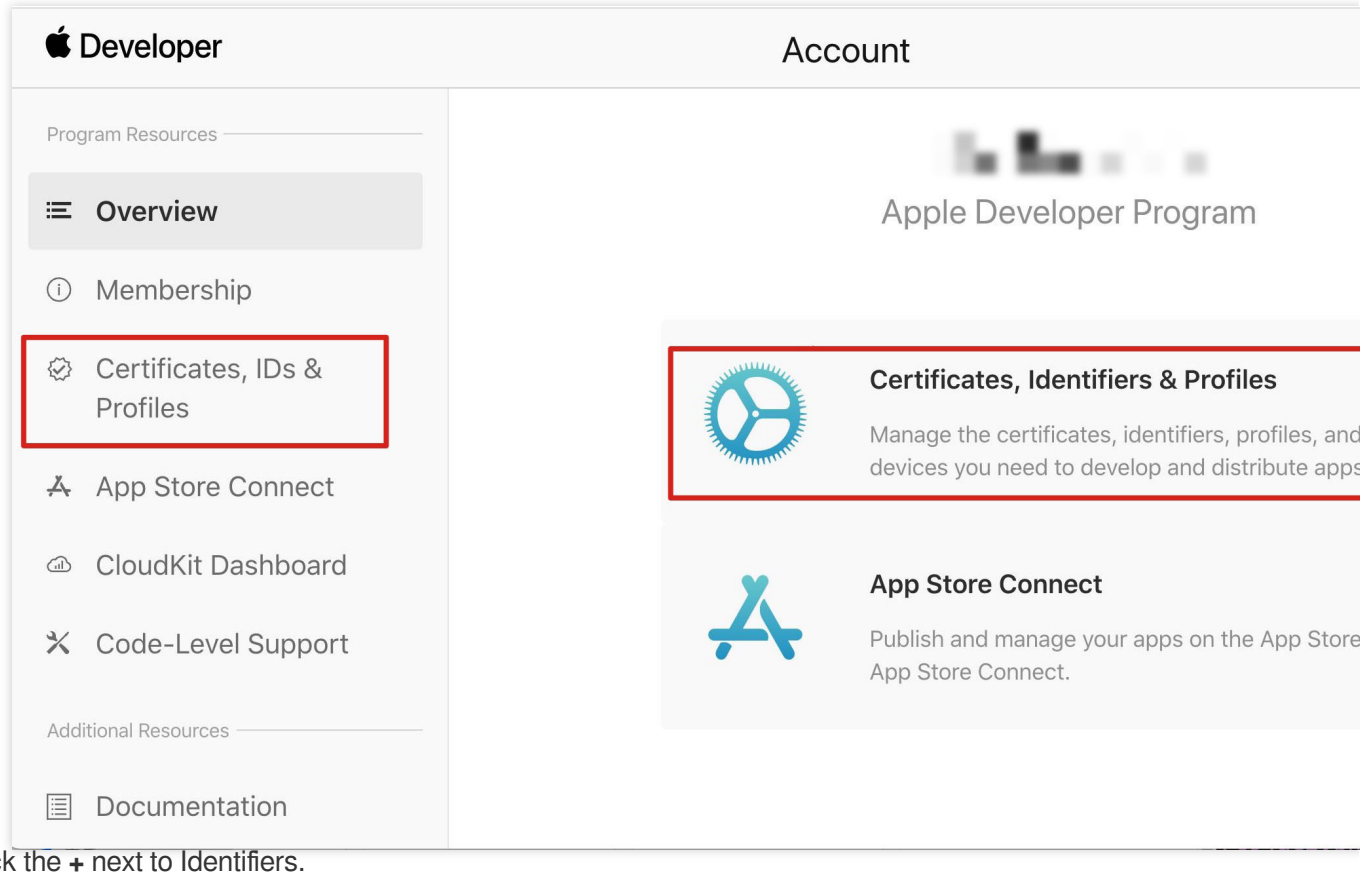
p8 Certificate: Supports Dynamic Island push notifications.

1. Using a p12 certificate (traditional push certificate)

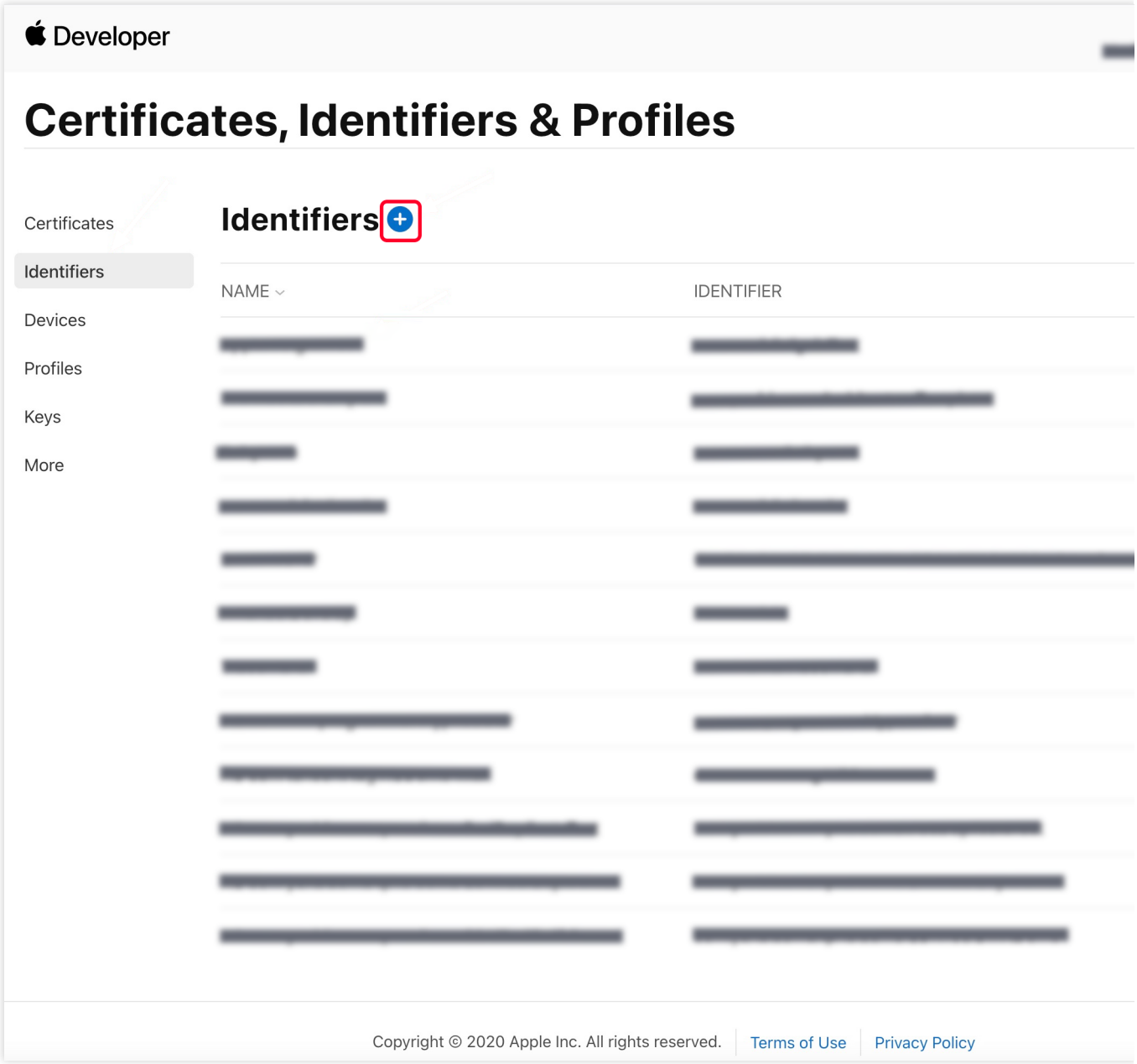
Step 1: Apply for an APNs certificate

Enable remote push for the app

1. log in to Apple Developer Center website, click **Certificates, Identifiers & Profiles** or the sidebar's **Certificates, IDs & Profiles**, enter the **Certificates, IDS & Profiles** page.



2. click the + next to Identifiers.

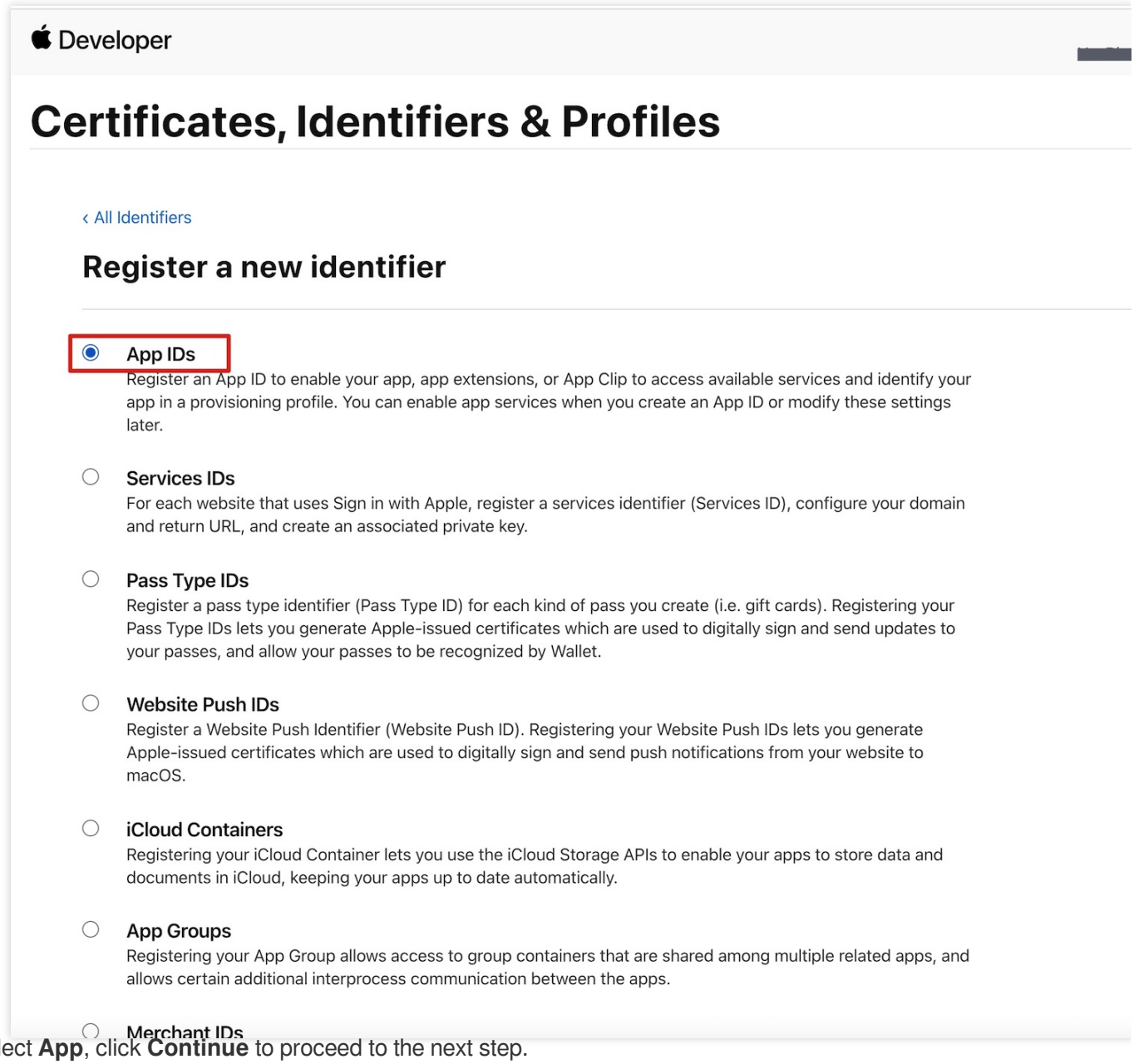


3. You can follow the steps below to create a new AppID or add a `Push Notification Service` to your existing AppID.

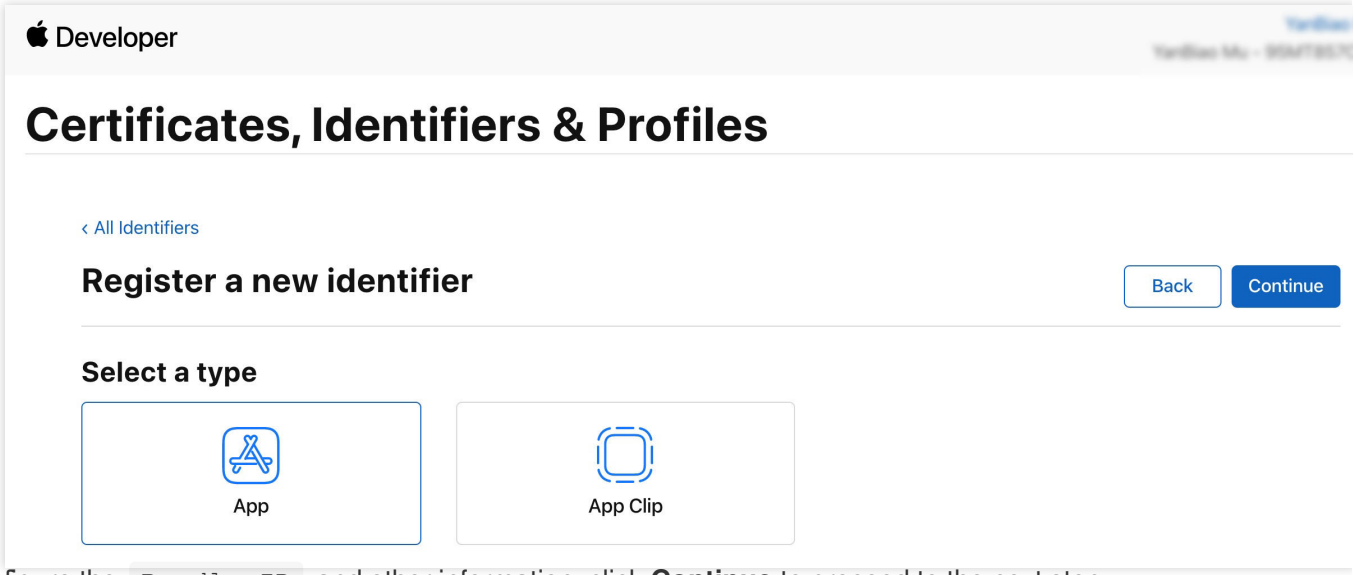
Note:

Your App's `Bundle ID` cannot use the wildcard `*`, otherwise, the remote push service cannot be used.

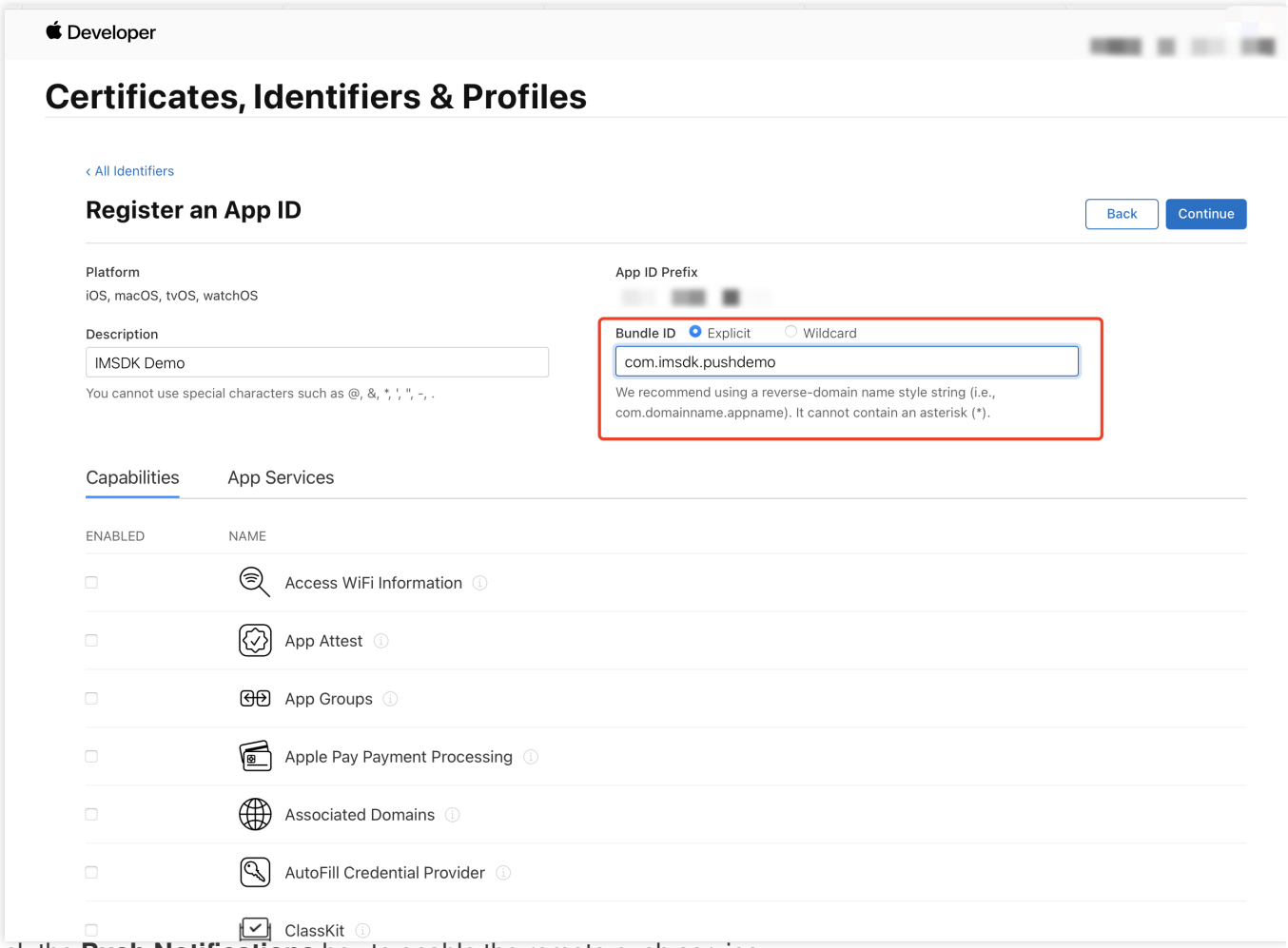
4. Check the **App IDs** box, click **Continue** to proceed to the next step.



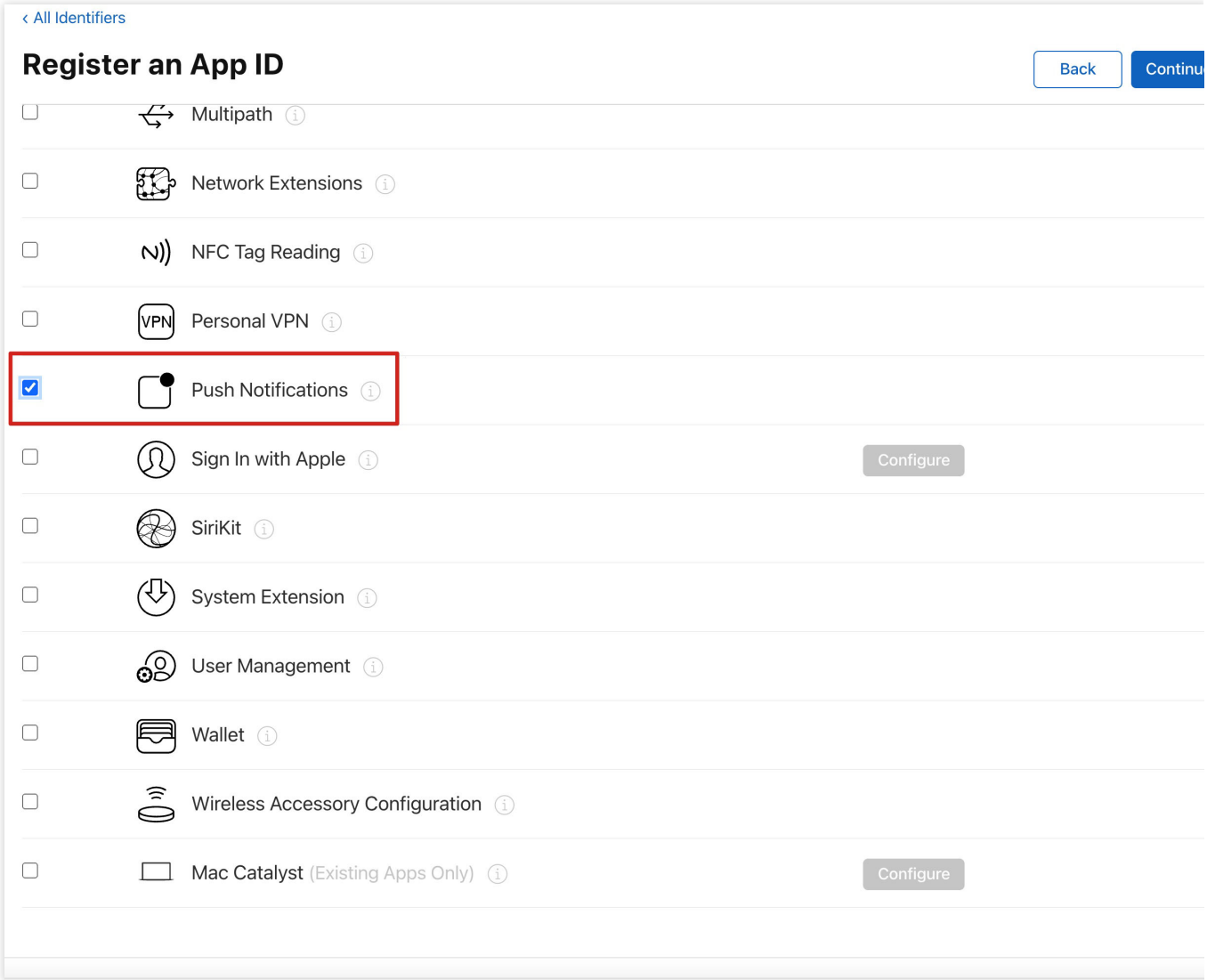
5. Select **App**, click **Continue** to proceed to the next step.



6. Configure the `Bundle ID` and other information, click **Continue** to proceed to the next step.

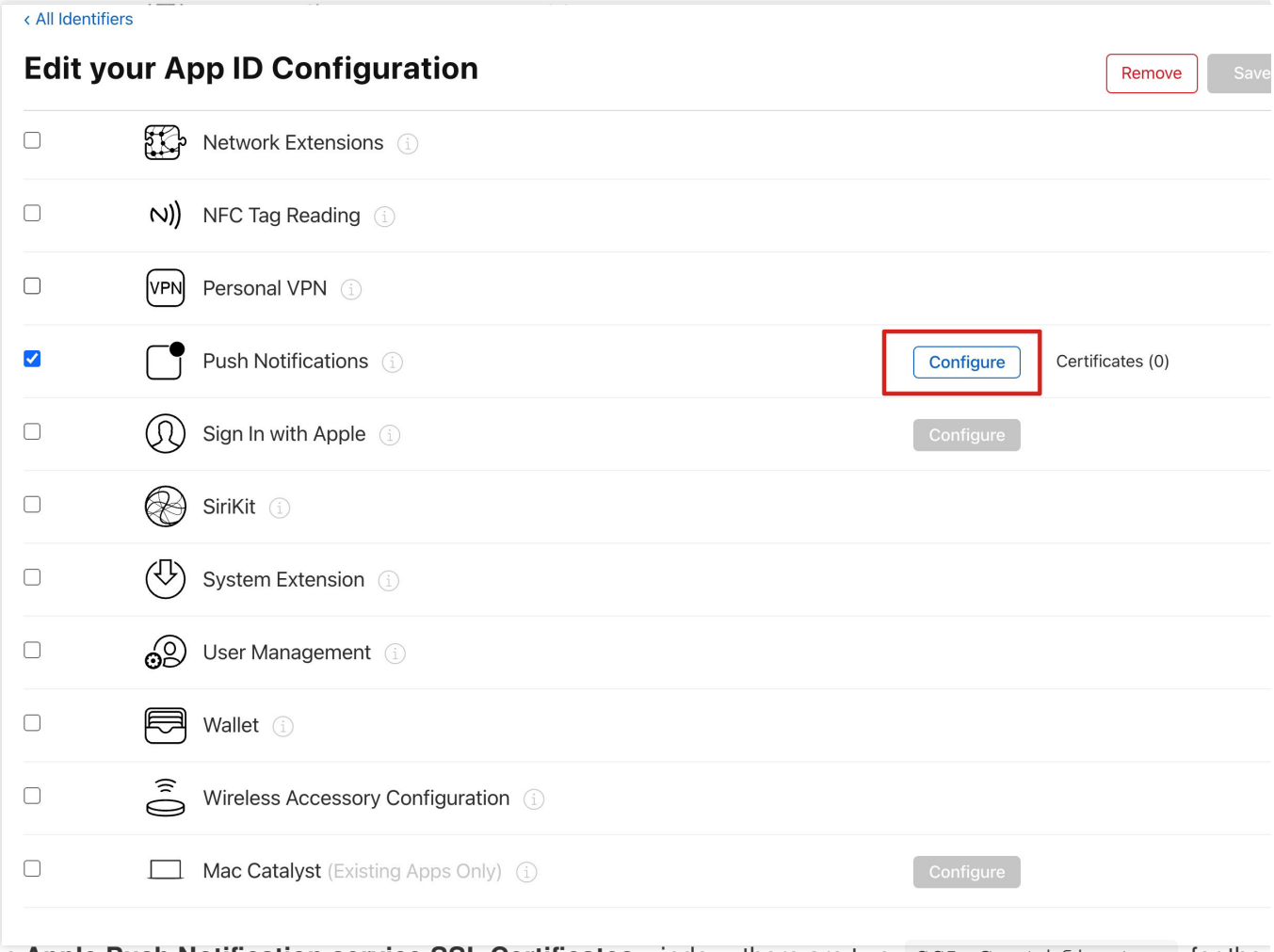


7. Check the **Push Notifications** box to enable the remote push service.

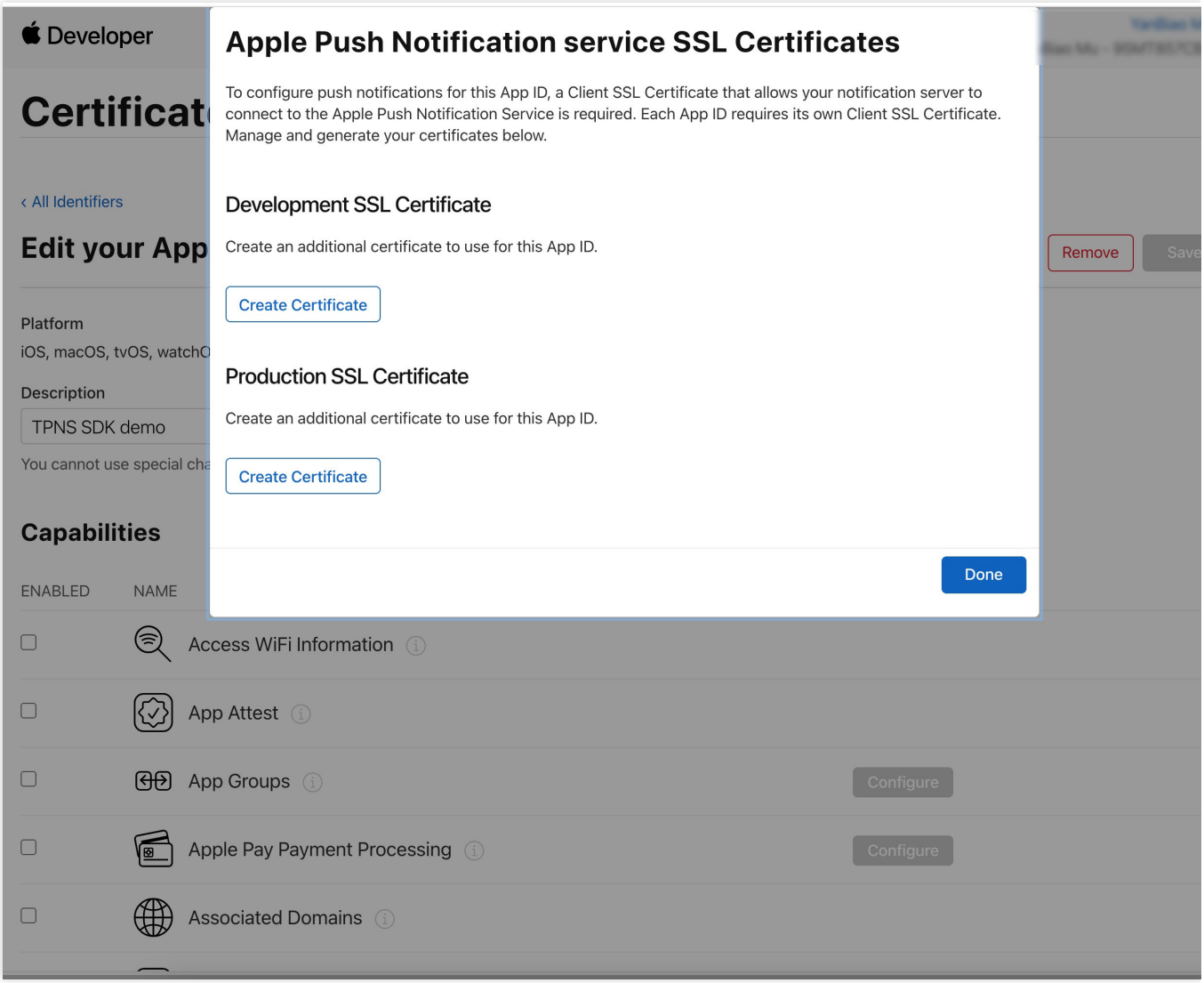


Certificate Generation

1. Select your AppID and choose **Configure**.

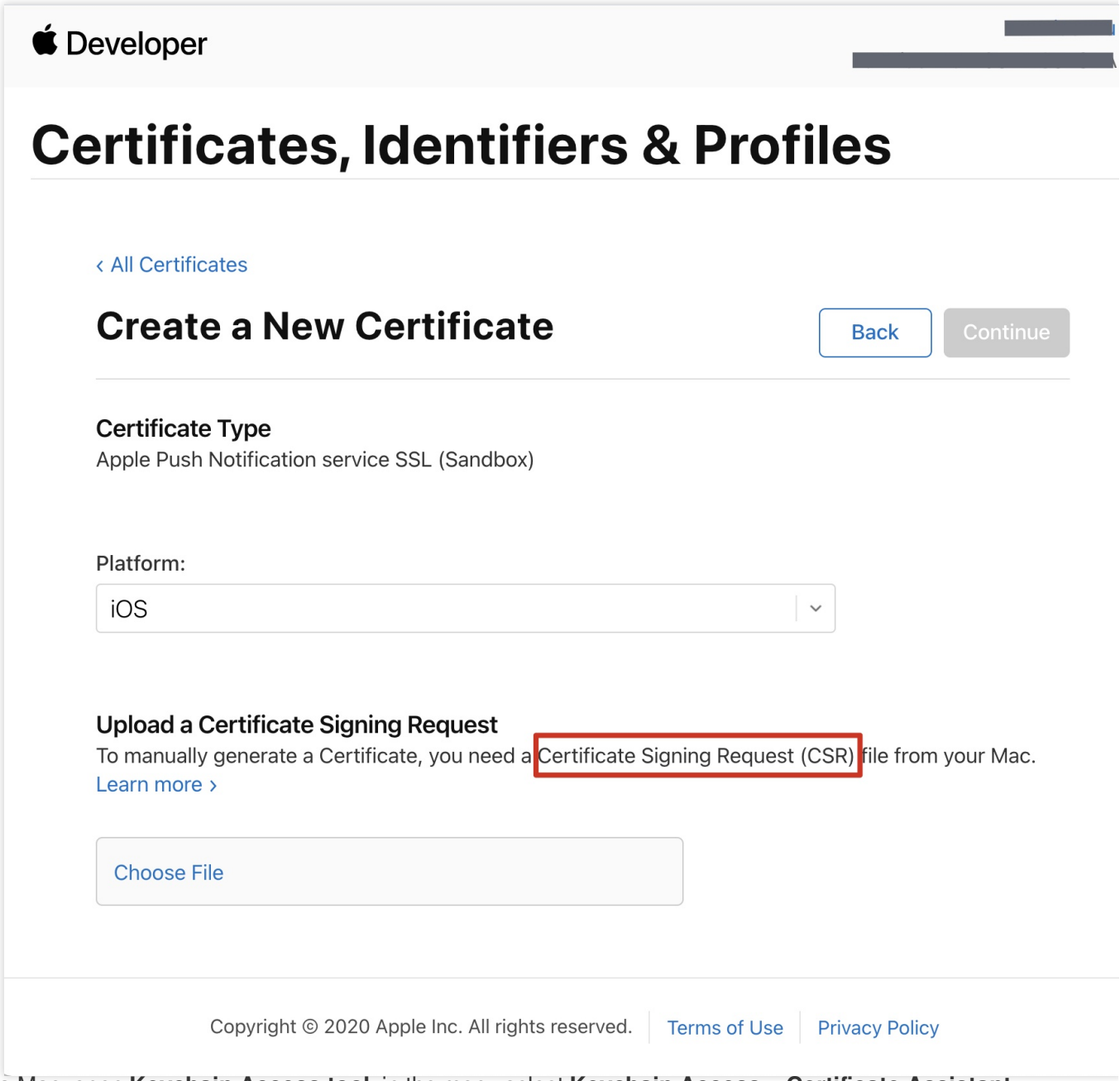


2. In the **Apple Push Notification service SSL Certificates** window, there are two `SSL Certificates` for the development environment (Development) and the production environment (Production), as shown below:

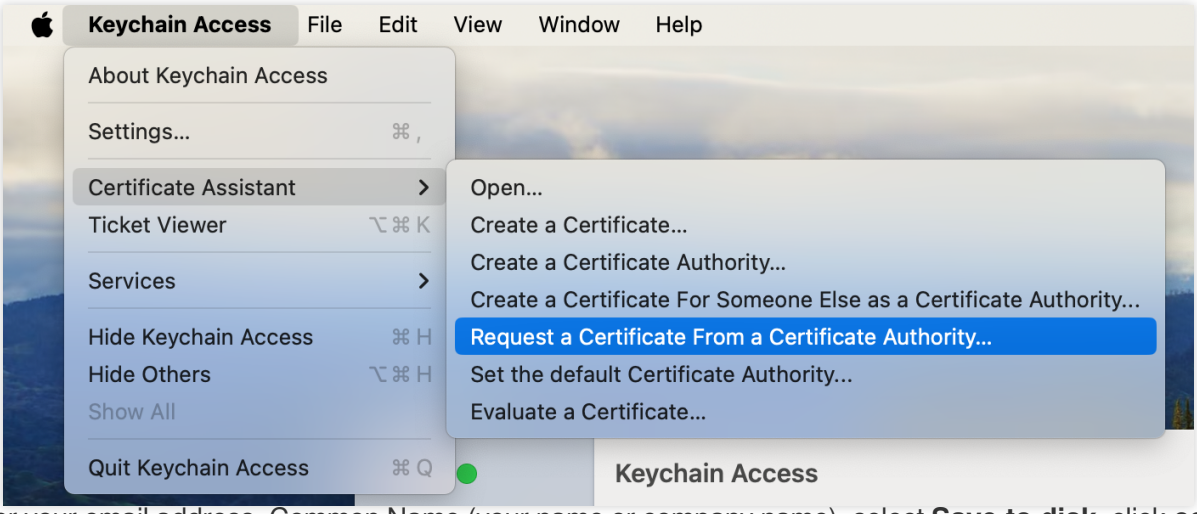


3.

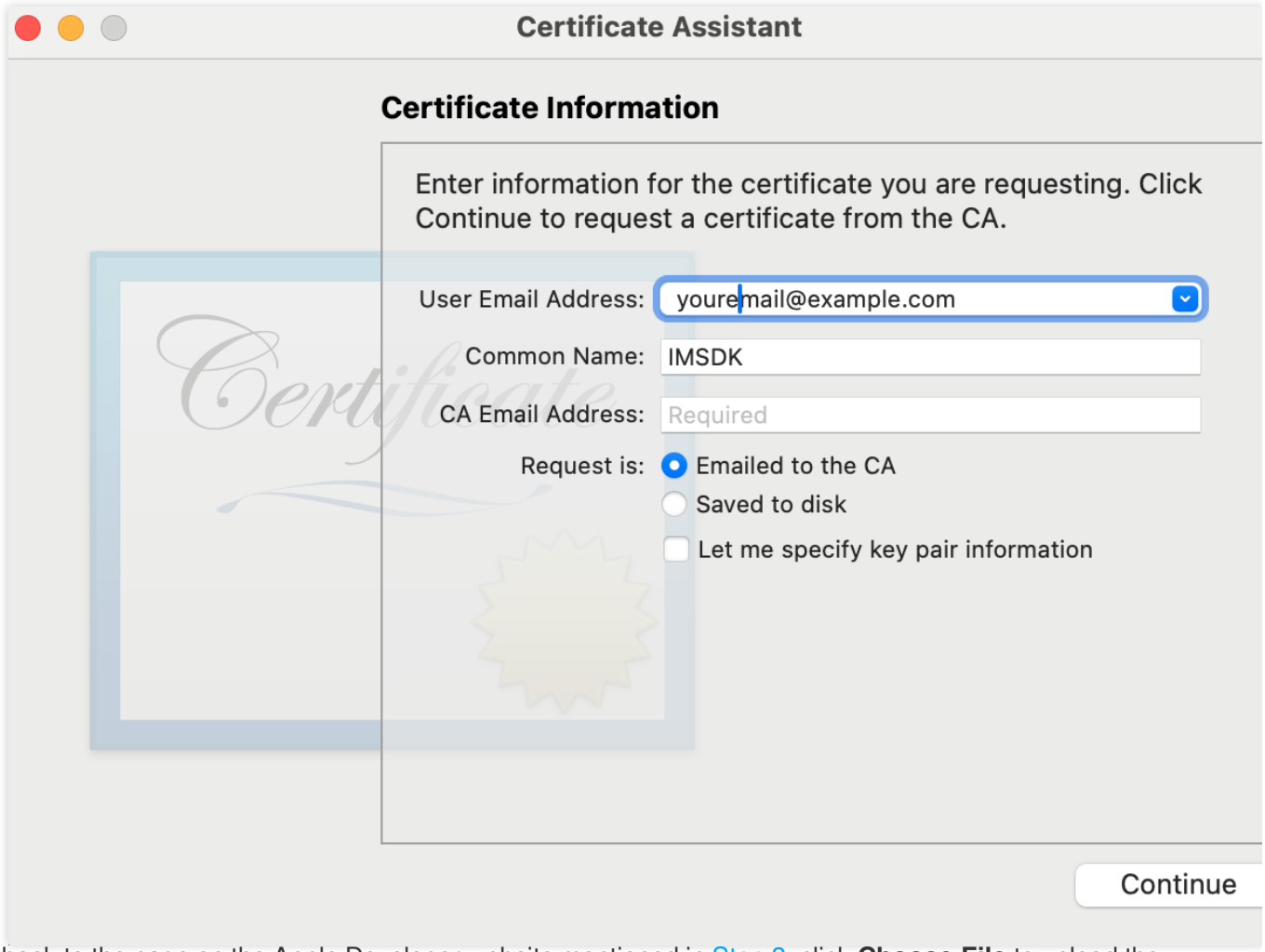
We first select the **Create Certificate** for the Development environment, the system will prompt us that we need a Certificate Signing Request (CSR).



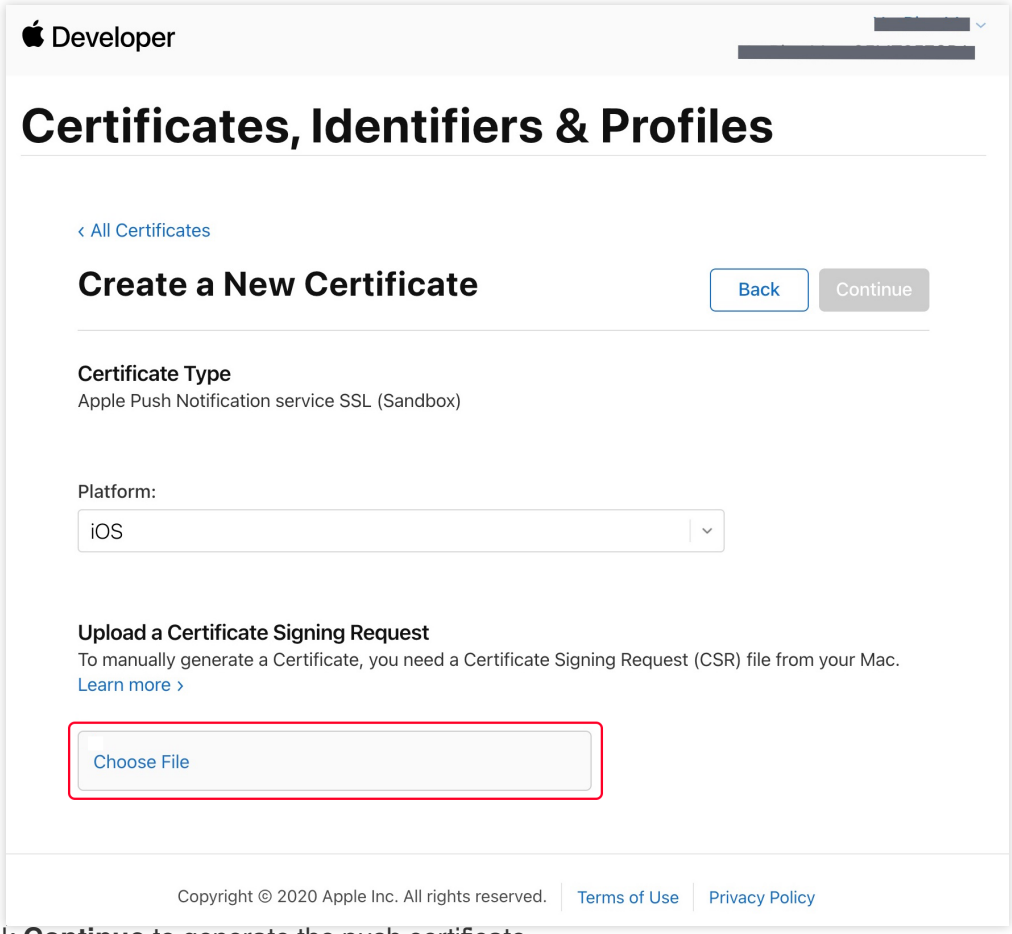
4. On a Mac, open **Keychain Access tool**, in the menu select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority** (`Keychain Access - Certificate Assistant - Request a Certificate From a Certificate Authority`).



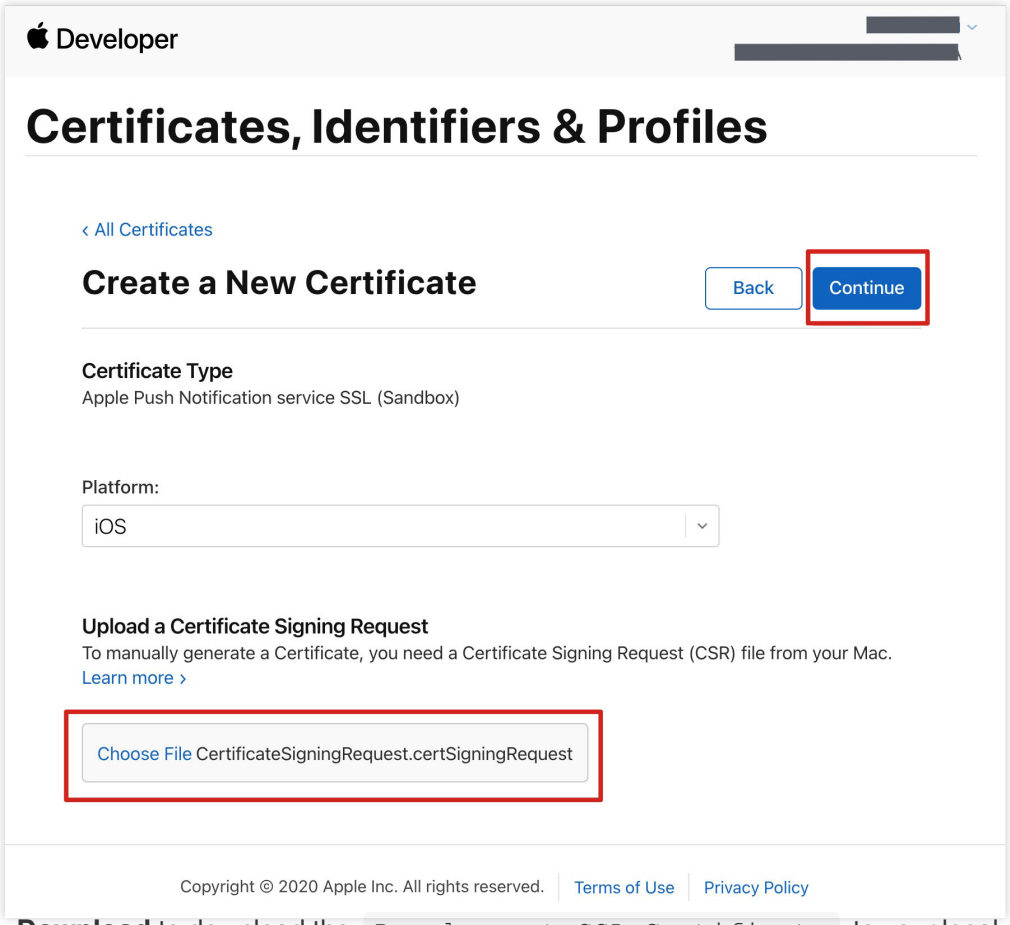
5. Enter your email address, Common Name (your name or company name), select **Save to disk**, click **continue**, the system will generate a `*.certSigningRequest` file.



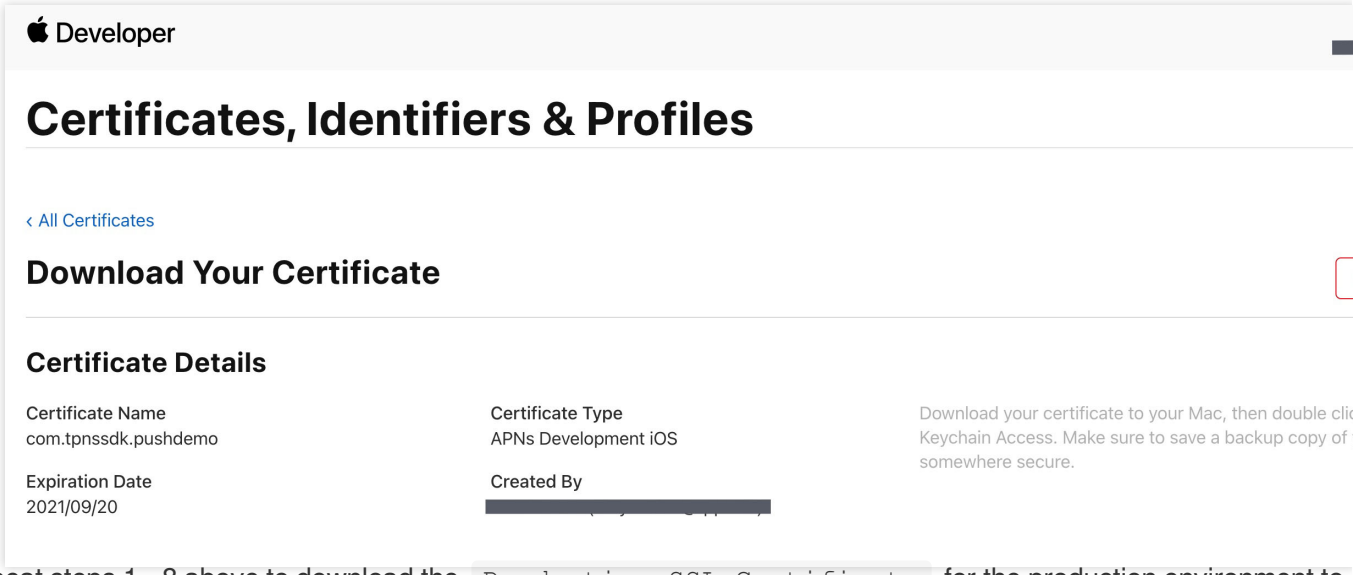
6. Go back to the page on the Apple Developer website mentioned in Step 3, click **Choose File** to upload the generated `*.certSigningRequest` file.



7. Click **Continue** to generate the push certificate.



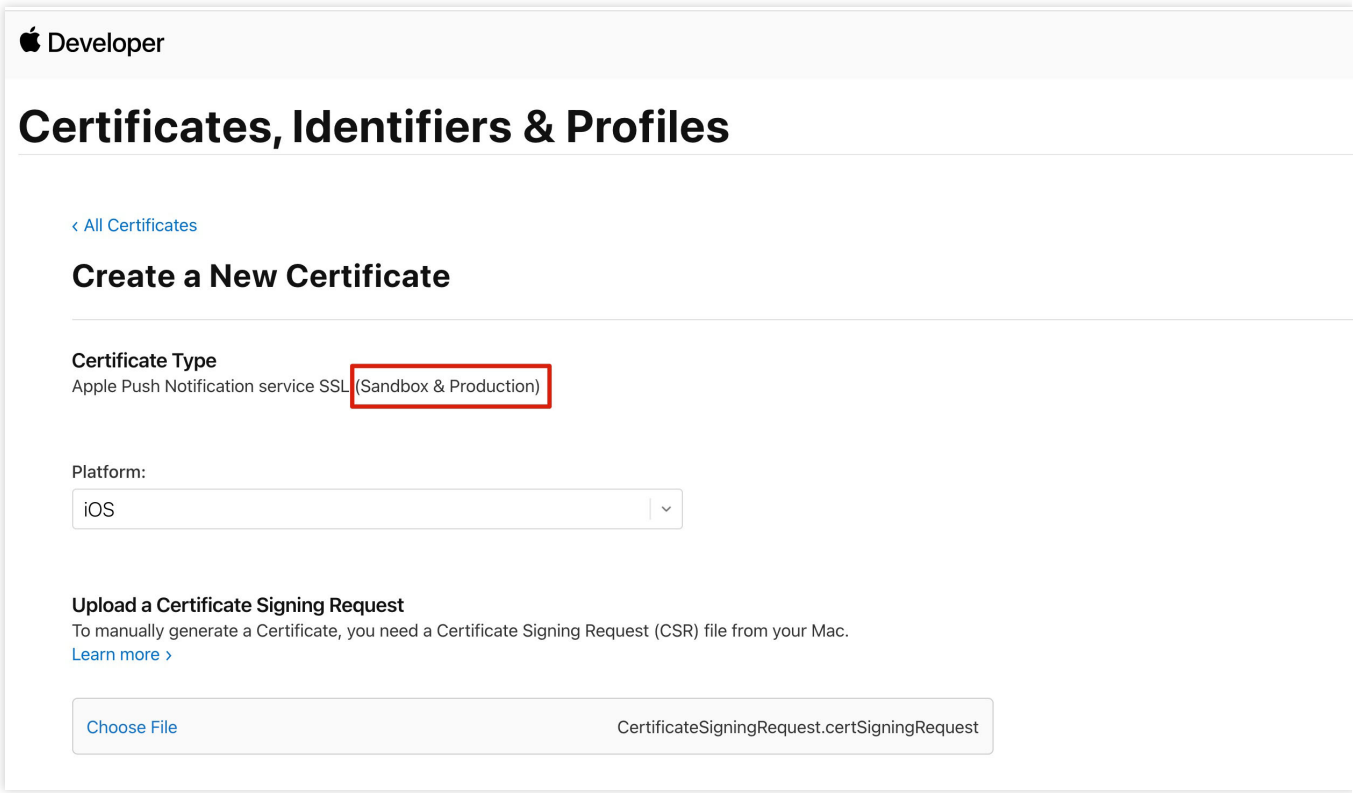
8. click **Download** to download the `Development SSL Certificate` to your local environment.

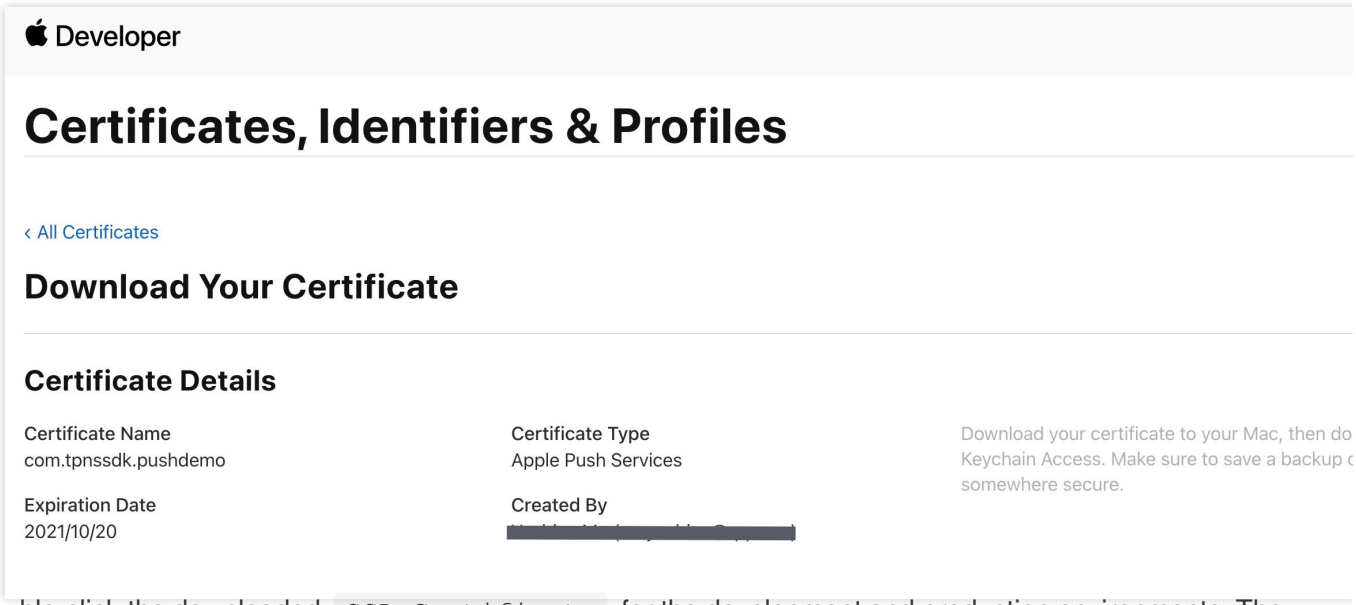


9. Repeat steps 1 - 8 above to download the `Production SSL Certificate` for the production environment to your local machine.

Note:

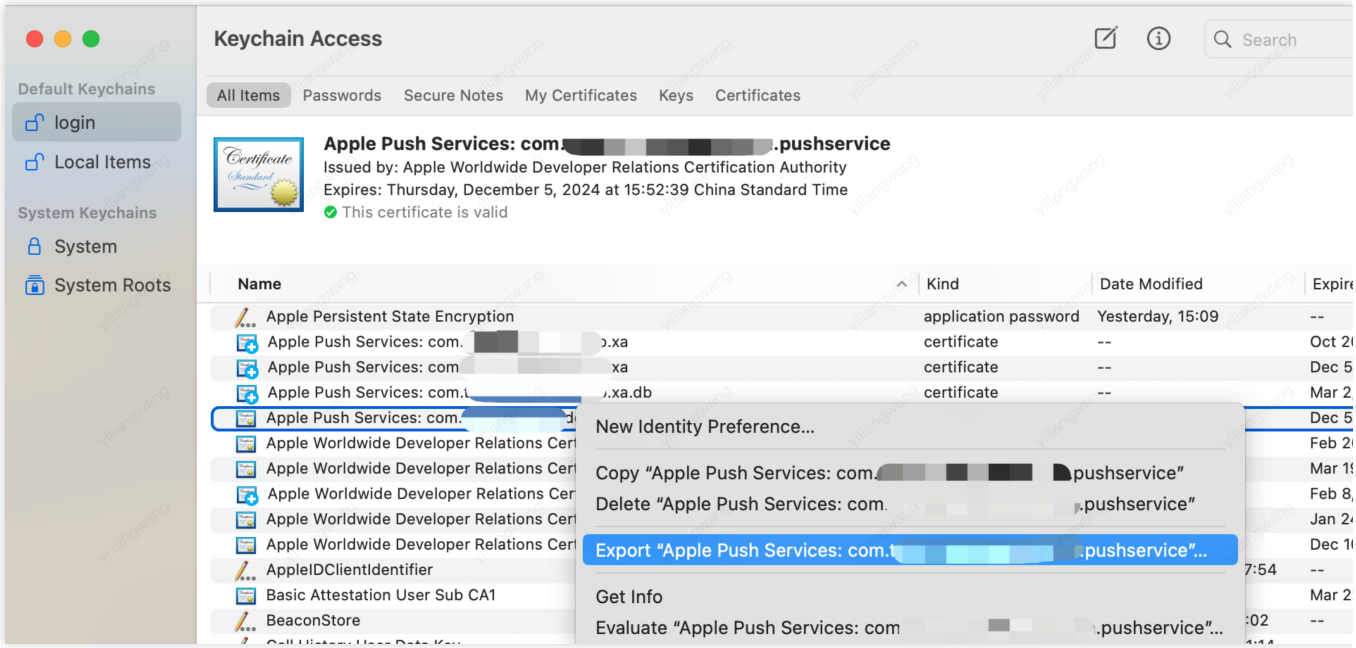
The certificate for the production environment is actually a combined certificate of Development (Sandbox) + Production, and it can be used as a certificate for both the development and production environments.





10. Double-click the downloaded `SSL Certificate` for the development and production environments. The system will import it into the keychain.

11. Open the Keychain App, go to **log in to** > **My Certificates**, right-click to export the newly created `Apple Development IOS Push Services` and `Apple Push Services` for the development and production environments as `p12` files respectively.

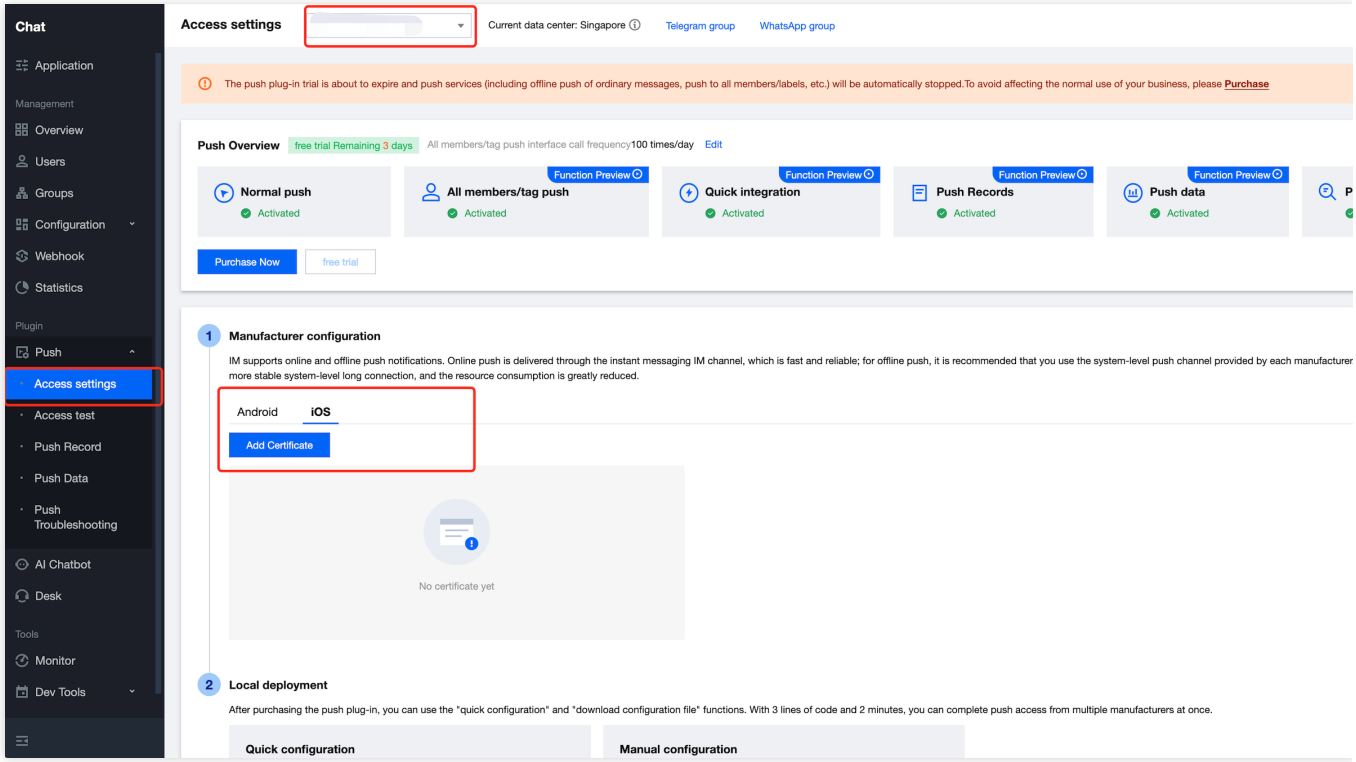


Note

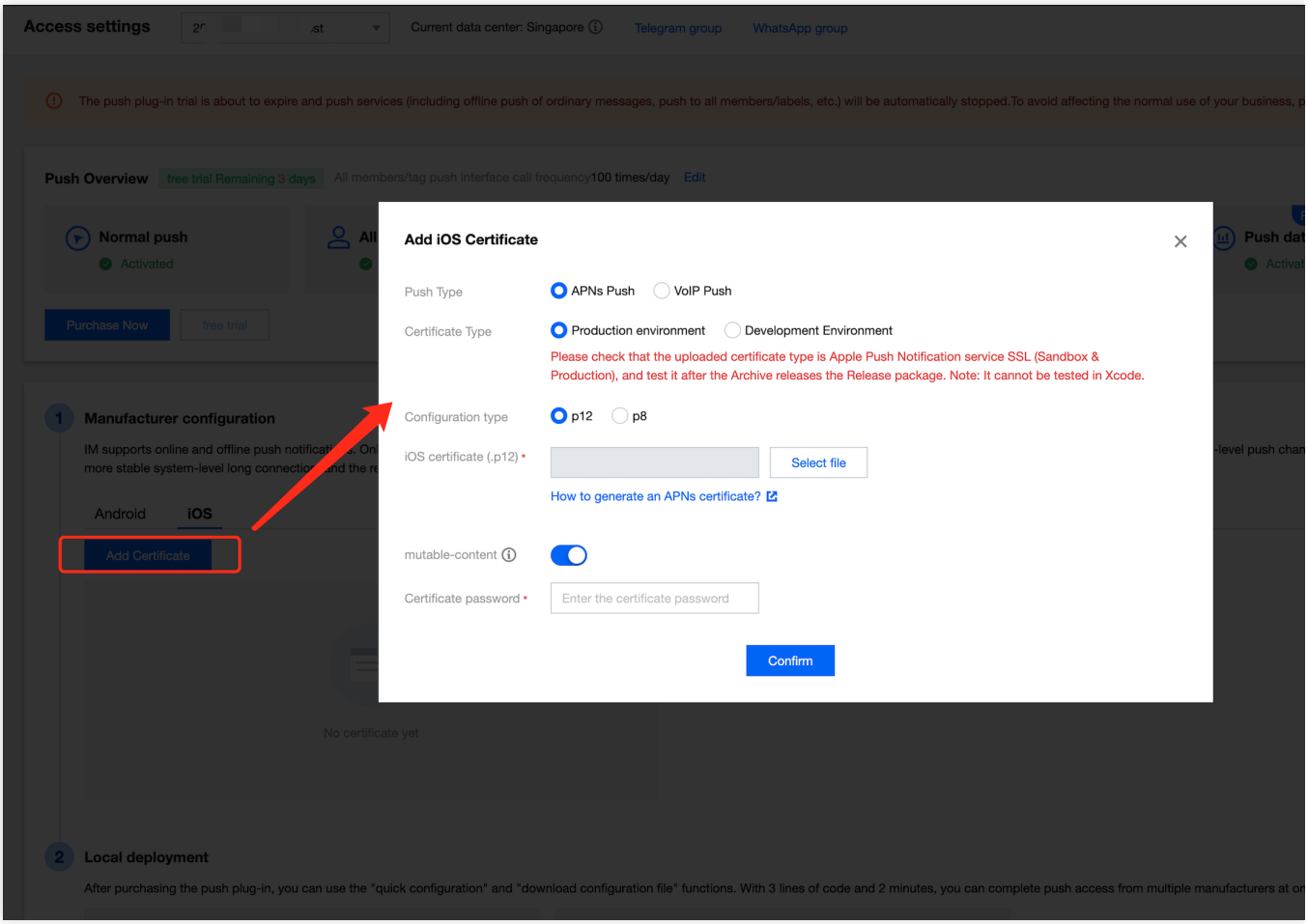
Set a password when saving the `.p12` file.

Step 2: Upload the certificate to the console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page



3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the certificate type, upload the iOS Certificate (.p12), set the certificate password, and click **Confirm**.



Note:

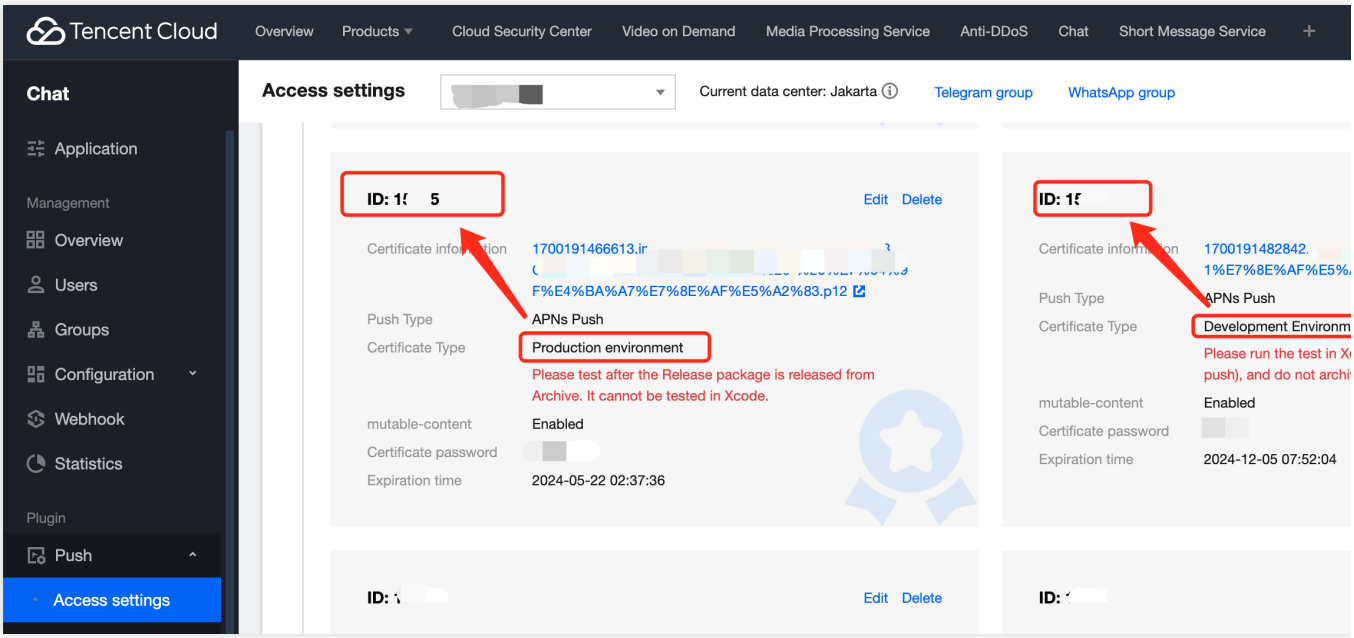
We recommend naming the uploaded certificate in English (special characters such as brackets are not allowed).

You need to set a password for the uploaded certificate. Without a password, push notifications cannot be received.

For an app published on App Store, the environment of the certificate must be the production environment. Otherwise, push notifications cannot be received.

The uploaded .p12 certificate must be your own authentic and valid certificate.

5. After the pending certificate information is generated, record the certificate's ID.

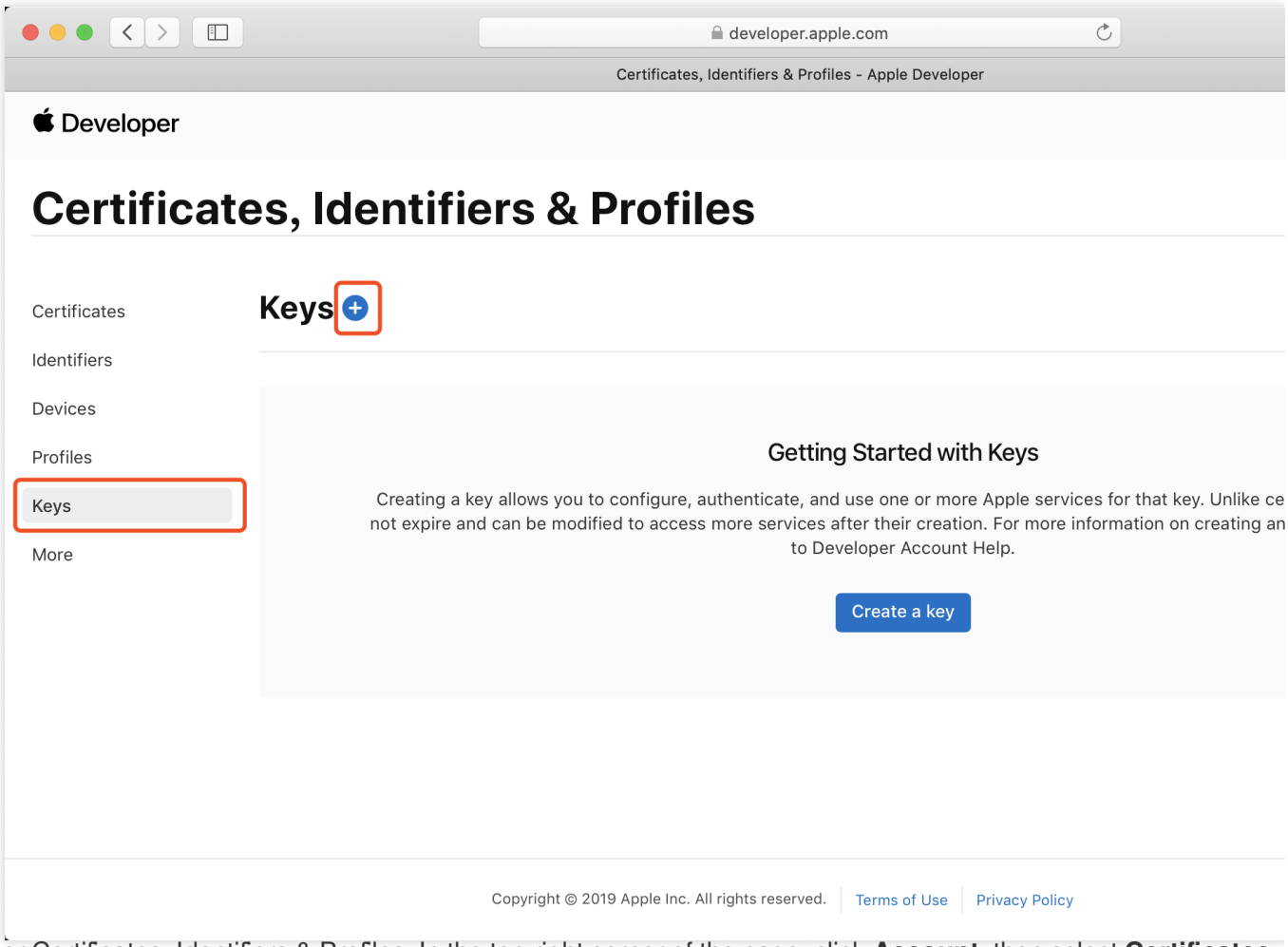


Second, using a p8 certificate (supports Dynamic Island push notifications)

p8 Certificate: A p8 certificate does not have an expiration date, so you don't have to worry about the certificate expiring. Moreover, using a p8 certificate can simplify certificate management, as you can use a single p8 certificate to provide push notification services for multiple applications. In addition, p8 certificates support Dynamic Island push notifications.

Step 1: Apply for an APNs certificate

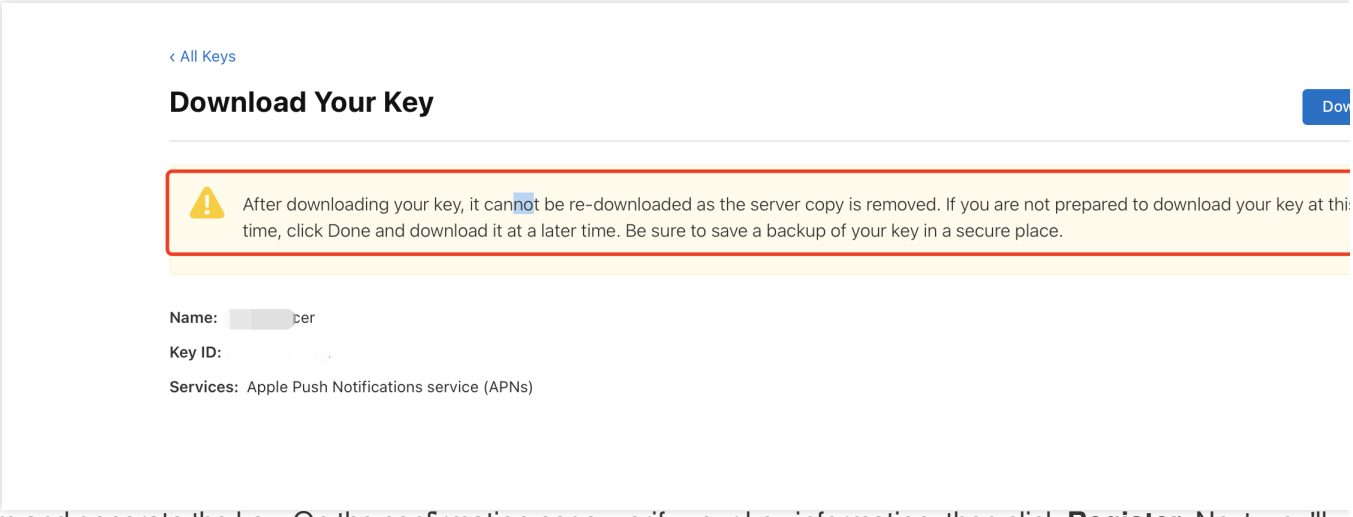
To create a p8 certificate file, first log in to Apple Developer Center.



1. Enter Certificates, Identifiers & Profiles: In the top right corner of the page, click **Account**, then select **Certificates, Identifiers & Profiles** from the dropdown menu.

2. To create a new App ID: in the left-hand menu, click **Identifiers**, then click the **+** on the right to create a new App ID. Fill in the relevant information and click **Continue**.

3. To create a new key: in the left-hand menu, click **Keys**, then click the **+** on the right to create a new key. Enter the name of the key, then check **Apple Push Notifications service (APNs)** and click **Continue**.



Confirm and generate the key: On the confirmation page, verify your key information, then click **Register**. Next, you'll see a page prompting you to download the key. Click **Download** and save the generated .p8 file to your computer.

Note:

The p8 certificate can only be downloaded once; please save it properly.

Please safeguard the downloaded p8 file, as you will not be able to download it again. You can use this p8 certificate to configure your iOS applications to receive push notifications.

Step 2: Upload the p8 certificate to the IM console

1. Log in to the Chat Console.
2. Click Plugin Service-Push-Access Settings to enter the access settings page
3. Click Add Certificate at the bottom of iOS in Vendor Configuration.
4. Select the .p8 certificate

Operation step

Step 1. Register your app with vendor push platforms

To utilize the offline push feature, you need to register your app on each vendor's push platform to obtain parameters such as AppID and AppKey. Currently, the mobile manufacturers supported in China include: Mi, Huawei, Honor, OPPO, Vivo, Meizu, and internationally Google FCM is supported.

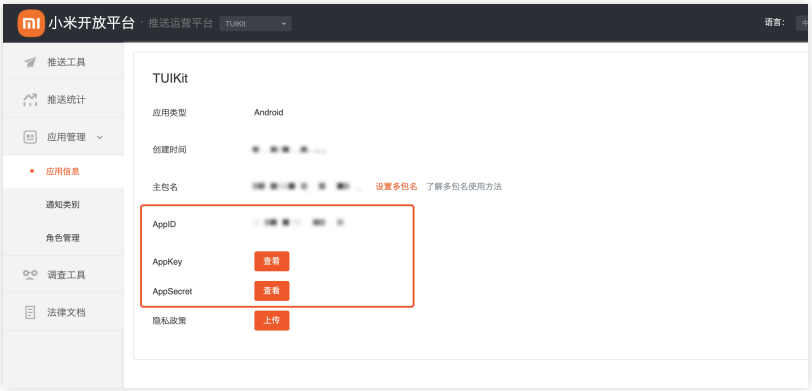
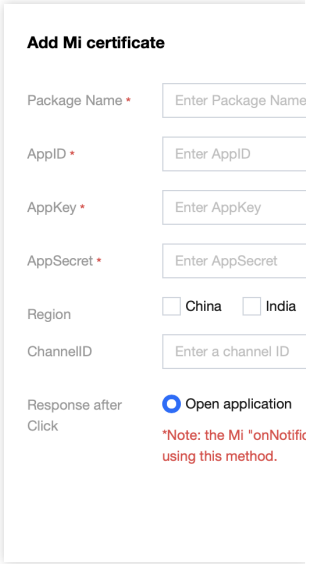
Step 2. Create resources in the IM console

Log in to Tencent Cloud Chat Console, then in the **Push Management** > **Access Settings** feature section, add each vendor's push certificate, and configure the AppID, AppKey, AppSecret, and other parameters obtained in Step 1 to the added push certificate.

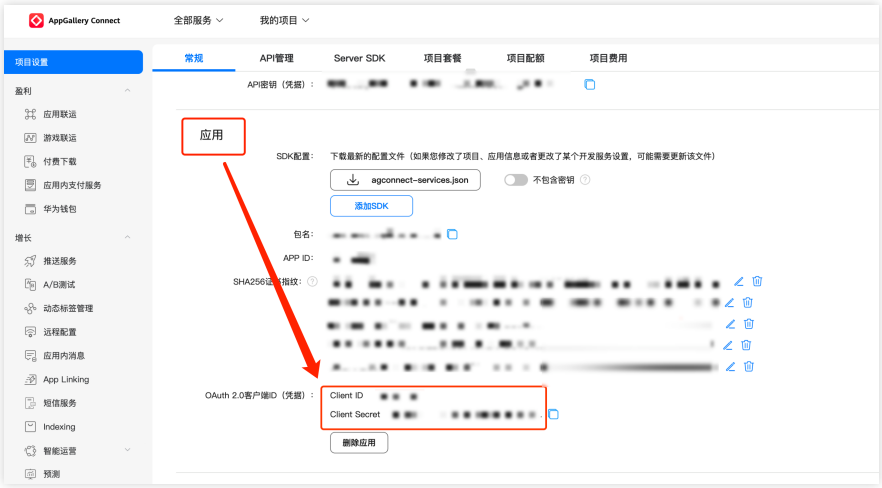
Explanation of the **Subsequent Actions** option:

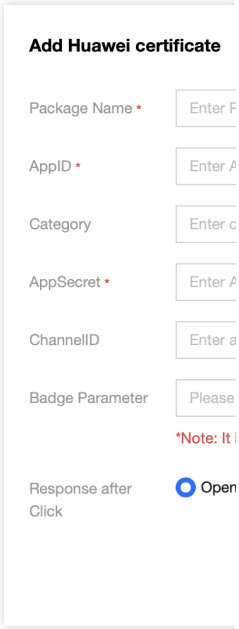
Open Application: Clicking the notification bar launches the app, by default starting the app's Launcher interface;

Open Web Page: Clicking the notification bar will redirect to the configured web link;
Open the specified interface within the app: clicking the notification bar will redirect the interface based on the configured self Definition, see Custom Redirect on Click.
Mi
Huawei
OPPO
vivo
Meizu
HONOR
Google FCM

Vendor Push Platform	Configuring in the IM console
	

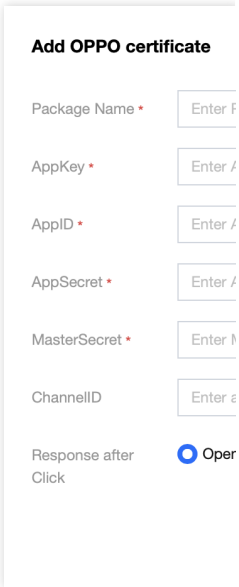
Vendor Push Platform	Configuring in the IM cons



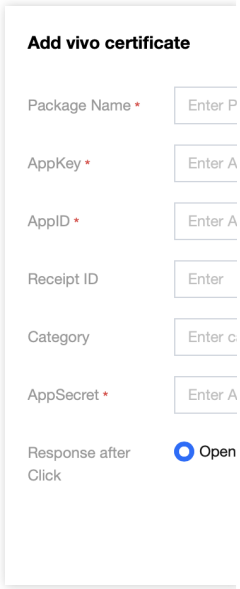


Note:
Client ID corresponds to A

Vendor Push Platform	Configuring in the IM cons

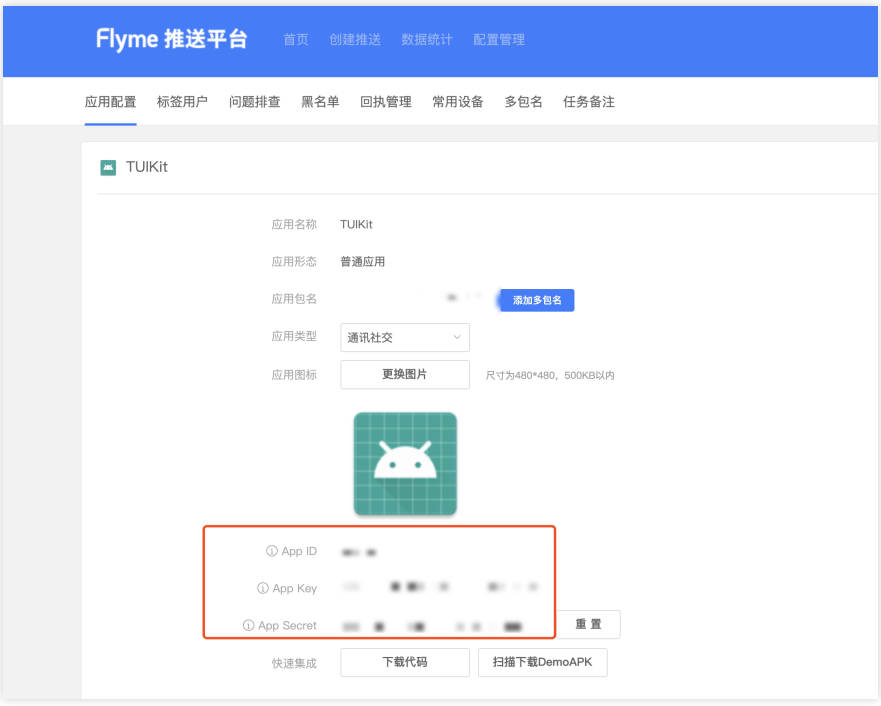


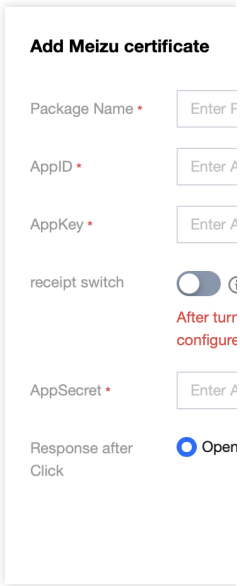
Vendor Push Platform	Configuring in the IM cons



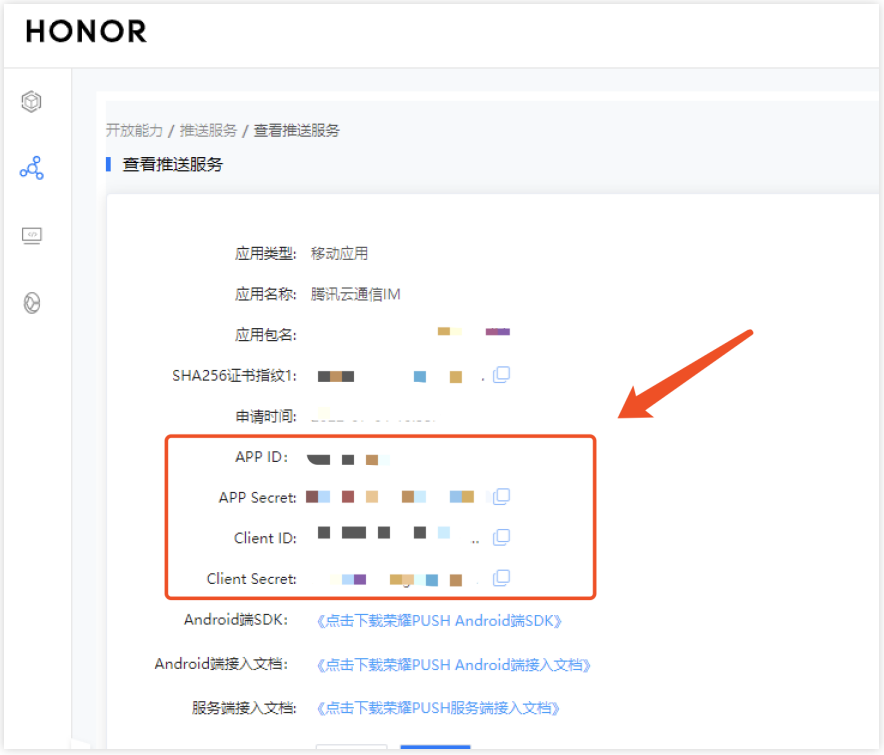
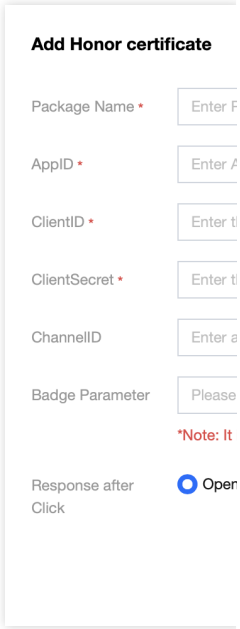
For receipt configuration, please refer to: Message reach statistics configuration - vivo.

Vendor Push Platform	Configuring in the IM cons

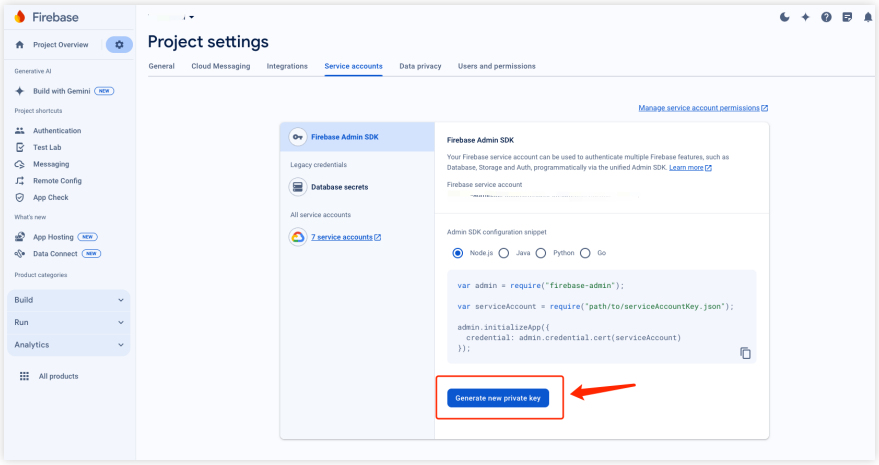


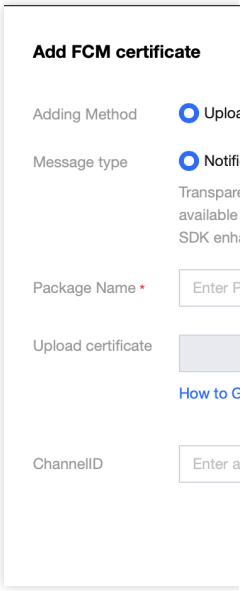


For receipt configuration, please refer to: Message reach statistics configuration - Meizu.

Vendor Push Platform	Configuring in the IM cons
	

Vendor Push Platform	Configuring in the IM cons





Note:

Regarding **Click for Subsequent Actions** supports the Report Statistics feature:

1. If you choose to open an app or a web page, purchasing the plugin will by default support reporting statistics.

2. If you choose to open a specified interface within the application:

For new certificate status, please directly use the auto-fill default value to support click statistics reporting.

If there was a certificate configured previously, continue using the old certificate and modify it to the default value to support reporting statistics, or regenerate a new certificate.

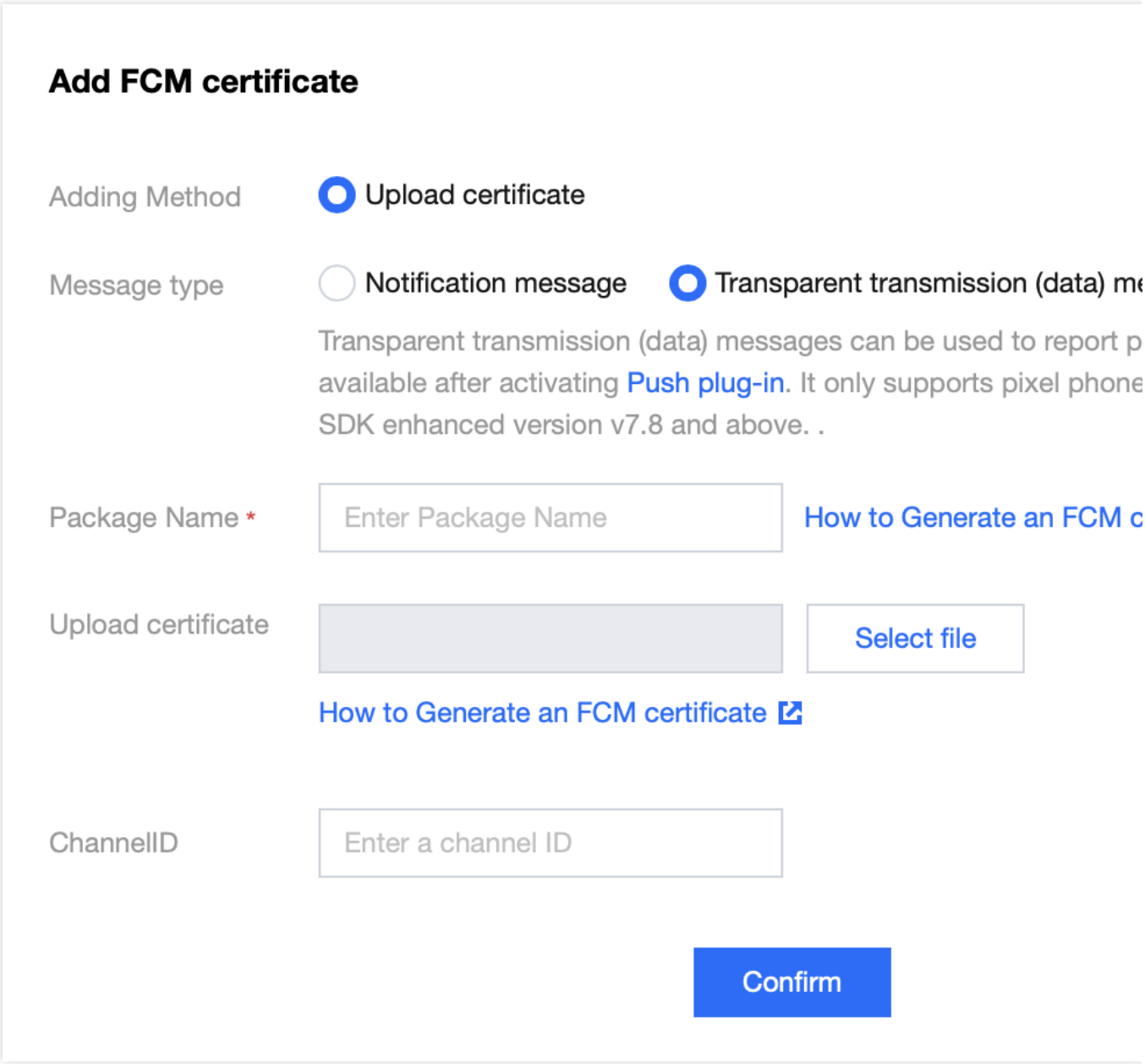
About FCM Data Messaging

FCM provides two push methods: Notification Message and Data Messaging.

Notification Message, with a simple style not differentiated by device, can support offline push once successfully integrated;

Data Messaging, offering rich customization for specific devices, supports reach and click reporting, and requires testing on the device before going live after integration.

The console defaults to Notification Message, and switching between modes can be done in the IM Console:



Note:

FCM Data Messaging capability is only supported on Pixel phones with TIMPush 7.8 and above, other manufacturers' devices need to be self-tested for support;

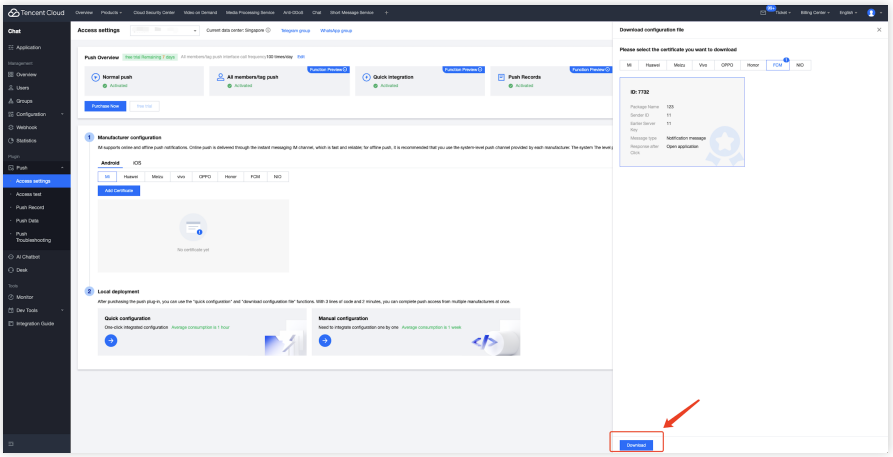
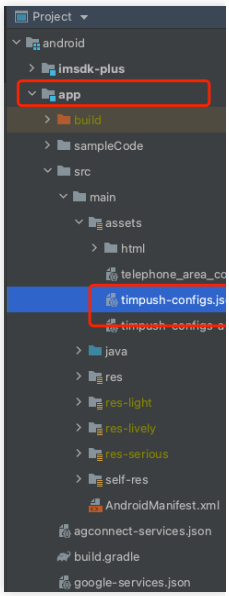
Quick Integration
Android

Last updated：2024-08-01 11:13:26

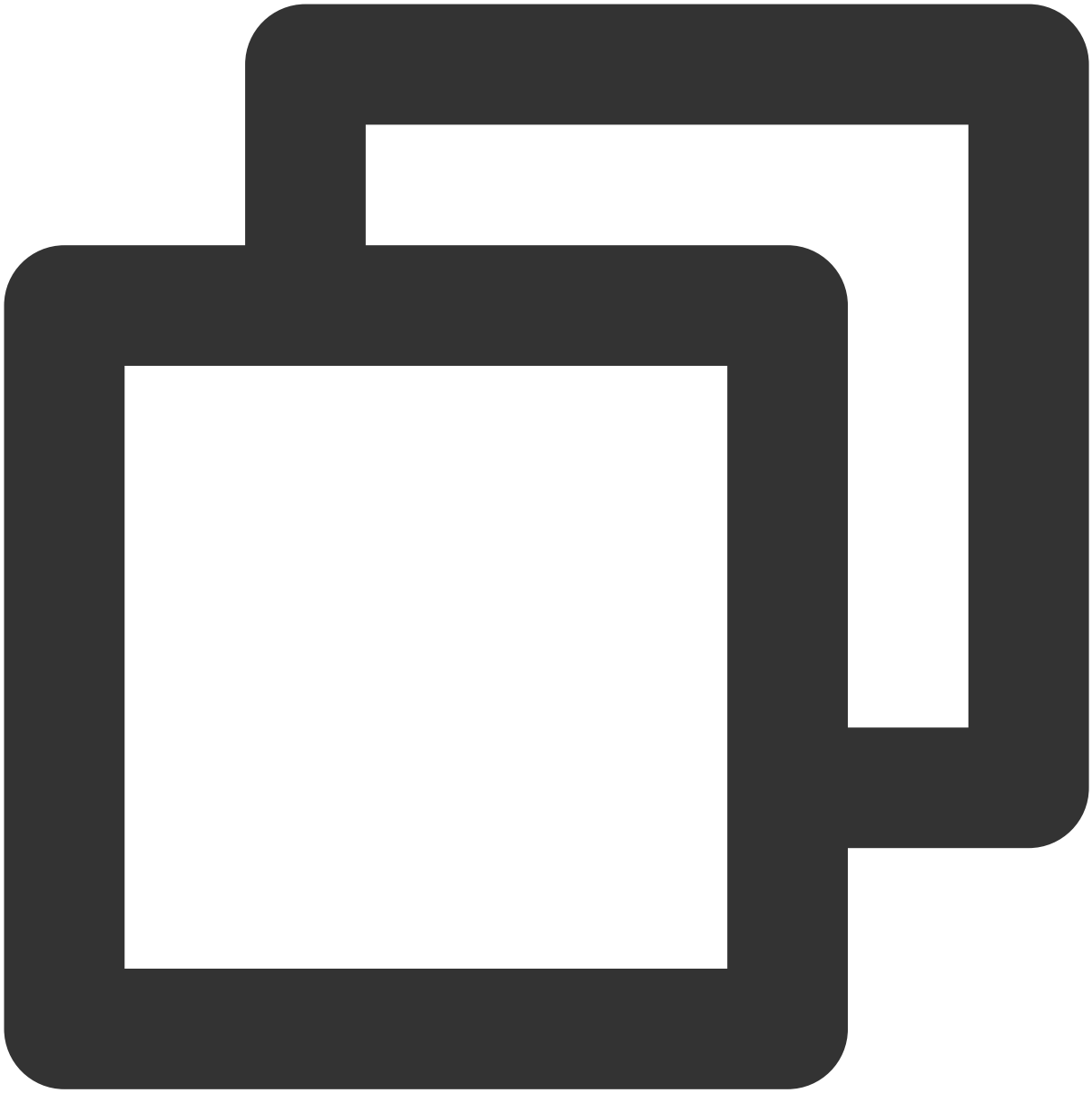
Step 1: Download and add the configuration file

After completing the console manufacturer push information, download and add the configuration file to the project.

Add the downloaded timpush-configs.json file to the assets directory of the application module:

1. Choose to download the configuration file timpush-configs.json	1. Add to the project
	

Step 2: Integrate the TIMPush plugin



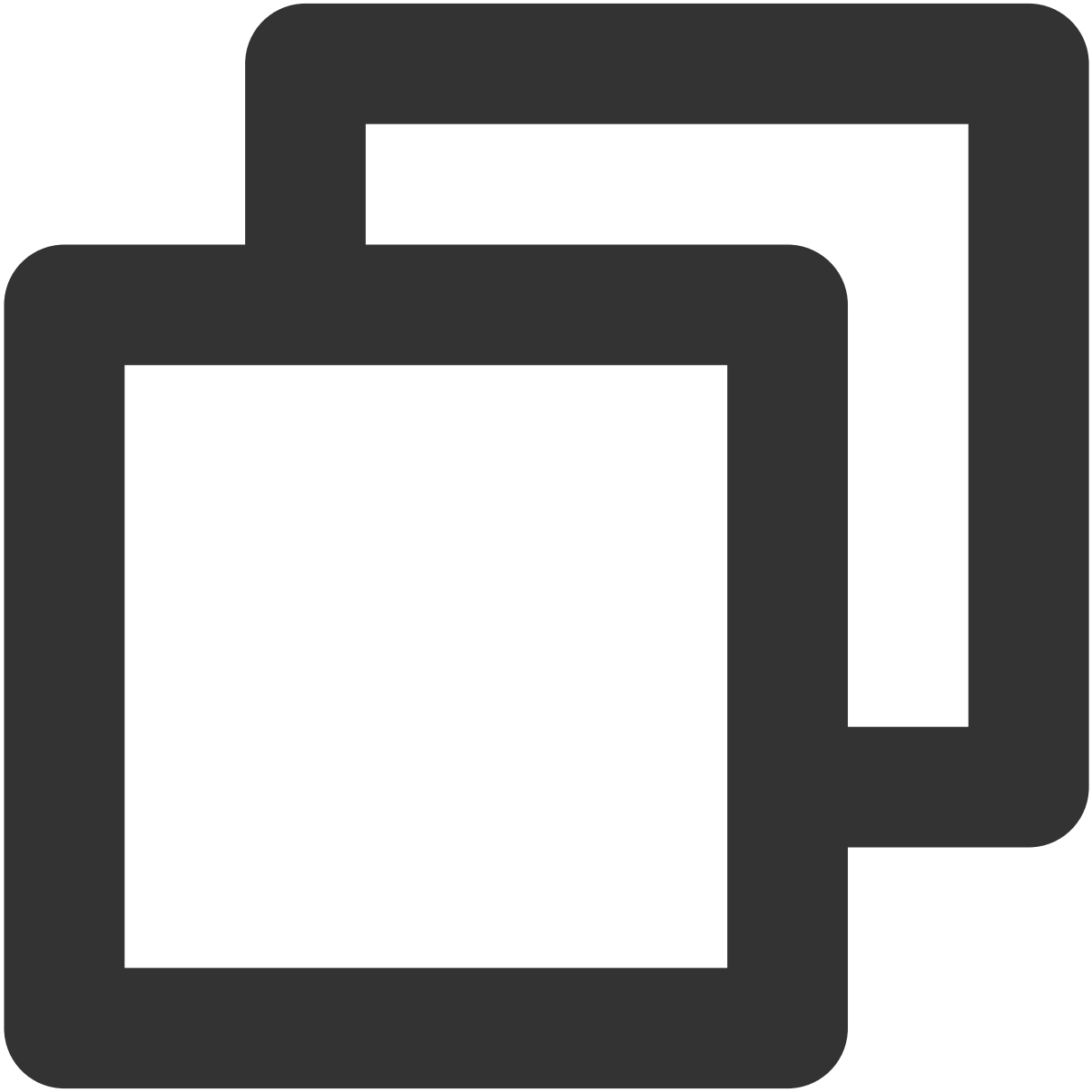
```
// Integration of the push main package is mandatory
implementation 'com.tencent.timpush:timpush:8.0.6897'
// Integrate the corresponding manufacturer as needed
implementation 'com.tencent.timpush:fcm:8.0.6897'
implementation 'com.tencent.timpush:huawei:8.0.6897'
implementation 'com.tencent.timpush:xiaomi:8.0.6897'
implementation 'com.tencent.timpush:vivo:8.0.6897'
implementation 'com.tencent.timpush:honor:8.0.6897'
implementation 'com.tencent.timpush:meizu:8.0.6897'
// Choose one of the two below
// For the China region, choose to integrate this package
```

```
implementation 'com.tencent.timpush:oppo:8.0.6897'
// For other regions, choose to integrate this package
implementation 'com.tencent.timpush:oppo-intl:8.0.6897'
```

Note:

1. TIMPush requires integration with IMSDK version 7.6.5011 or above.

2. For users without UI or who haven't integrated other plugins, it is necessary to add integration with TUICore. It supports both source and Maven integration, as follows:



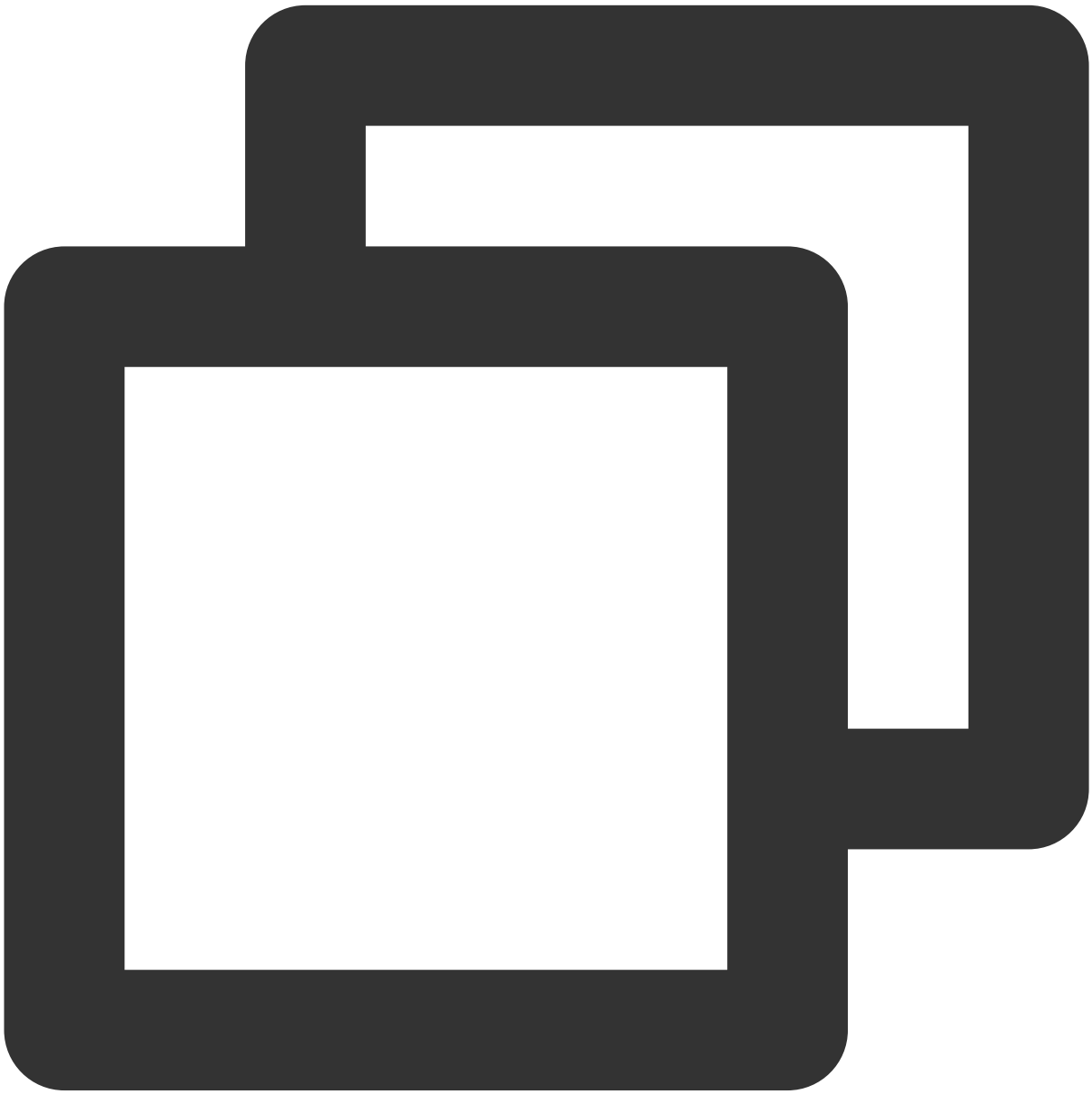
```
def projects = this.rootProject.getAllprojects().stream().map { project -> project.
api projects.contains("tuicore") ? project(':tuicore') : "com.tencent.liteav.tuikit
```

vivo and Honor configuration

According to vivo and Honor manufacturer's access guidance, it is necessary to add the APPID and APPKEY to the manifest file, otherwise compilation issues will occur.

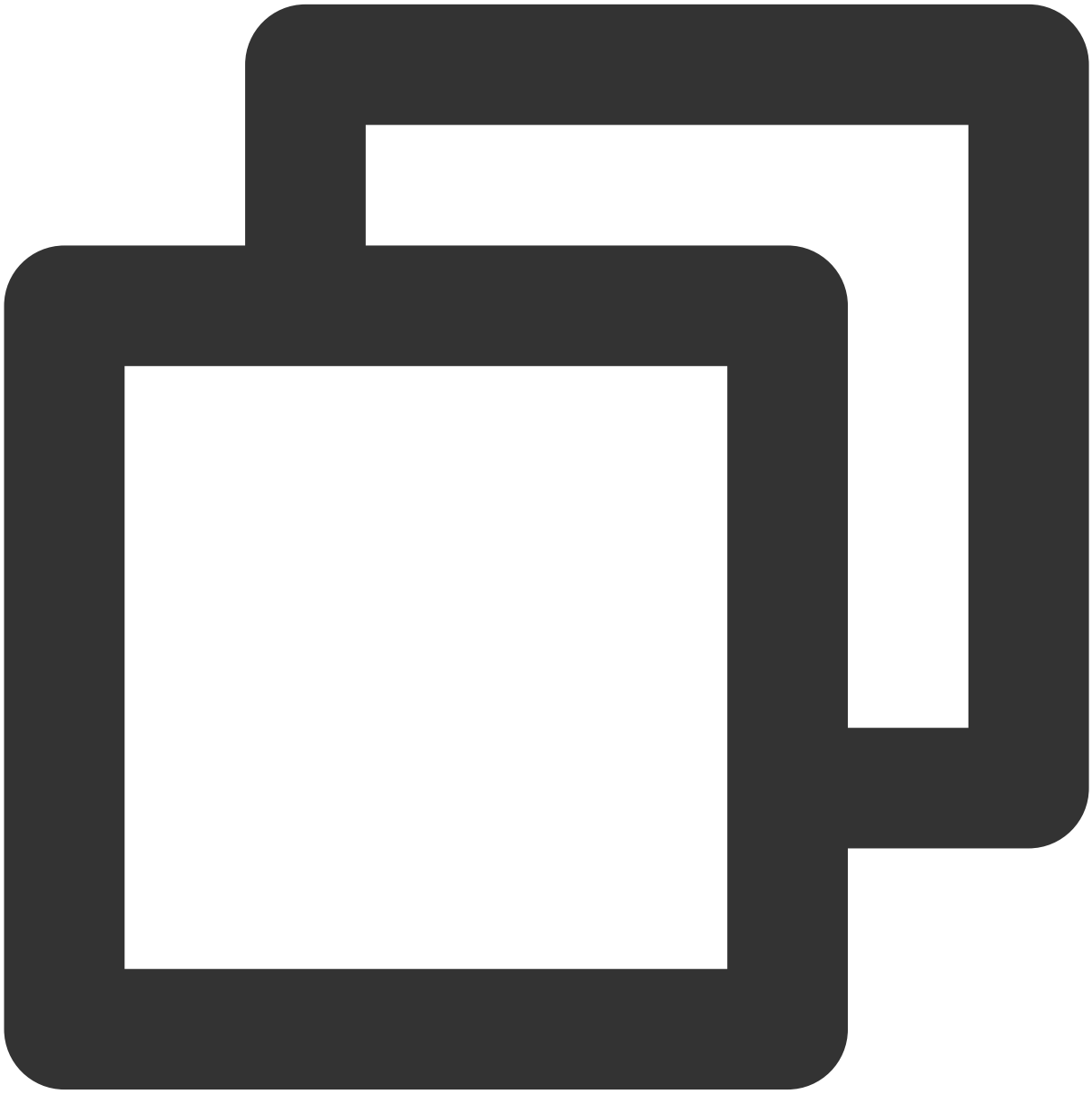
Method 1

Method 2



```
android {
    ...
    defaultConfig {
        ...
```

```
        manifestPlaceholders = [
            "VIVO_APPKEY" : "`APPKEY` of the certificate assigned to your applicati
            "VIVO_APPID" : "`APPID` of the certificate assigned to your application
            "HONOR_APPID" : "`APPID` of the certificate assigned to your applicatio
        ]
    }
}
```



```
// vivo begin
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.api_key"
```

```
    android:value="`APPKEY` of the certificate assigned to your application" />
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.app_id"
    android:value="`APPID` of the certificate assigned to your application" />
// vivo end

// honor begin
<meta-data tools:replace="android:value"
    android:name="com.hihonor.push.app_id"
    android:value="`APPID` of the certificate assigned to your application" />
// honor end
```

Huawei, HONOR, and Google FCM Adaptation

Follow the vendor method to integrate the corresponding plugin and JSON configuration file.

Note:

The following adaptation for HONOR only requires configuration for version 7.7.5283 and above.

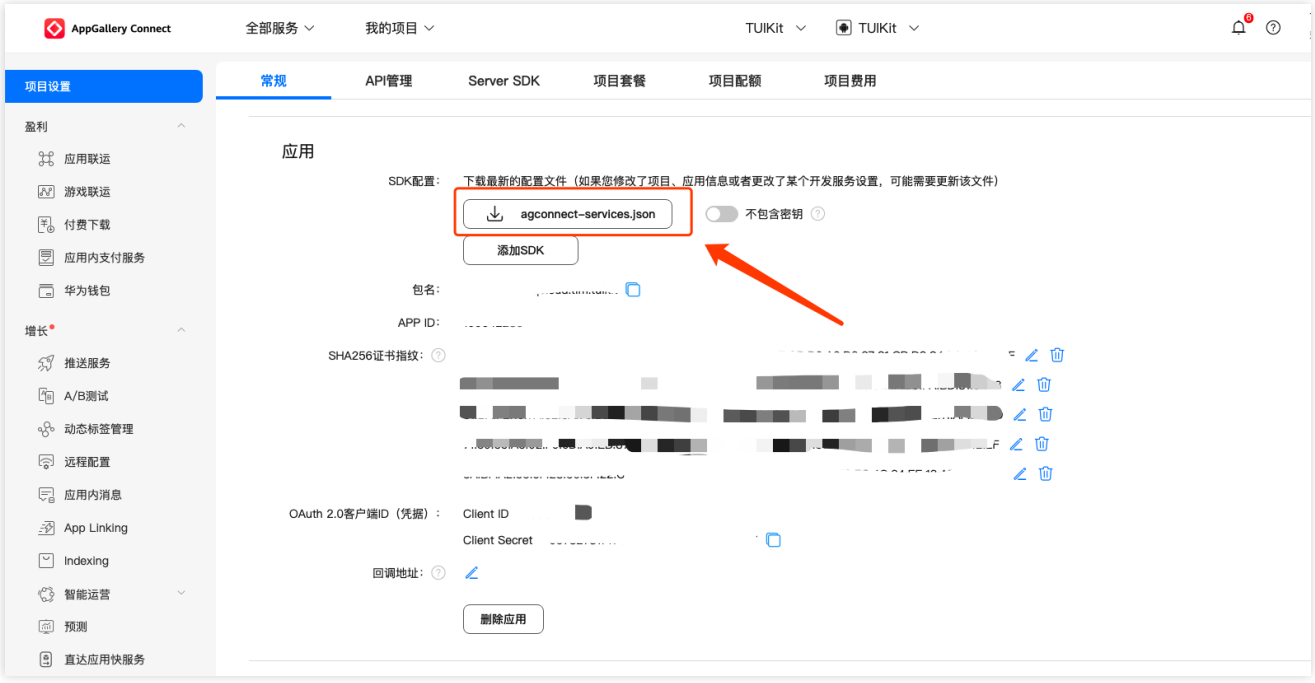
1.1 Download the configuration file and place it under the root directory of the project:

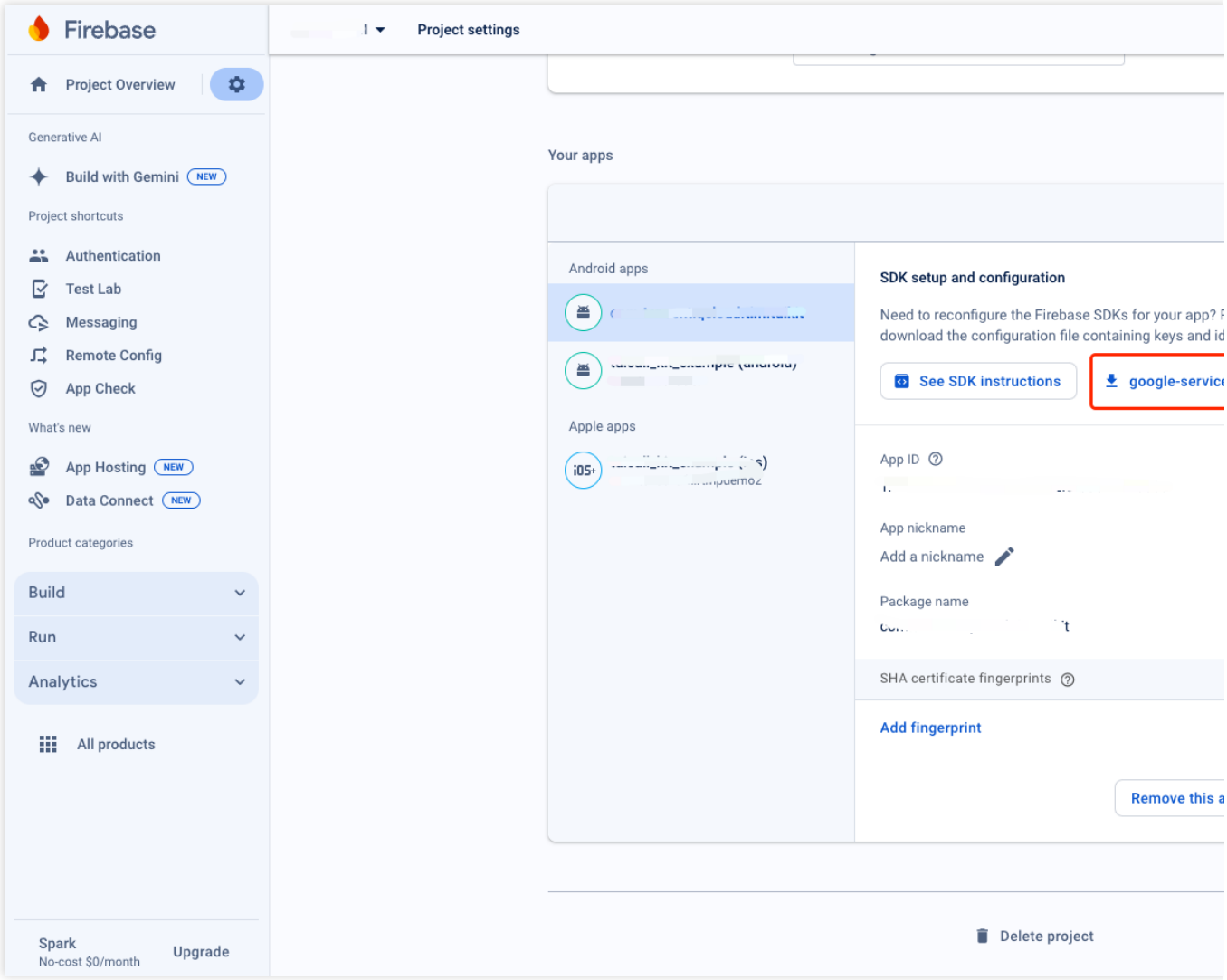
Huawei

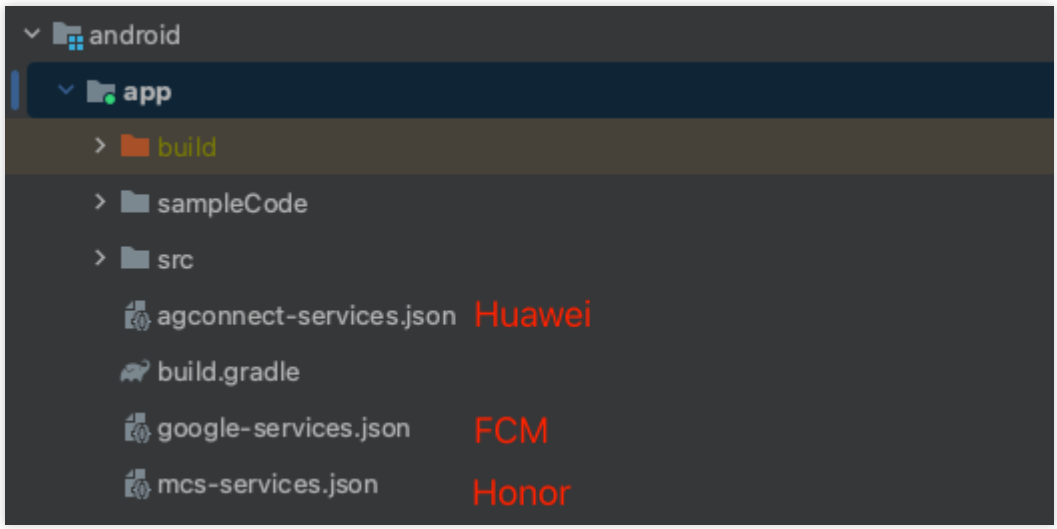
HONOR

Google FCM

Operation Path







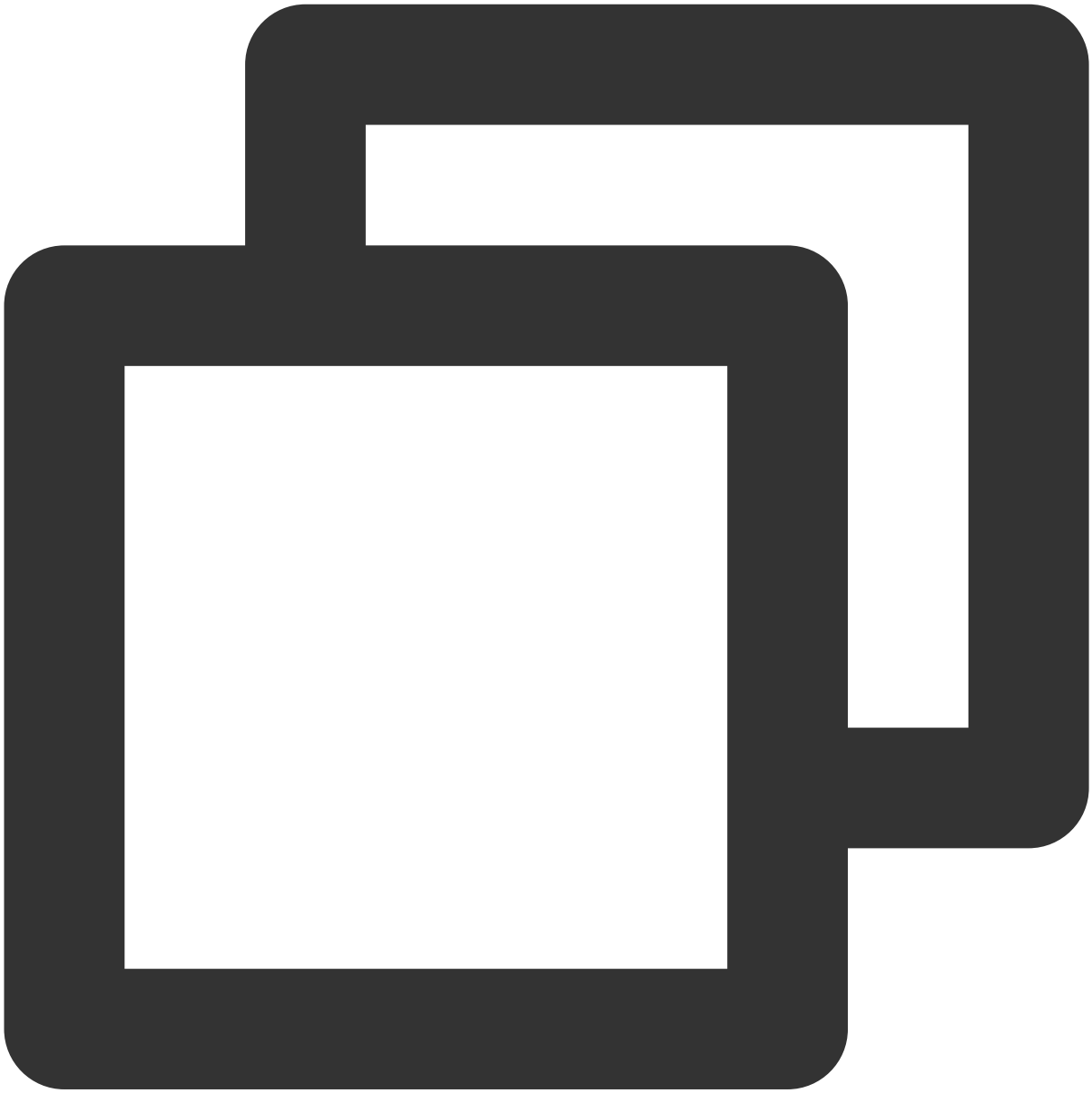
1.2 Add the following configuration under buildscript -> dependencies in your project-level build.gradle file:

For Gradle version 7.1 and above

Gradle version 7.0

Versions Below Gradle 7.0

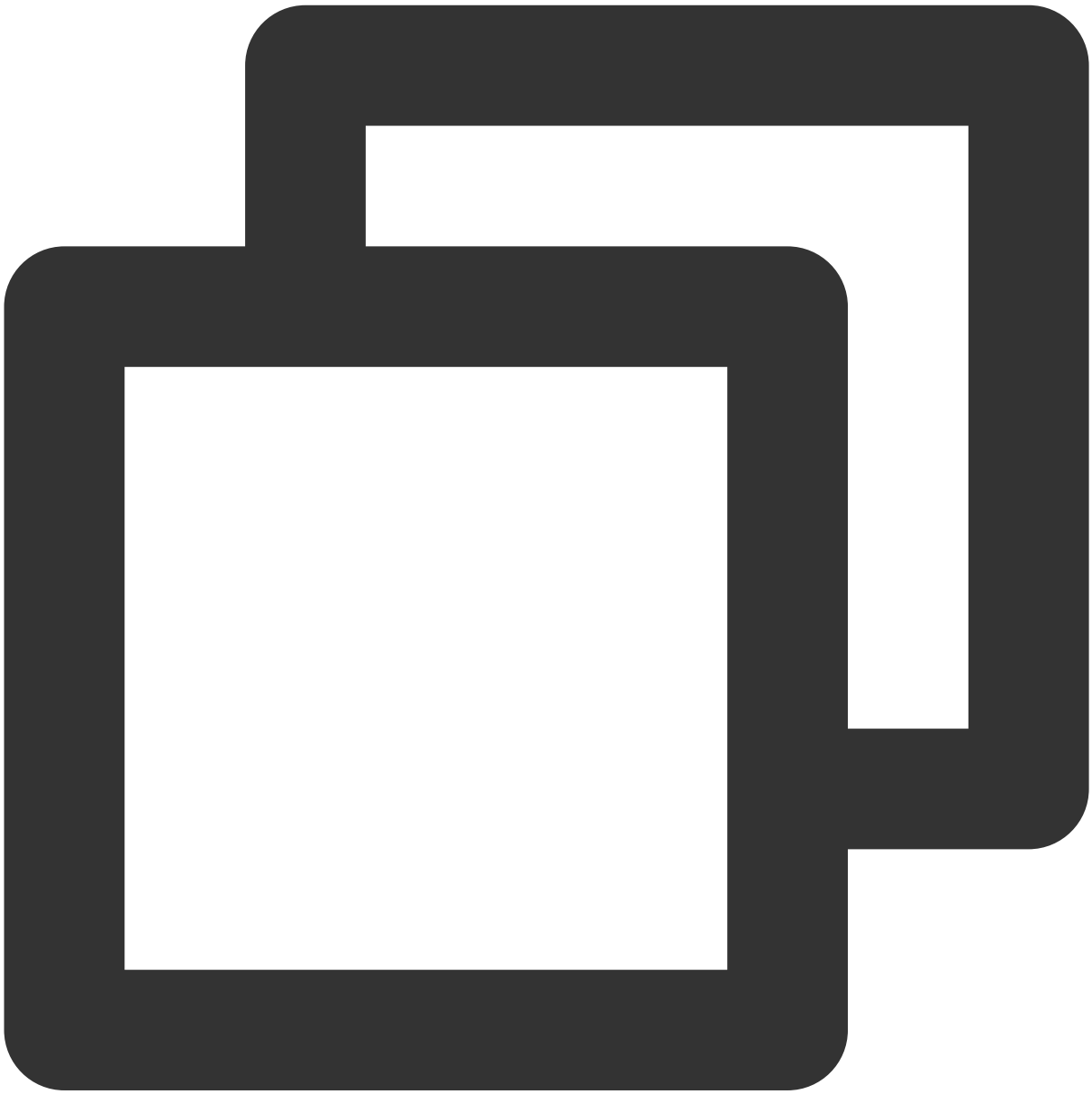
Add the following configuration under buildscript -> dependencies in your project-level build.gradle file:



```
buildscript {
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
```

```
    }
}
```

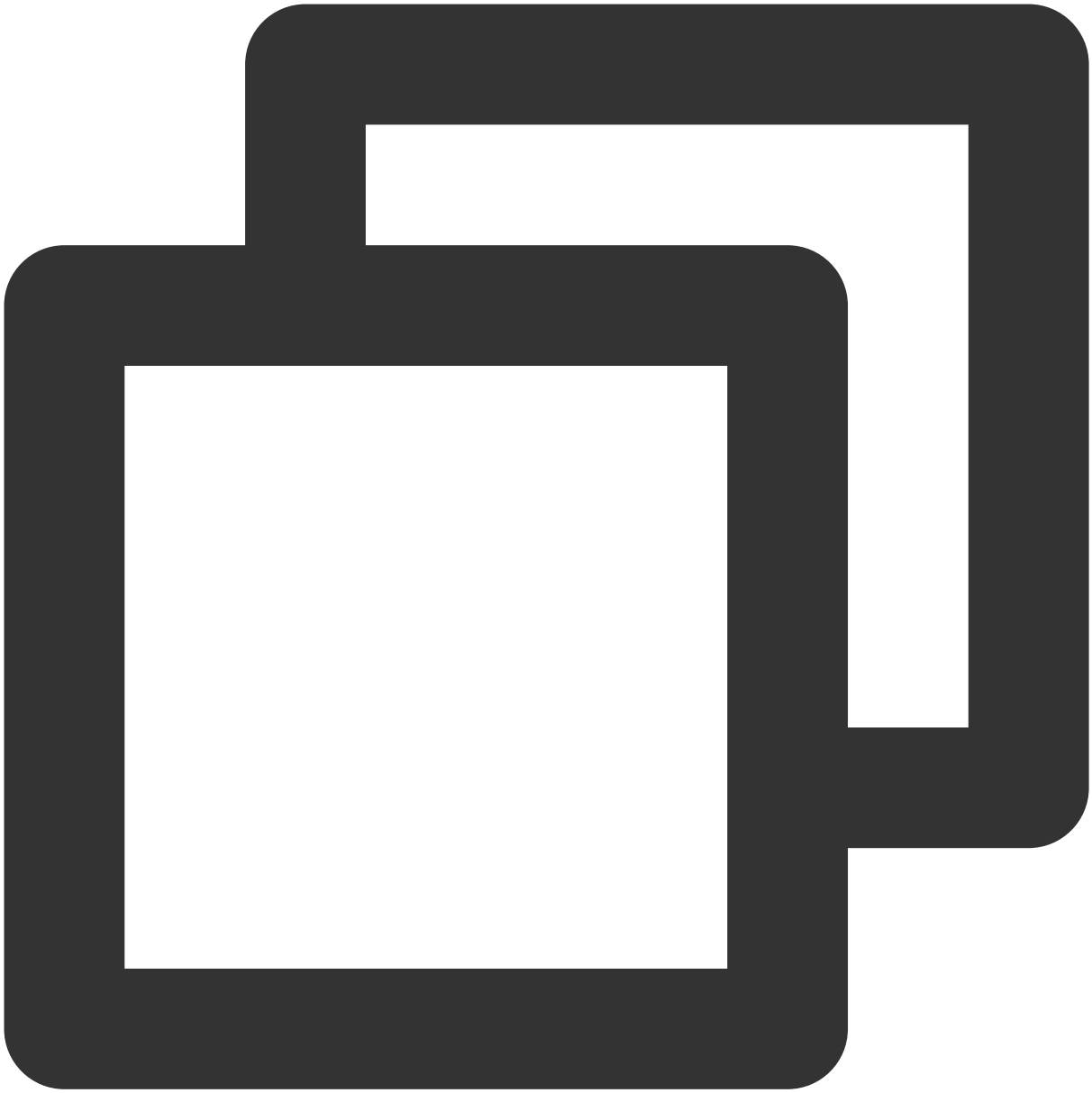
In the project-level settings.gradle file, add the following repository configurations under pluginManagement -> repositories and dependencyResolutionManagement -> repositories:



```
pluginManagement {
    repositories {
        gradlePluginPortal()
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
```

```
        // Configure the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
    }
}
dependencyResolutionManagement {
    ...
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configure the Maven repository address for HMS Core SDK.
       maven {url 'https://developer.huawei.com/repo/'}
       maven {url 'https://developer.hihonor.com/repo'}
    }
    }
}
```

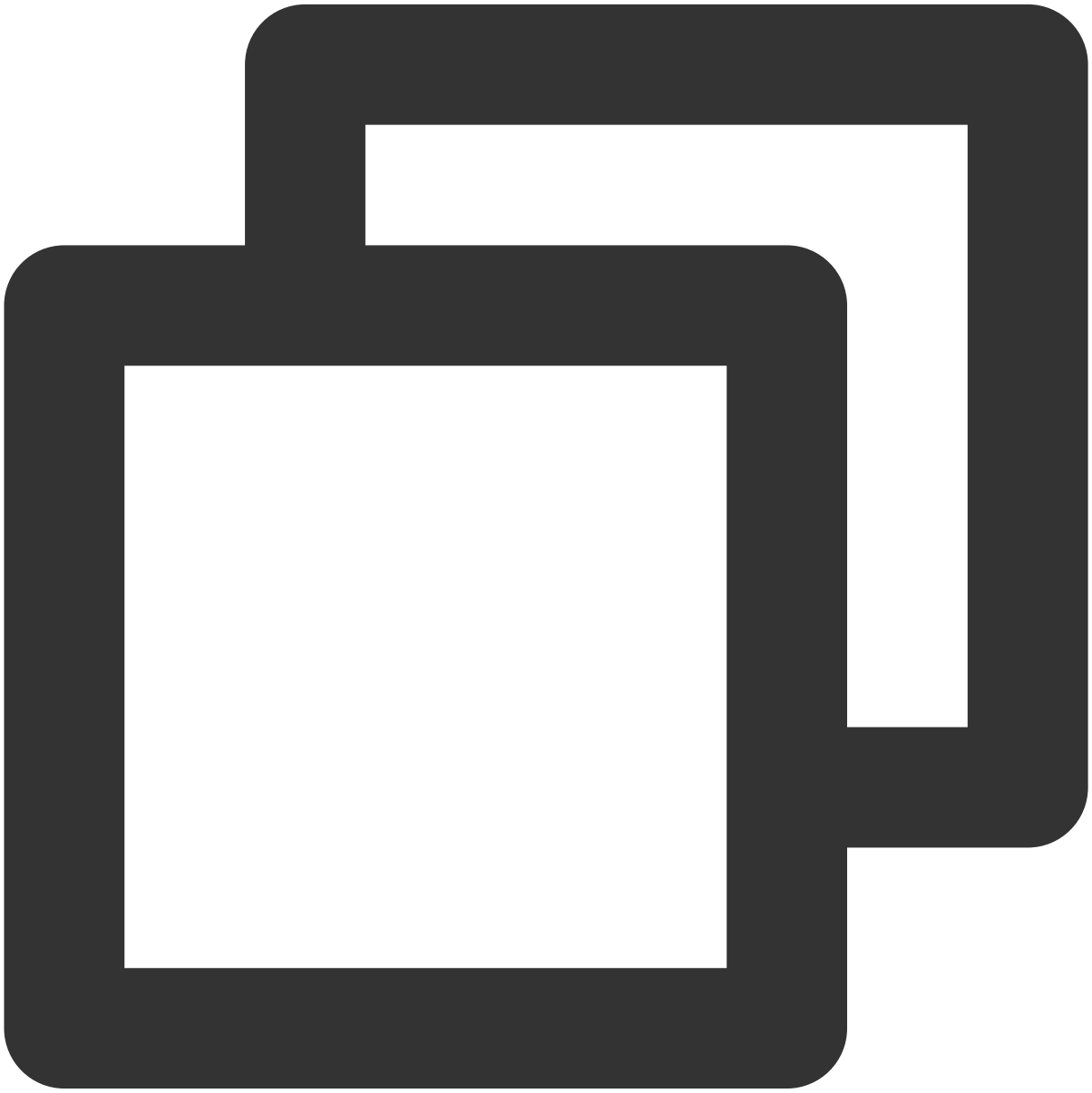
Add the following configuration under buildscript in your project-level build.gradle file:



```
buildscript {
    repositories {
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
```

```
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

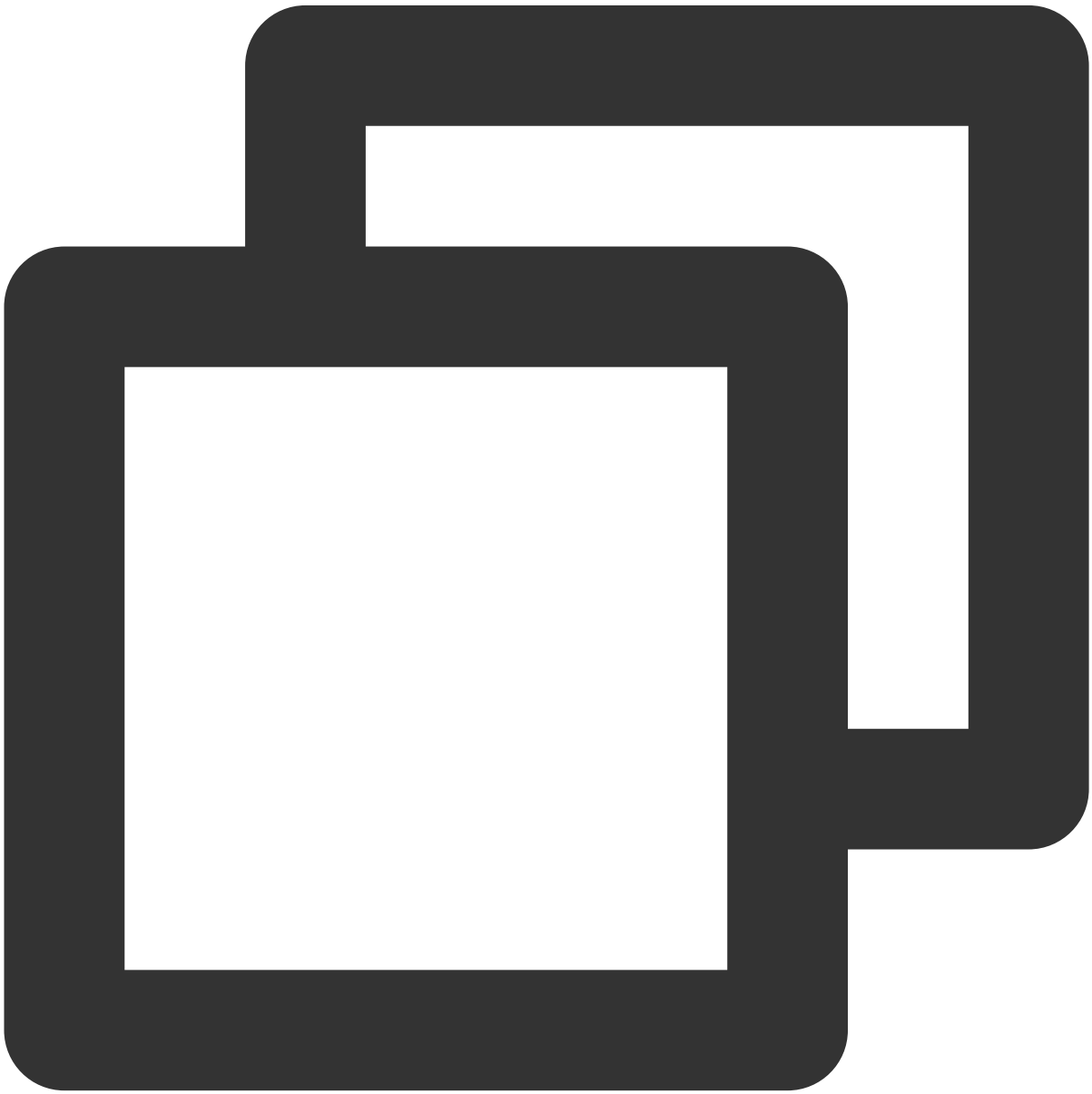
Add the following repository configurations under dependencyResolutionManagement -> repositories in your project-level settings.gradle file:



```
dependencyResolutionManagement {
    ...
    repositories {
```

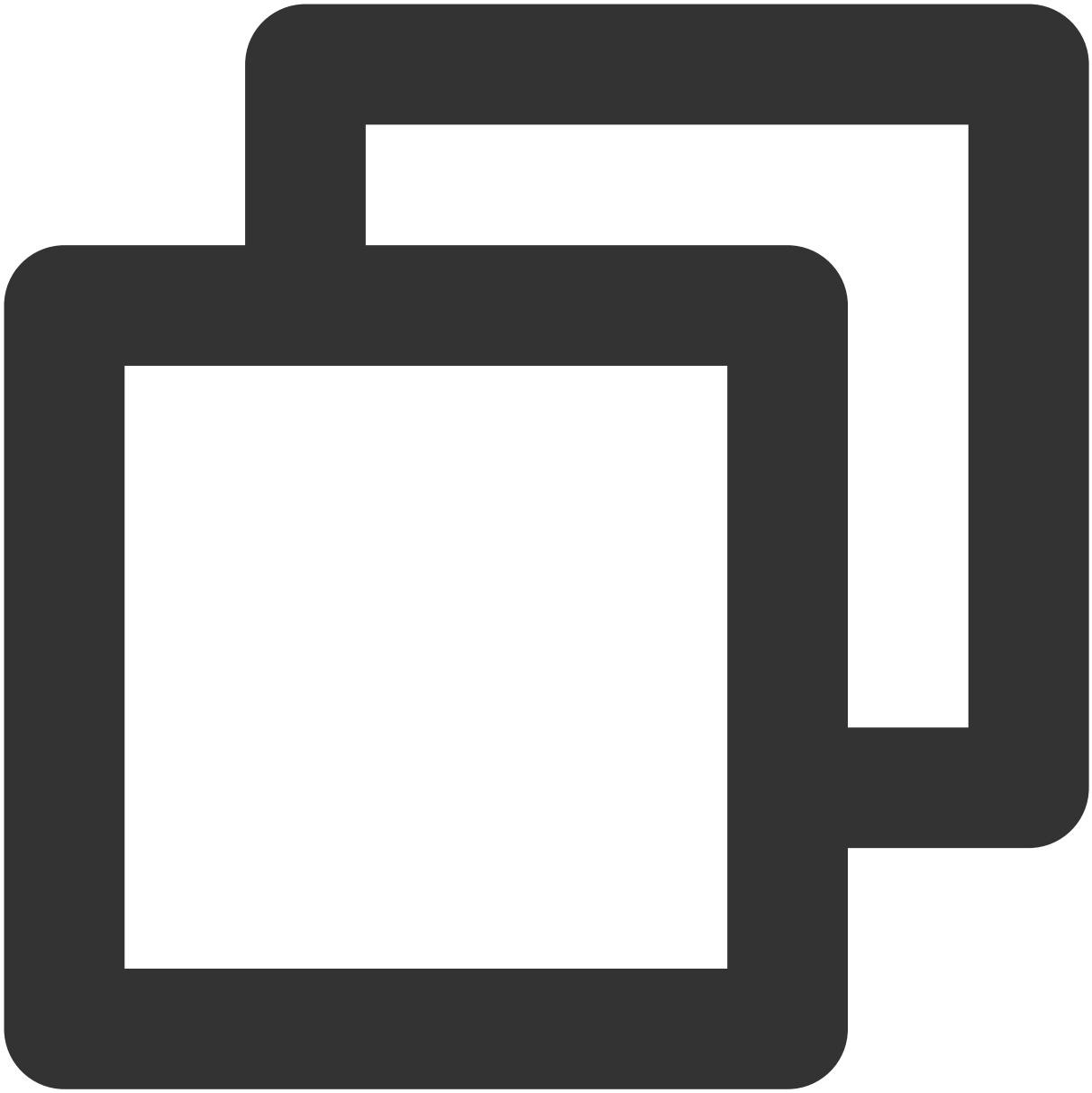
```
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

Add the following configuration under buildscript and allprojects in the project-level build.gradle file:



```
buildscript {
    repositories {
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}

allprojects {
    repositories {
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

1.3 Add the following configuration in the app-level build.gradle file:



```
apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.huawei.agconnect'
apply plugin: 'com.hihonor.mcs.asplugin'
```

After the above steps are performed, offline push notifications can be received.

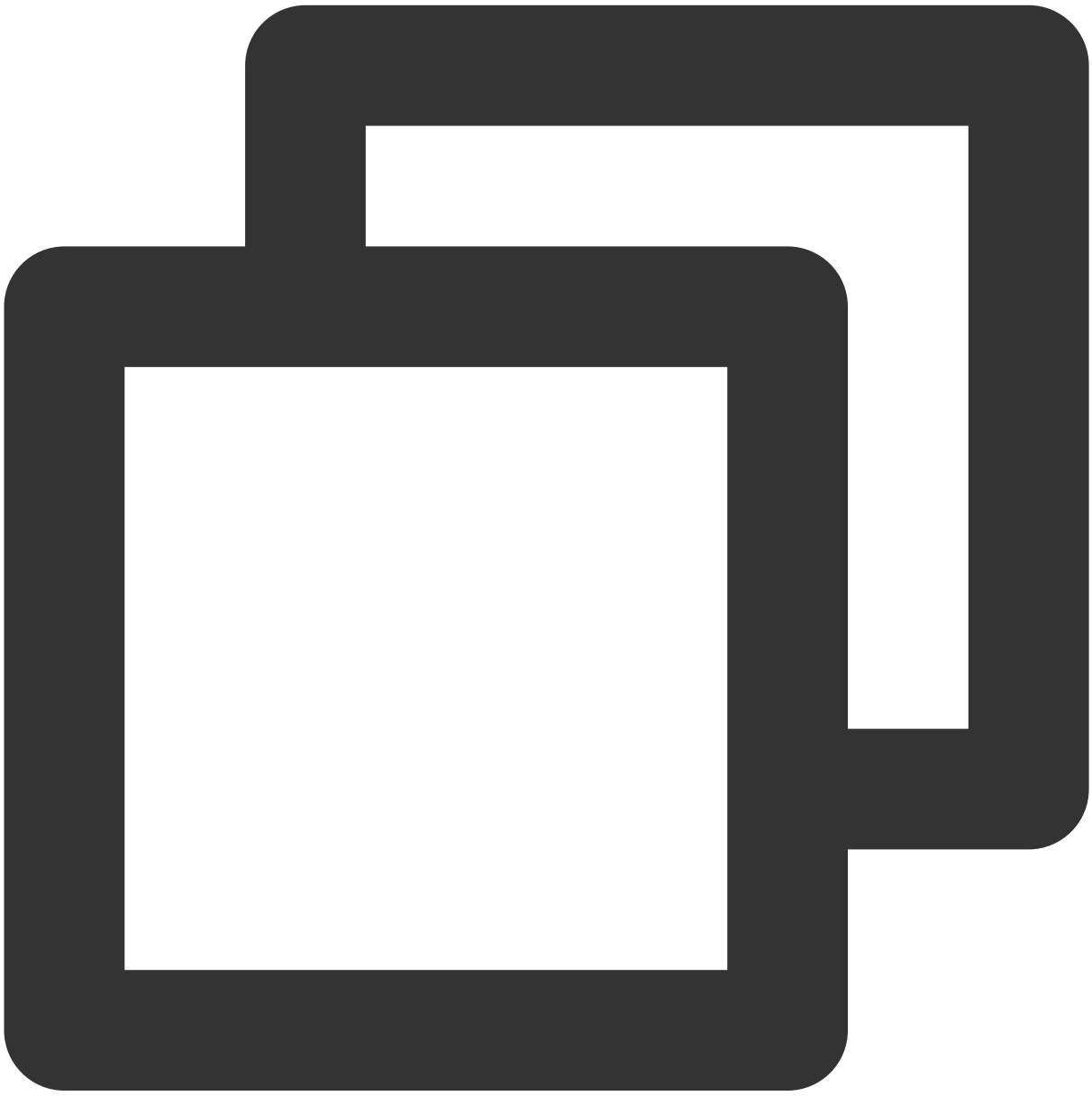
Note:

If you want to integrate the TIMPush component as simply as possible, you need to log in and log out using the login/logout APIs provided by TUILogin of the TUICore component, and the TIMPush component automatically

senses the log in and log out events. If you don't want to use the APIs provided by TUILogin, after completing the login/logout operation, you need to manually call the registerPush/unRegisterPush APIs of TIMPushManager.

Step 3: Set obfuscation rules

In the proguard-rules.pro file, add TIMPush-related classes to the non-obfuscation list:



```
-keep class com.tencent.qcloud.** { *; }
-keep class com.tencent.timpush.** { *; }
```

Step 4: Message Delivery Statistics Configuration

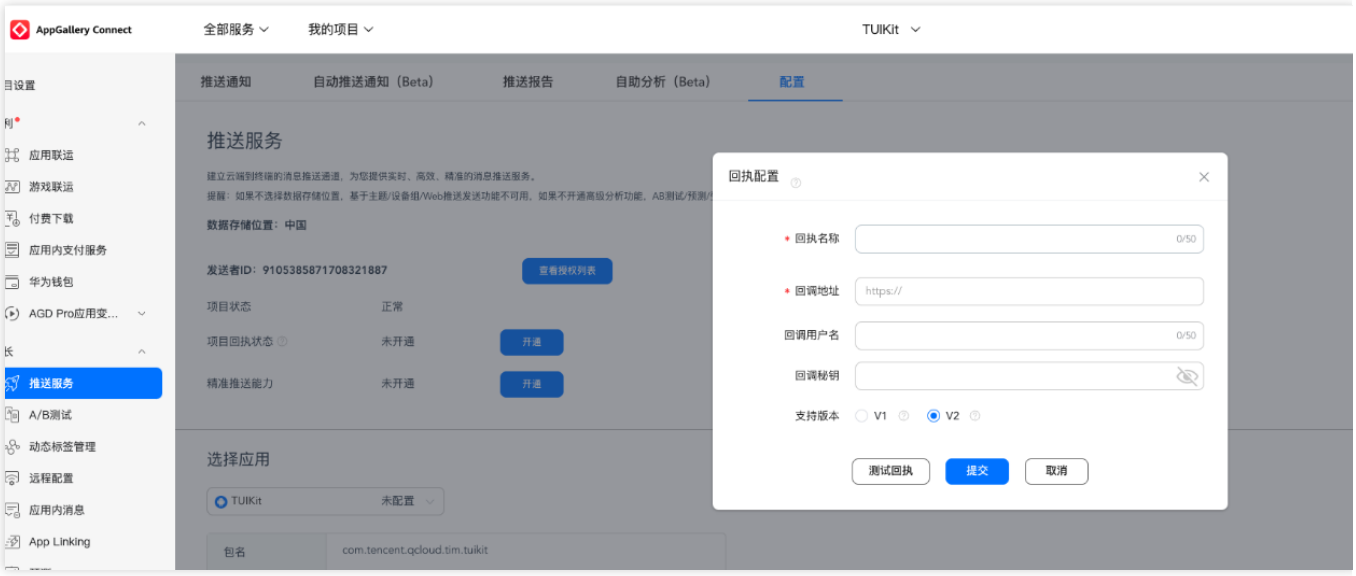
If you need to collect data reach statistics, please complete the configuration as follows:

Huawei

HONOR

vivo

Meizu

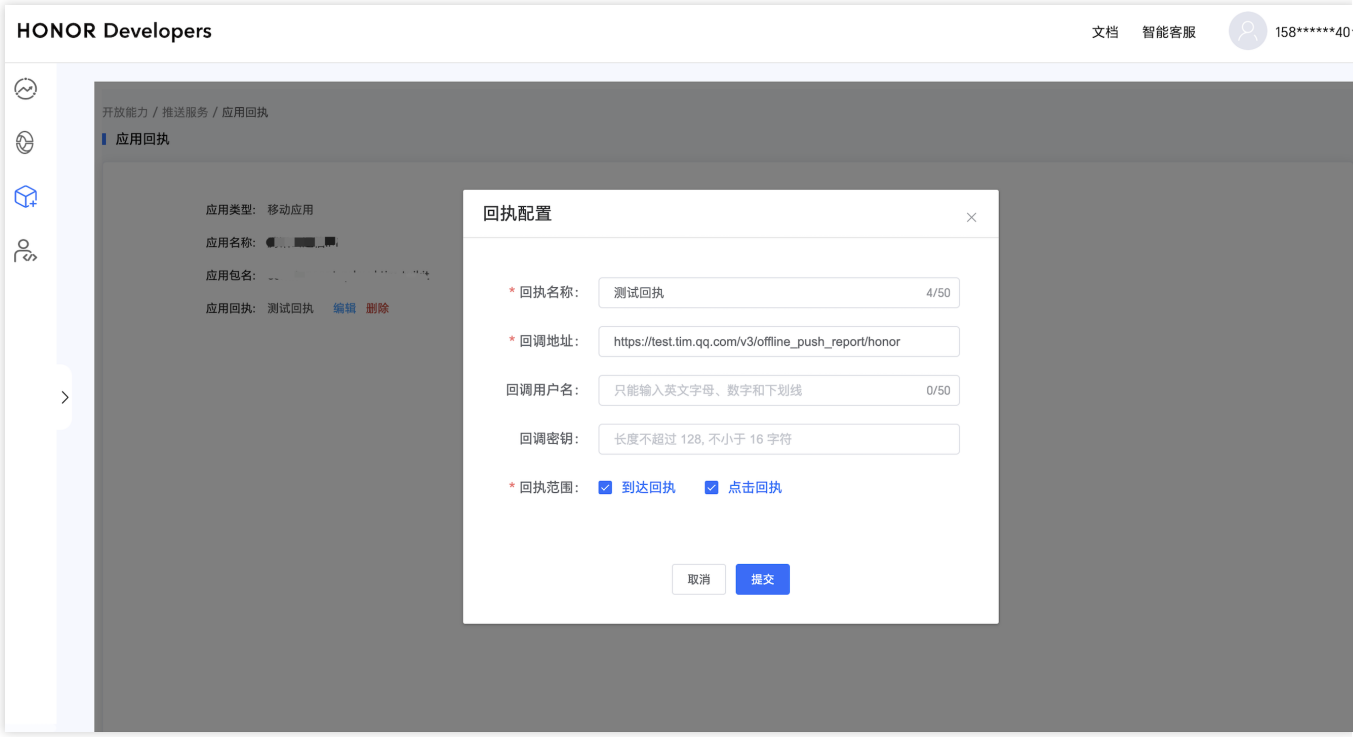


Receipt Address:

```
Singapore       https://apisgp.im.qcloud.com/v3/offline_push_report/huawei
South Korea     https://apikr.im.qcloud.com/v3/offline_push_report/huawei
USA     https://apiusa.im.qcloud.com/v3/offline_push_report/huawei
Germany         https://apiger.im.qcloud.com/v3/offline_push_report/huawei
Indonesia       https://apiidn.im.qcloud.com/v3/offline_push_report/huawei
Others in China      https://api.im.qcloud.com/v3/offline_push_report/huawei
```

Note:

Huawei Push Certificate ID <= 11344, using Huawei Push v2 interface, does not support Reach and Click Receipt, please regenerate and update Certificate ID.



Receipt Address:

Singapore	https://apisgp.im.qcloud.com/v3/offline_push_report/honor
South Korea	https://apikr.im.qcloud.com/v3/offline_push_report/honor
USA	https://apiusa.im.qcloud.com/v3/offline_push_report/honor
Germany	https://apiger.im.qcloud.com/v3/offline_push_report/honor
Indonesia	https://apiidn.im.qcloud.com/v3/offline_push_report/honor
Others in China	https://api.im.qcloud.com/v3/offline_push_report/honor

Callback Address Configuration	Configure Receipt ID in the
	

Receipt Address:

Singapore	https://apisgp.im.qcloud.com/v3/offline_push_report/vivo
South Korea	https://apikr.im.qcloud.com/v3/offline_push_report/vivo
USA	https://apiusa.im.qcloud.com/v3/offline_push_report/vivo
Germany	https://apiger.im.qcloud.com/v3/offline_push_report/vivo
Indonesia	https://apiidn.im.qcloud.com/v3/offline_push_report/vivo
Others in China	https://api.im.qcloud.com/v3/offline_push_report/vivo

Enable Receipt Switch	Configure Receipt Addres
	

Receipt Address:

Singapore	https://apisgp.im.qcloud.com/v3/offline_push_report/meizu
South Korea	https://apikr.im.qcloud.com/v3/offline_push_report/meizu
USA	https://apiusa.im.qcloud.com/v3/offline_push_report/meizu
Germany	https://apiger.im.qcloud.com/v3/offline_push_report/meizu
Indonesia	https://apiidn.im.qcloud.com/v3/offline_push_report/meizu
China	https://api.im.qcloud.com/v3/offline_push_report/meizu

Note:

After turning on the receipt toggle, please make sure the receipt address is correctly configured. Not configuring it or configuring the wrong address will affect the push notification feature.

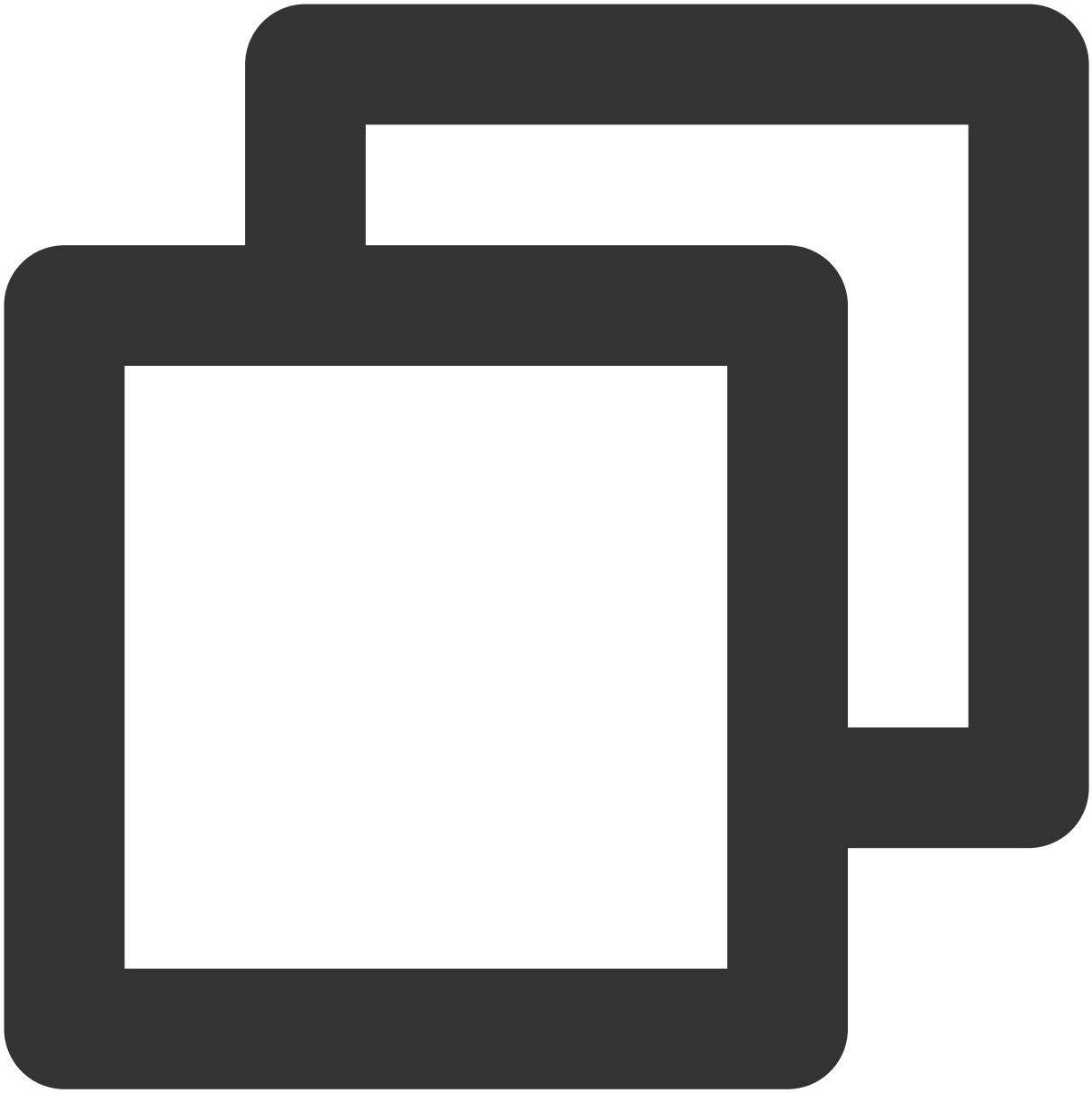
Note:

No configuration for message delivery statistics is required for other supported manufacturers.

FCM currently does not support the push notification statistics feature.

Step 5: Set the offline push parameters when sending a message

When calling sendMessage to send a message, you can set offline push parameters through V2TIMOfflinePushInfo. By calling V2TIMOfflinePushInfo's setExt, you can set custom Definition ext data, which allows you to access the ext field in the callback of the click notification jump when the user starts the App after receiving an offline push. Then, based on the content of the ext field, navigate to the specified UI interface. See the sendMessage() method of ChatProvider for reference:

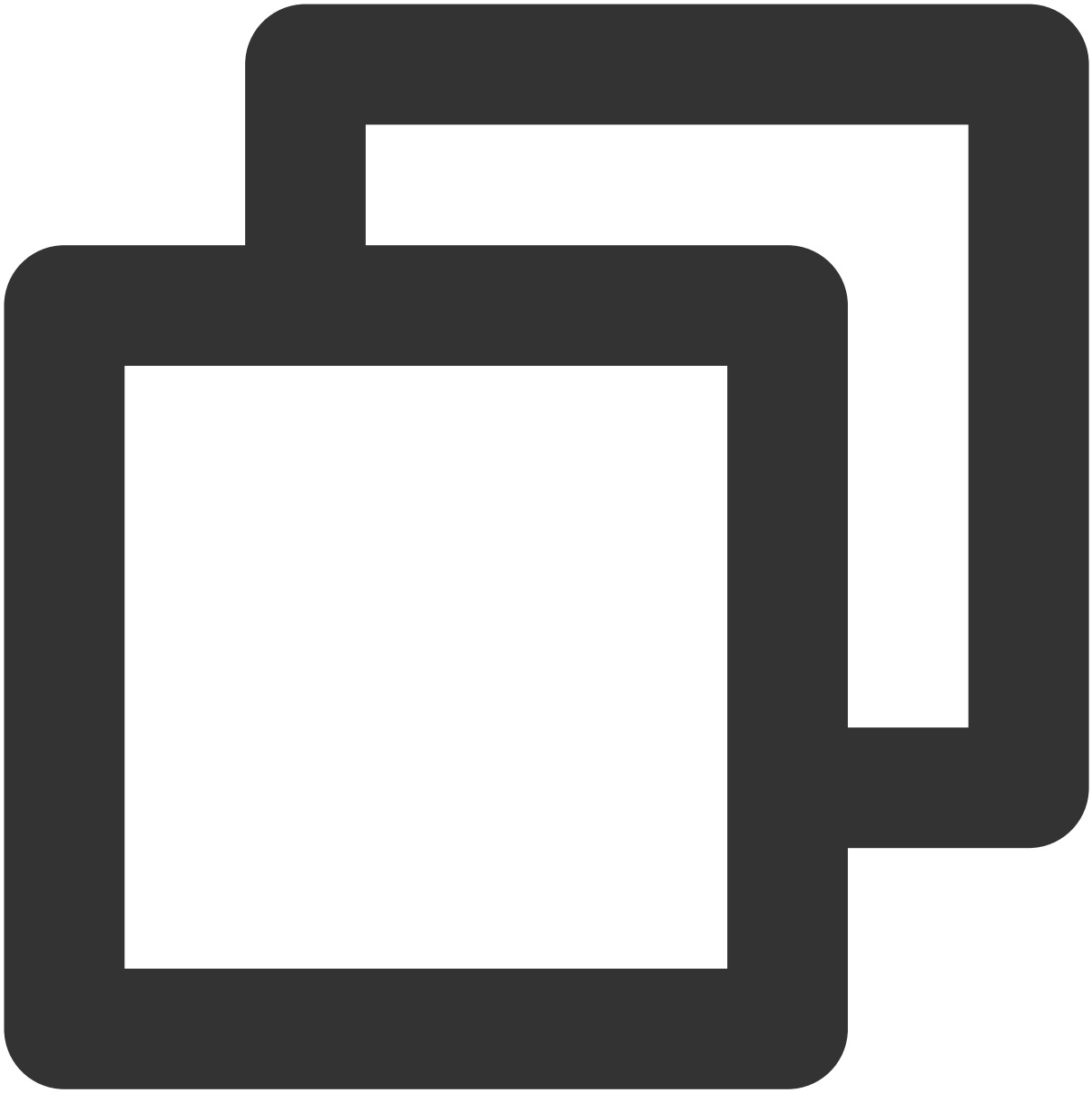


```
V2TIMOfflinePushInfo v2TIMOfflinePushInfo = new V2TIMOfflinePushInfo();
v2TIMOfflinePushInfo.setTitle("Push Title");
v2TIMOfflinePushInfo.setDesc("Push Content");

OfflinePushExtInfo offlinePushExtInfo = new OfflinePushExtInfo();
offlinePushExtInfo.getBusinessInfo().setSenderId("senderID");
offlinePushExtInfo.getBusinessInfo().setSenderNickName("senderNickName");
if (chatInfo.getType() == V2TIMConversation.V2TIM_GROUP) {
    offlinePushExtInfo.getBusinessInfo().setChatType(V2TIMConversation.V2TIM_GROUP)
    offlinePushExtInfo.getBusinessInfo().setSenderId("groupID");
```

```
}
v2TIMOfflinePushInfo.setExt(new Gson().toJson(offlinePushExtInfo).getBytes());

// For OPPO, you must set the `ChannelID` to receive push messages. The `ChannelID`
v2TIMOfflinePushInfo.setAndroidOPPOChannelID("tuikit");
v2TIMOfflinePushInfo.setAndroidHuaWeiCategory("IM");
v2TIMOfflinePushInfo.setAndroidVIVOCategory("IM");

final V2TIMMessage v2TIMMessage = message.getTimMessage();
String msgID = V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, isGroup ?
    V2TIMMessage.V2TIM_PRIORITY_DEFAULT, false, v2TIMOfflinePushInfo, new V2TIMSend
        @Override
        public void onProgress(int progress) {

        }

        @Override
        public void onError(int code, String desc) {
            TUIChatUtils.callbackOnError(callBack, TAG, code, desc);
        }

        @Override
        public void onSuccess(V2TIMMessage v2TIMMessage) {
            TUIChatLog.v(TAG, "sendMessage onSuccess:" + v2TIMMessage.getMsgID());
            message.setMsgTime(v2TIMMessage.getTimestamp());
            TUIChatUtils.callbackOnSuccess(callBack, message);
        }
    });
```

Step 6: Parsing Offline Push Messages

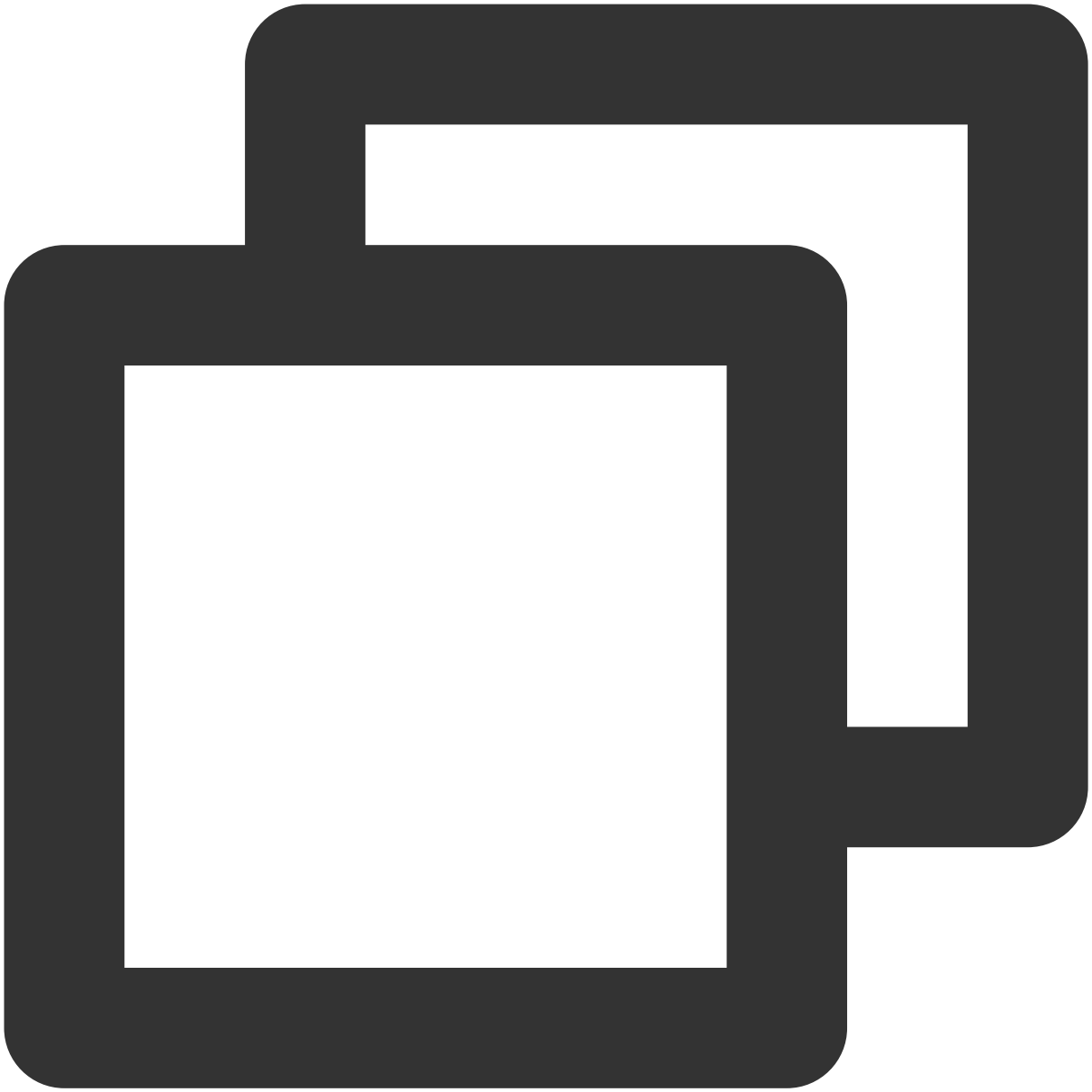
When a push is received, by clicking on the notification bar event, the component will notify the application in the form of a callback or broadcast. The application can then configure the App's redirect page in the callback. It is recommended to register the callback during the application's oncreate() function in the Application.
The callback method is as follows:



```
TUICore.registerEvent(TUIConstants.TIMPush.EVENT_NOTIFY, TUIConstants.TIMPush.EVENT
        @Override
        public void onNotifyEvent(String key, String subKey, Map<String, Object> pa
            Log.d(TAG, "onNotifyEvent key = " + key + "subKey = " + subKey);
            if (TUIConstants.TIMPush.EVENT_NOTIFY.equals(key)) {
                if (TUIConstants.TIMPush.EVENT_NOTIFY_NOTIFICATION.equals(subKey))
                    if (param != null) {
                        String extString = (String)param.get(TUIConstants.TIMPush.N
                        // Getting ext for Definition redirect

                        // Example: Redirect to the corresponding chat interface
```

```
                        OfflinePushExtInfo offlinePushExtInfo = null;
                        try {
                            offlinePushExtInfo = new Gson().fromJson(extString, Off
                            if (offlinePushExtInfo.getBusinessInfo().getChatAction(
                                String senderId = offlinePushExtInfo.getBusinessInf
                                if (TextUtils.isEmpty(senderId)) {
                                    return;
                                }
                                TUIUtils.startChat(senderId, offlinePushExtInfo.get
                            }
                        } catch (Exception e) {
                            Log.e(TAG, "getOfflinePushExtInfo e: " + e);
                        }
                    }
                }
            }
        }
    });
```

The broadcast method is as follows:



```
// Dynamic Broadcast Registration
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(TUIConstants.TIMPush.NOTIFICATION_BROADCAST_ACTION);
LocalBroadcastManager.getInstance(context).registerReceiver(localReceiver, intentFi

// Broadcast Receiver
public class OfflinePushLocalReceiver extends BroadcastReceiver {
    public static final String TAG = OfflinePushLocalReceiver.class.getSimpleName()

    @Override
    public void onReceive(Context context, Intent intent) {
```

```
        DemoLog.d(TAG, "BROADCAST_PUSH_RECEIVER intent = " + intent);
        if (intent != null) {
            String ext = intent.getStringExtra(TUIConstants.TIMPush.NOTIFICATION_EX
            // Getting ext for Custom Redirect

            // Example: Redirect to the corresponding chat interface
            OfflinePushExtInfo offlinePushExtInfo = null;
            try {
                offlinePushExtInfo = new Gson().fromJson(extString, OfflinePushExtI
                if (offlinePushExtInfo.getBusinessInfo().getChatAction() == Offline
                    String senderId = offlinePushExtInfo.getBusinessInfo().getSende
                    if (TextUtils.isEmpty(senderId)) {
                        return;
                    }
                    TUIUtils.startChat(senderId, offlinePushExtInfo.getBusinessInfo
                }
             } catch (Exception e) {
                 Log.e(TAG, "getOfflinePushExtInfo e: " + e);
             }
        } else {
            Log.e(TAG, "onReceive ext is null");
        }
    }
}
```

Congratulations! You have completed the integration of the push plugin. Please be reminded: After the trial period or subscription expires, the push service (including regular message offline push, all-staff/Tag push, etc.) will automatically cease. To avoid affecting the normal use of your services, please make sure to purchase/renew in advance.

uniapp

Last updated：2024-06-13 10:21:45

Preconditions

1. To integrate Chat TUIKit, see UI Integration Solution (Recommended) -> Integrate Basic feature -> uni-app (Vue2/Vue3)

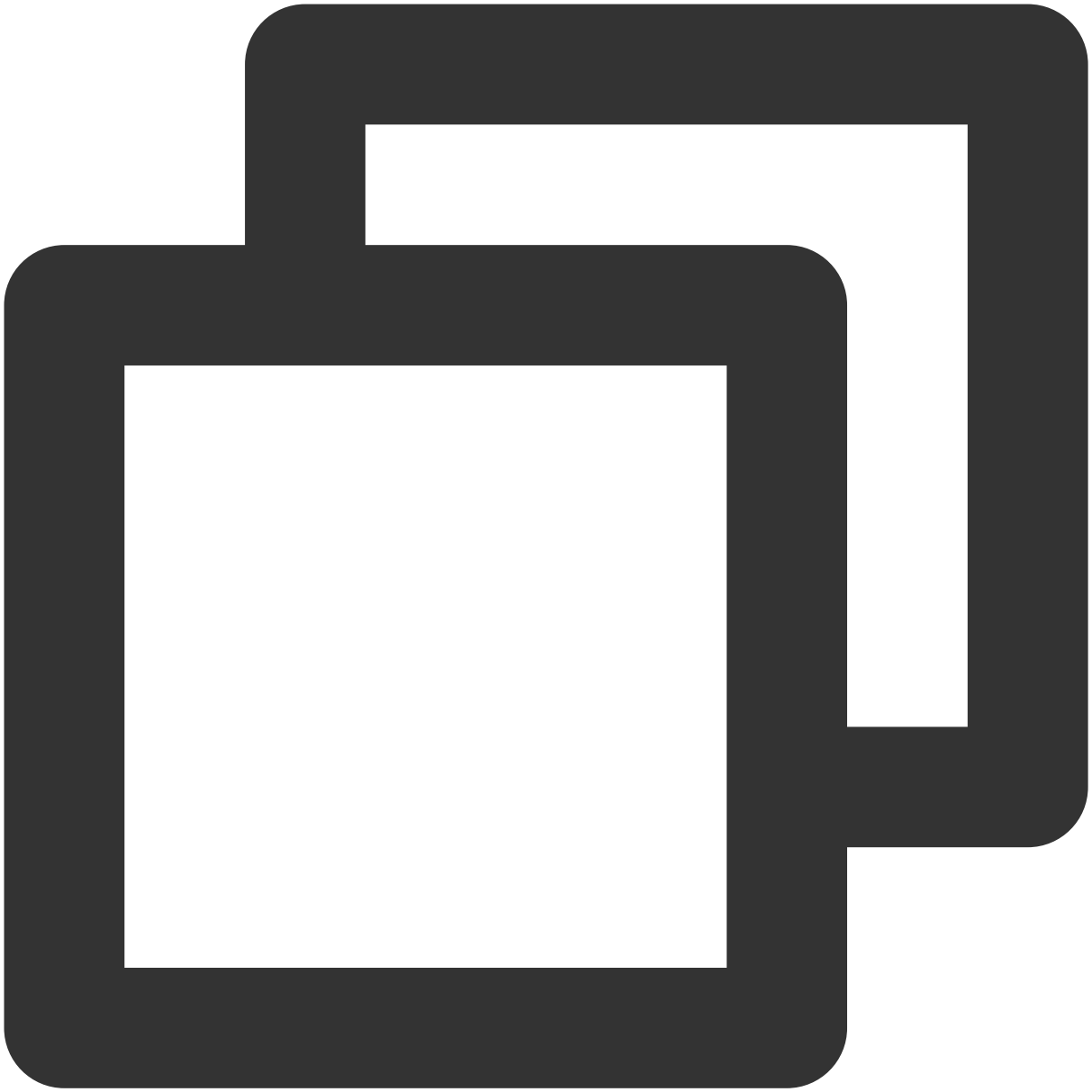
Note:

1. If you have already integrated Chat TUIKit, you can ignore this step.
2. If you are integrating Chat without UI, you can ignore this step.

2. Upgrade @tencentcloud/chat to the latest version.

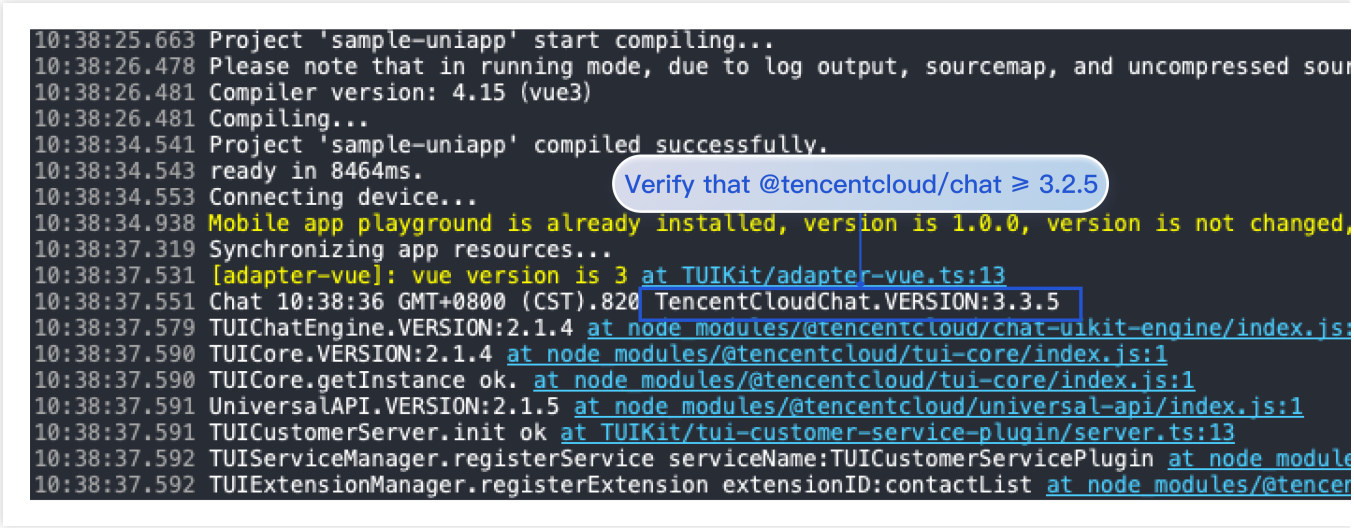
Note:

@tencentcloud/chat is backward compatible, feel free to upgrade. If you are currently using **tim-js-sdk** or **tim-wx-sdk**, please refer to our Upgrade Guide**.**



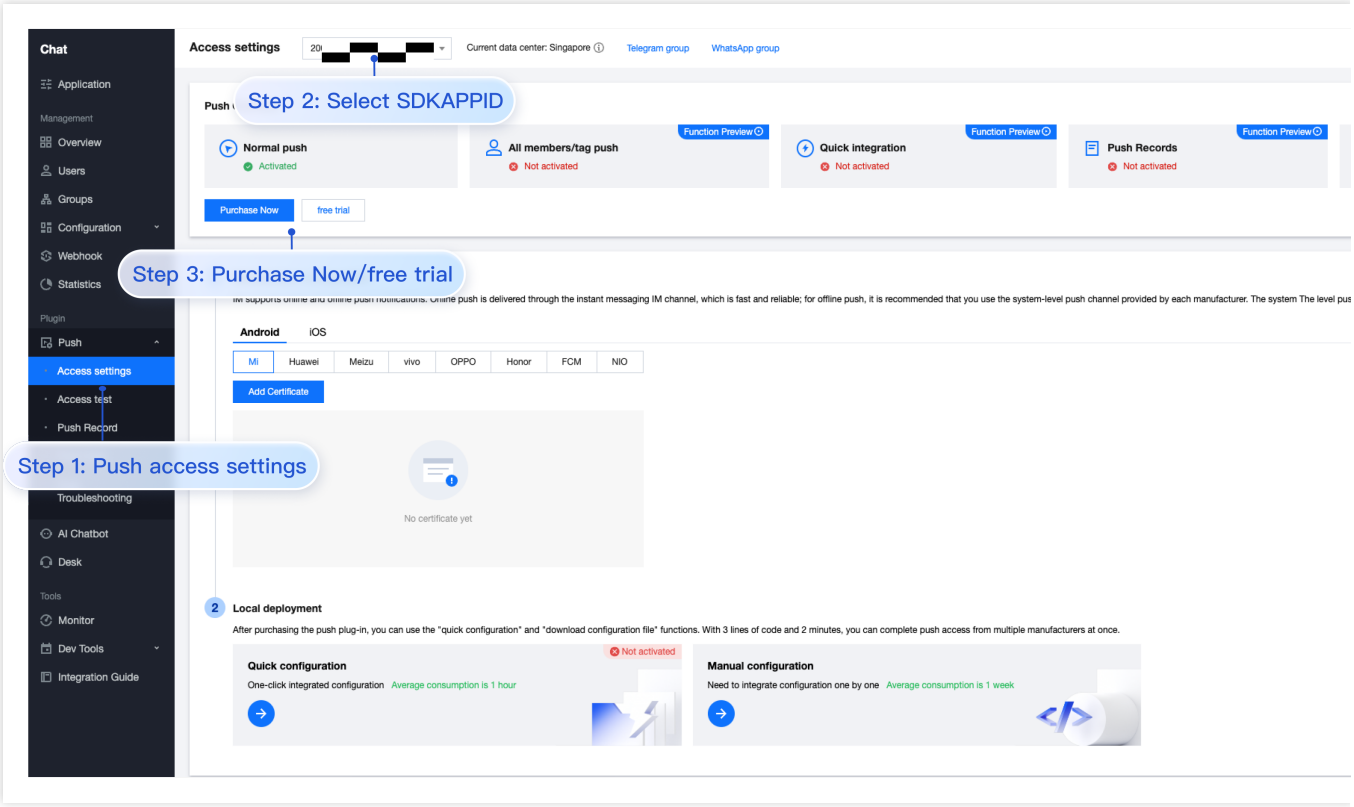
```
npm install @tencentcloud/chat@latest
```

Check the version number of TencentCloudChat.VERSION in the HBuilder log to confirm **@tencentcloud/chat ≥ 3.2.5** as shown:



3. Activate the Push Plugin

Go to IM Console > Push, click **Purchase Now** or **Free Trial**. (Each application can try it for free once, valid for 7 days.)



Note:

After the trial or purchase of the Push Plugin expires, the push service (including normal message offline push, all member/Tag push, etc.) will automatically stop. To avoid affecting the normal use of your business, please

purchase/renew in advance.

4. Manufacturer Configuration

Note:

The uniapp Manufacturer Configuration includes Android Manufacturer Configuration and iOS Manufacturer Configuration, see uniapp.

Integrating TencentCloud-TIMPush

Step 1: manifest.json App Module Configuration

In the project [**manifest.json**] > [**App Module Configuration**], configure the Message Push Module as shown:

Step 2: Activate the TencentCloud-TIMPush cloud packaging service and fill in the relevant parameters.

1. Go to the plugin marketplace and activate the TencentCloud-TIMPush cloud packaging service. As shown in the image:

Note:

1. In the plugin marketplace, the **appId** for the cloud packaging service project must match the **appid** in the project's **manifest.json**.
2. Activation of the **TencentCloud-TIMPush** cloud packaging service is for a single project only. It only concerns the project.

2. In the project's [**manifest.json**] > [**App Native Plugin Configuration**] > [**Cloud Plugins**], select **TencentCloud-TIMPush** and set the relevant parameters.

Note:

1. Note that parameters might appear out of order in HBuilderX. Please fill them out carefully and accurately.
2. Each parameter is required; otherwise, a compilation error will occur. The default is 0.

com.hihonor.push.app_id

com.vivo.push.app_id && com.vivo.push.api_key

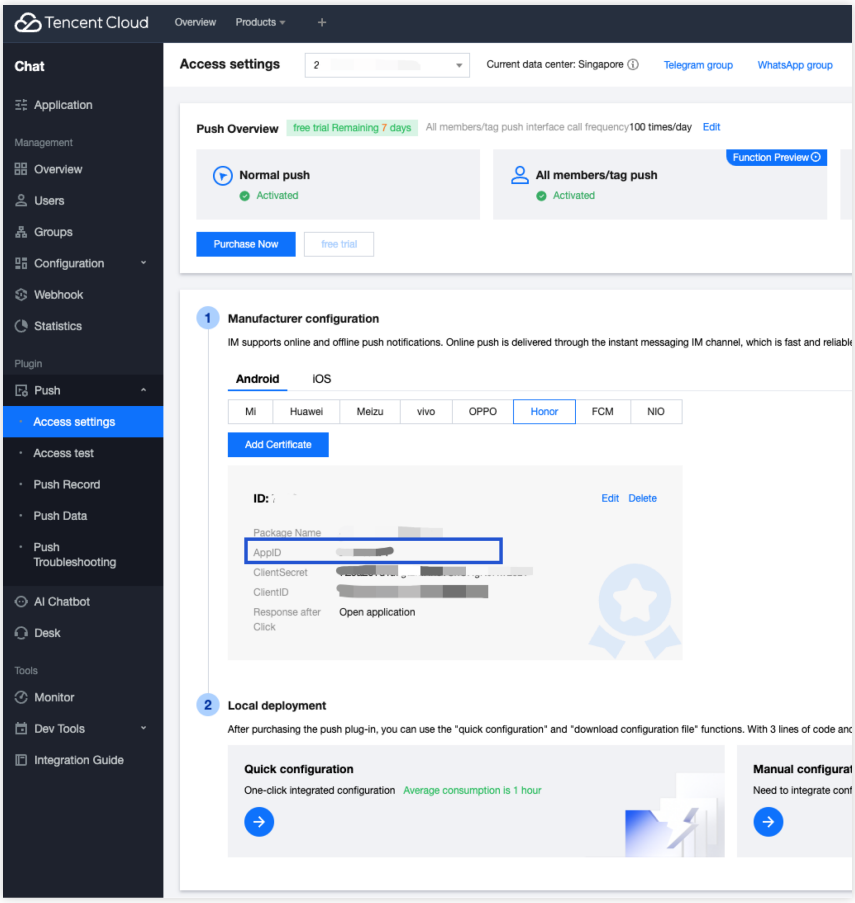
TIMPushAppGroupID

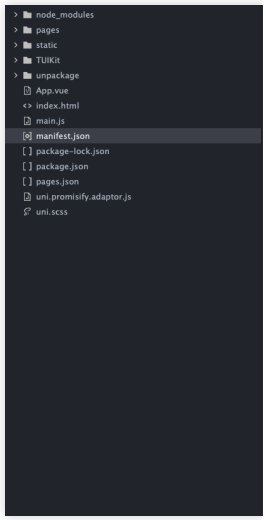
Note:

com.hihonor.push.app_id corresponds to the appID of hihonor.

When not enabling hihonor push, com.hihonor.push.app_id can be set to 0 by default.

Console configuration	manifest.json Honor Configu





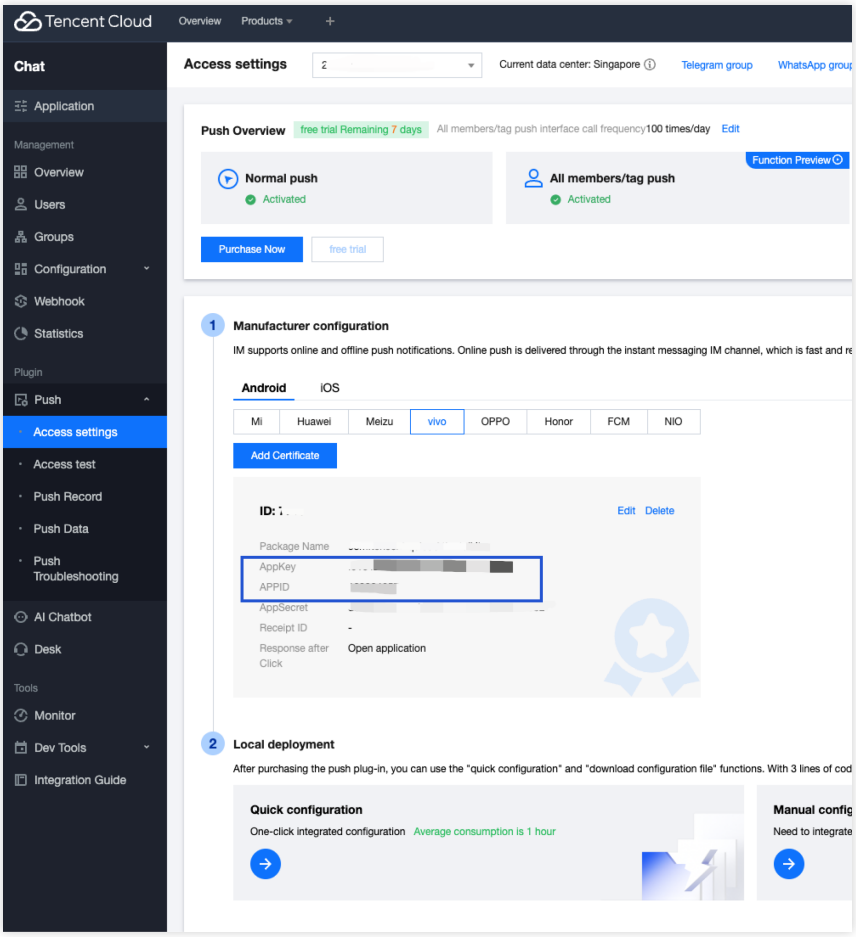
Note:

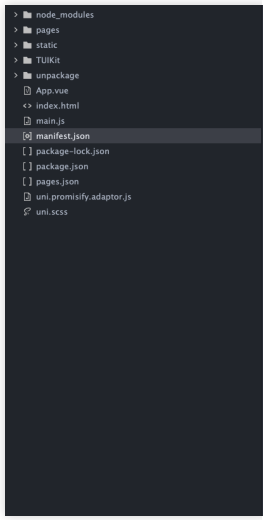
com.vivo.push.api_id corresponds to vivo's appID.

com.vivo.push.api_key corresponds to vivo's appKey.

When not enabling vivo push, com.vivo.push.api_id and com.vivo.push.api_key can be set to 0 by default.

Console configuration	manifest.json vivo Configura





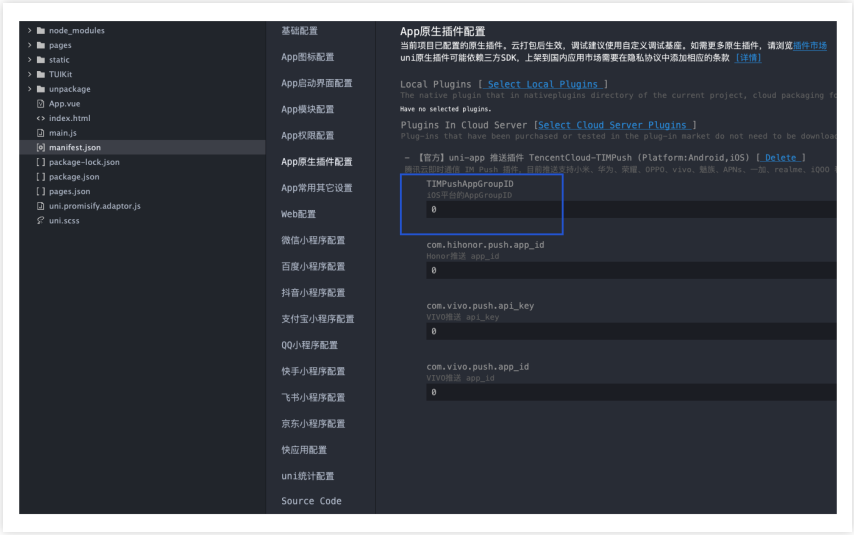
TIMPushAppGroupID corresponds to iOS's appGroupID. It is a configuration item for iOS reach reporting, refer to configuring-app-groups to generate appGroupID.

Note:

Not configuring TIMPushAppGroupID will not affect the normal push feature.

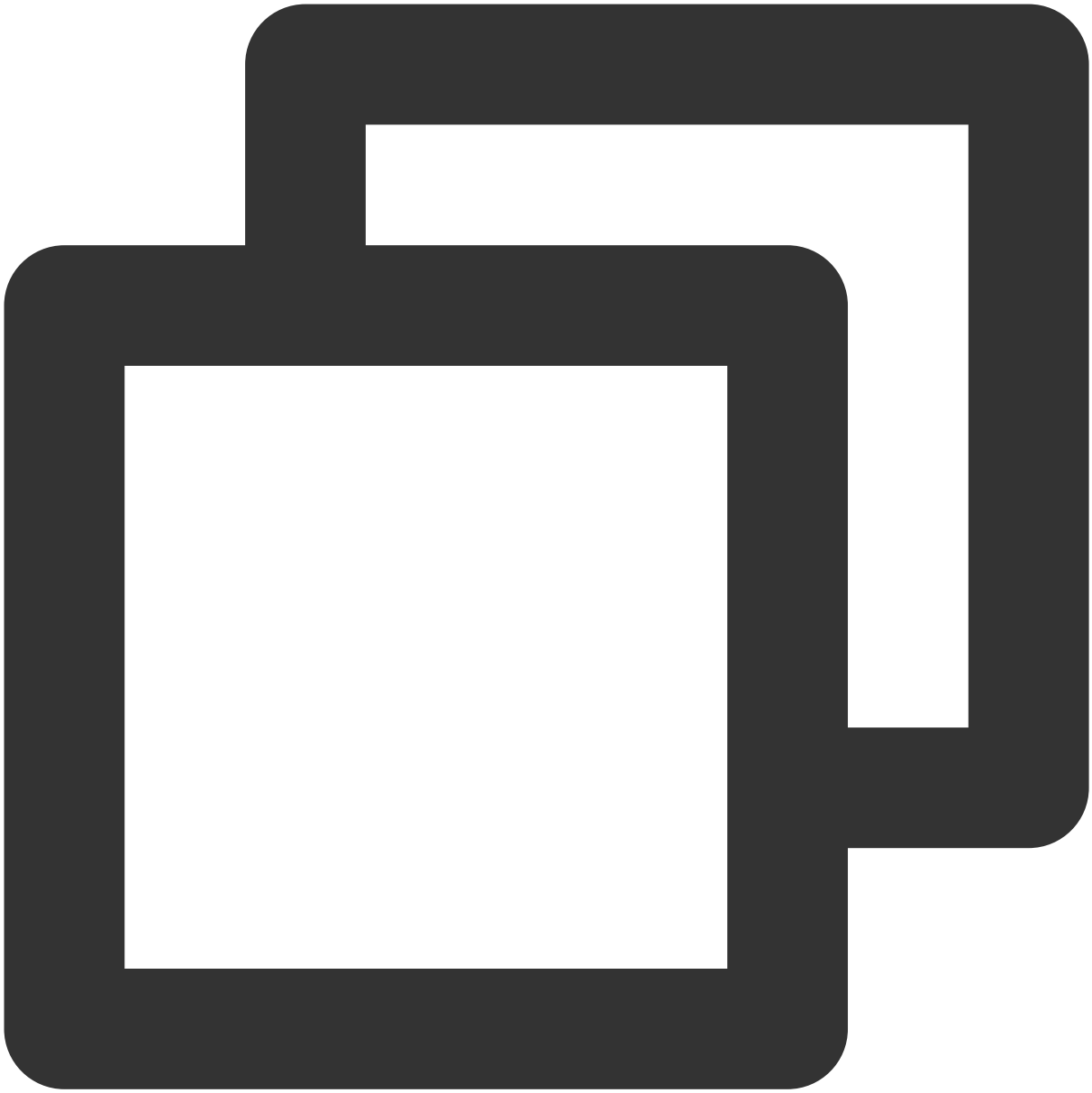
When iOS reach reporting is not enabled, TIMPushAppGroupID can be set to `0` by default;

iOS appGroupID Generation Guidelines	manifest.json iOS Configuration
You can refer to configuring-app-groups to generate appGroupID	

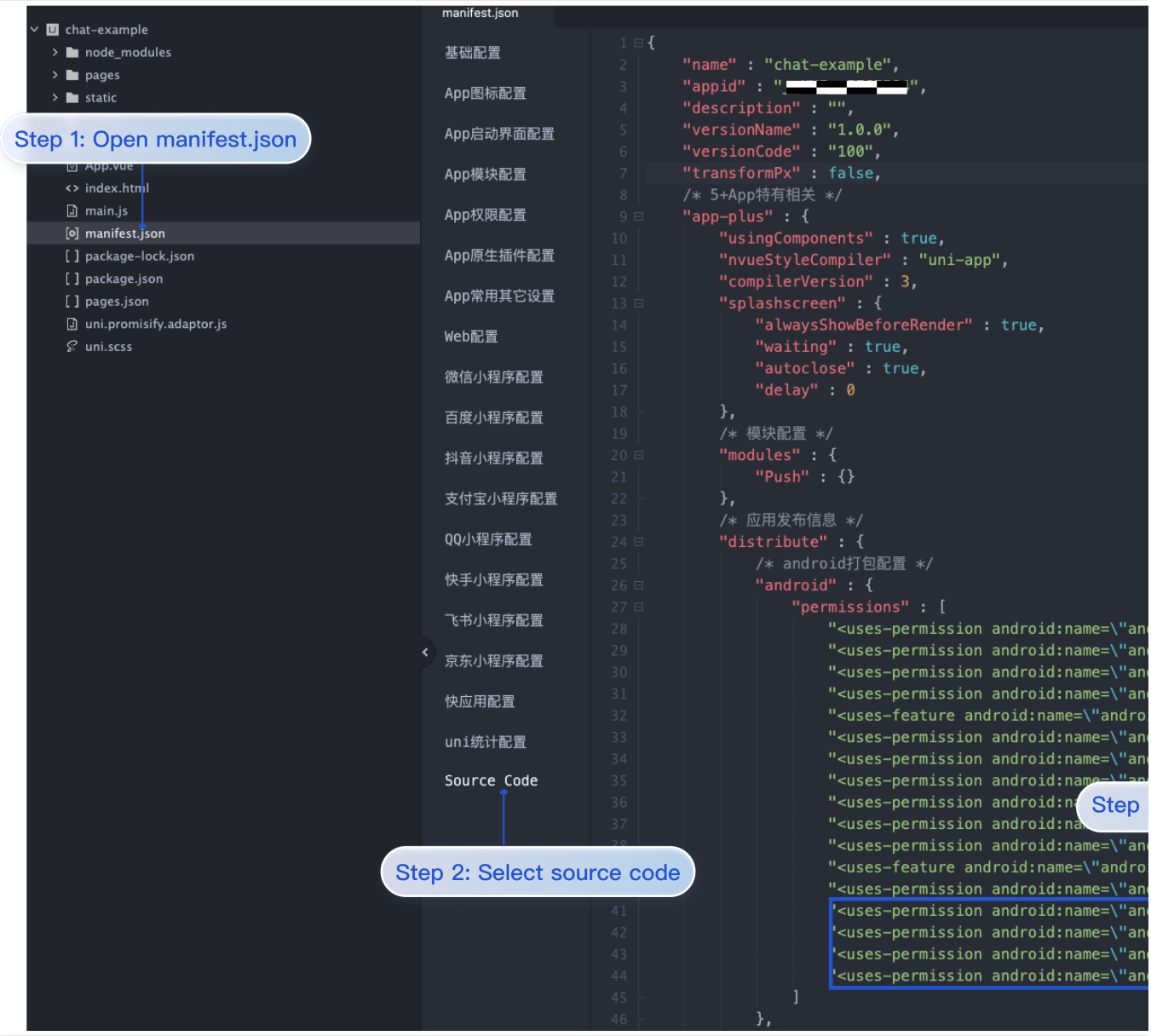


Step 3: manifest.json Android Permission Configuration

Append the following permissions in **manifest.json > Source View > app-plus > distribute > android > permissions**, as shown:



```
"<uses-permission android:name=\\"android.permission.INTERNET\\" />",
"<uses-permission android:name=\\"android.permission.ACCESS_NETWORK_STATE\\" />",
"<uses-permission android:name=\\"android.permission.ACCESS_WIFI_STATE\\" />",
"<uses-permission android:name=\\"android.permission.WRITE_EXTERNAL_STORAGE\\" />"
```



Step 4. Register TencentCloud-TIMPush

Note:

@tencentcloud/chat ≥ 3.2.5 supports TencentCloud-TIMPush.

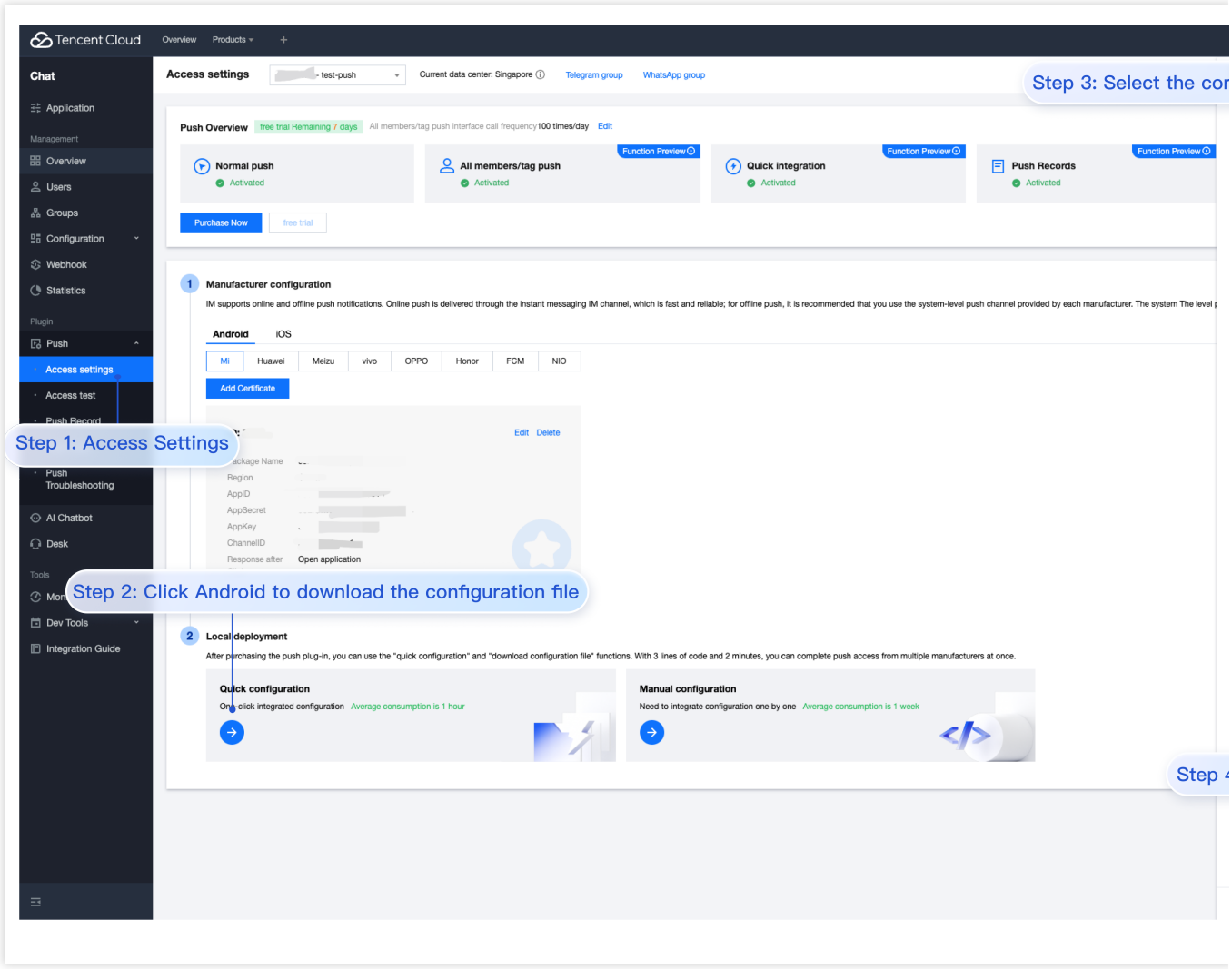
androidConfig is the Android push configuration. If you don't need to package an Android App, you can pass null.

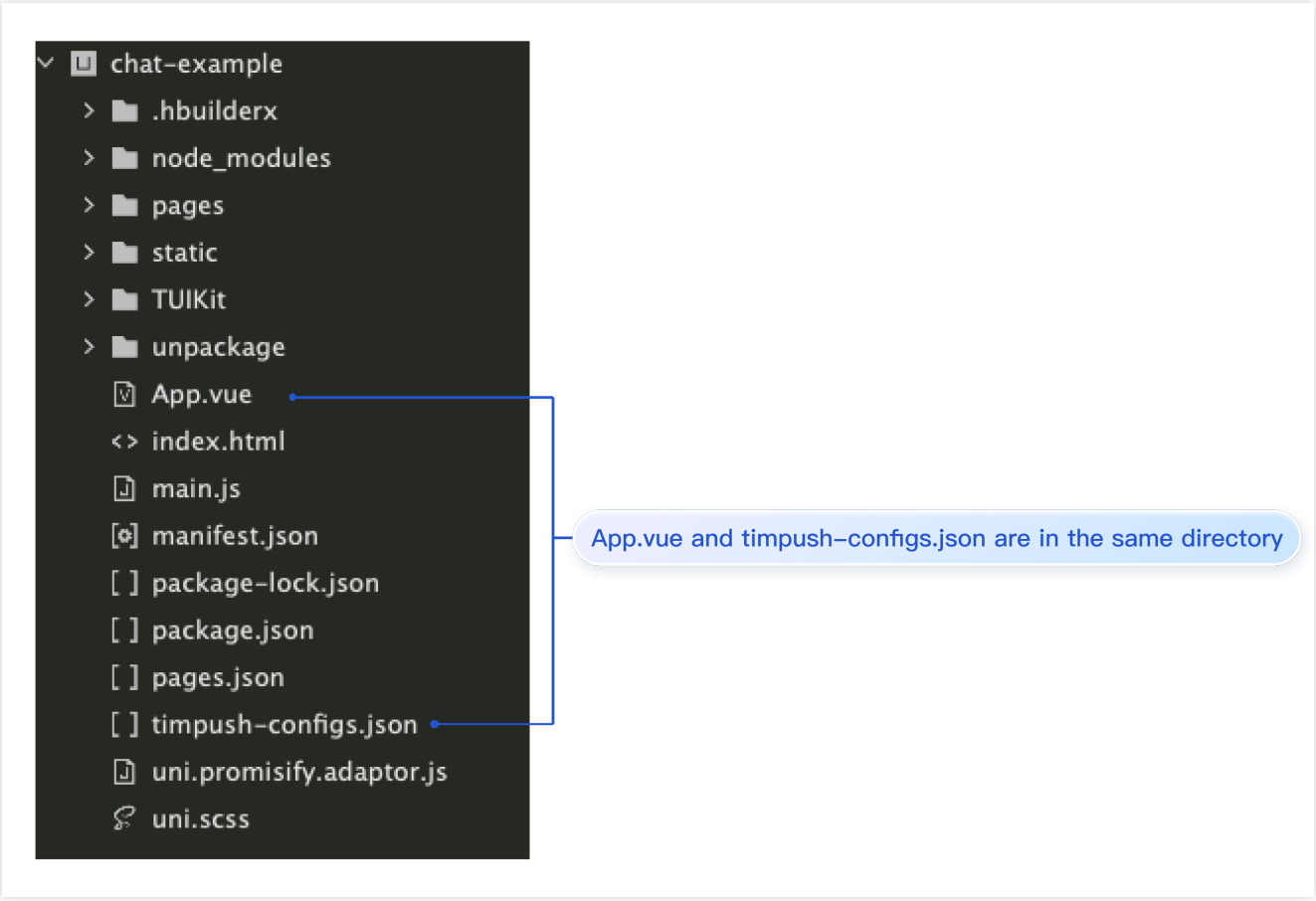
iOSConfig is the iOS push configuration. If you don't need to package an iOS App, you can pass null.

Retrieve the Android configuration timpush-configs.json

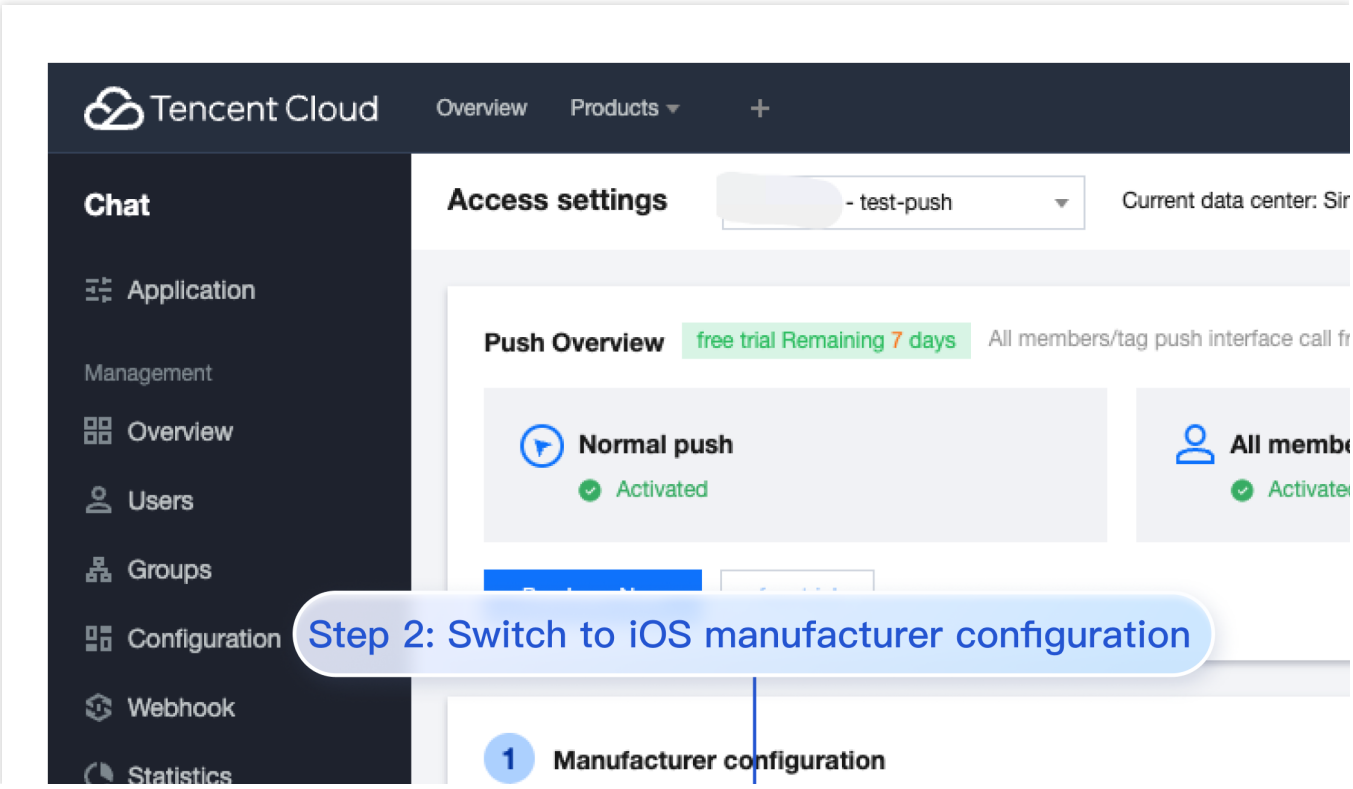
Retrieve iOS configuration iOSBusinessID

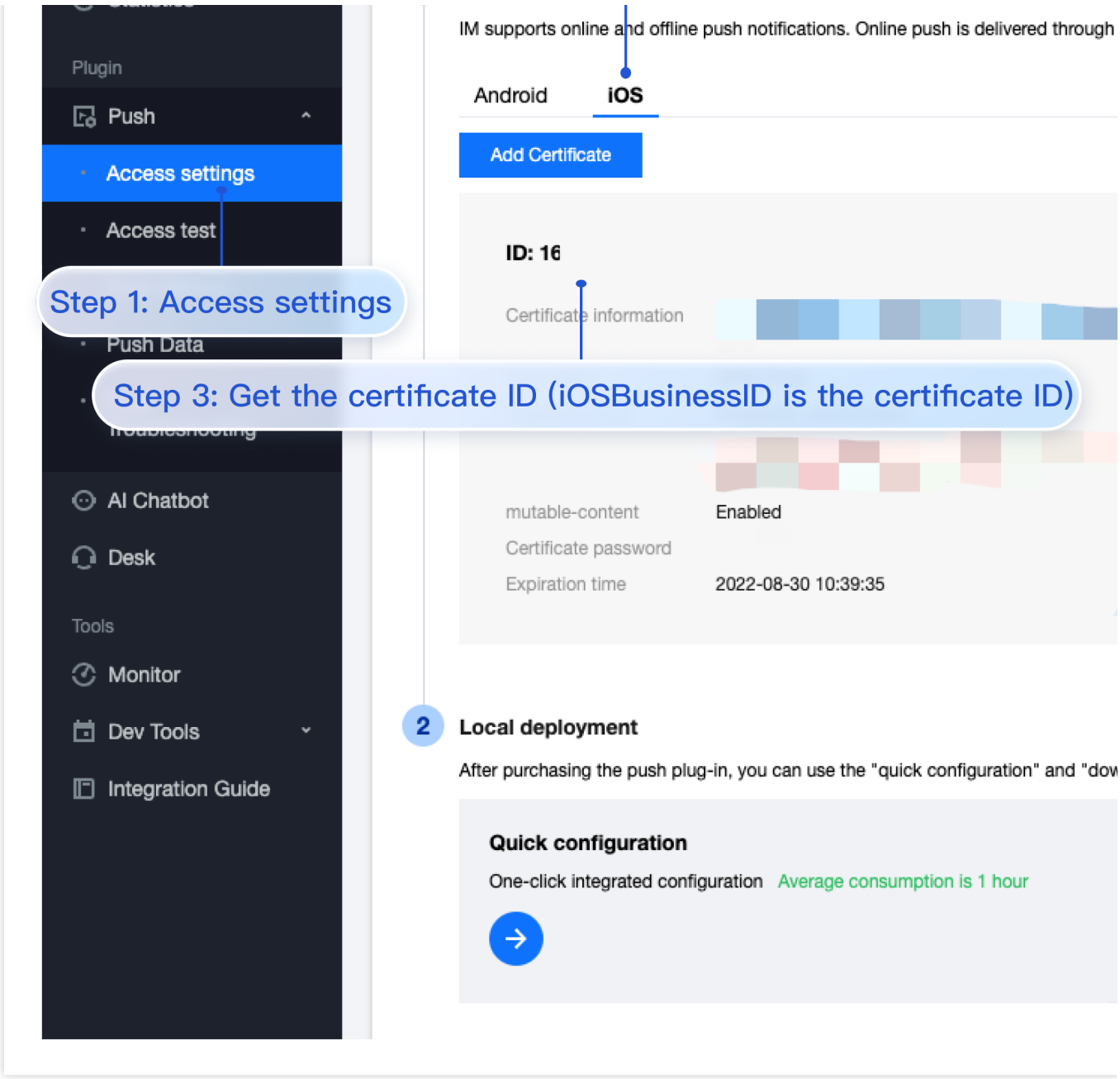
timpush-configs.json can be downloaded from IM console > Push > Access Settings, and placed in the same directory as `App.vue` , as shown:





iOS iOSBusinessID can be obtained from IM console > Push > Access Settings, as shown:





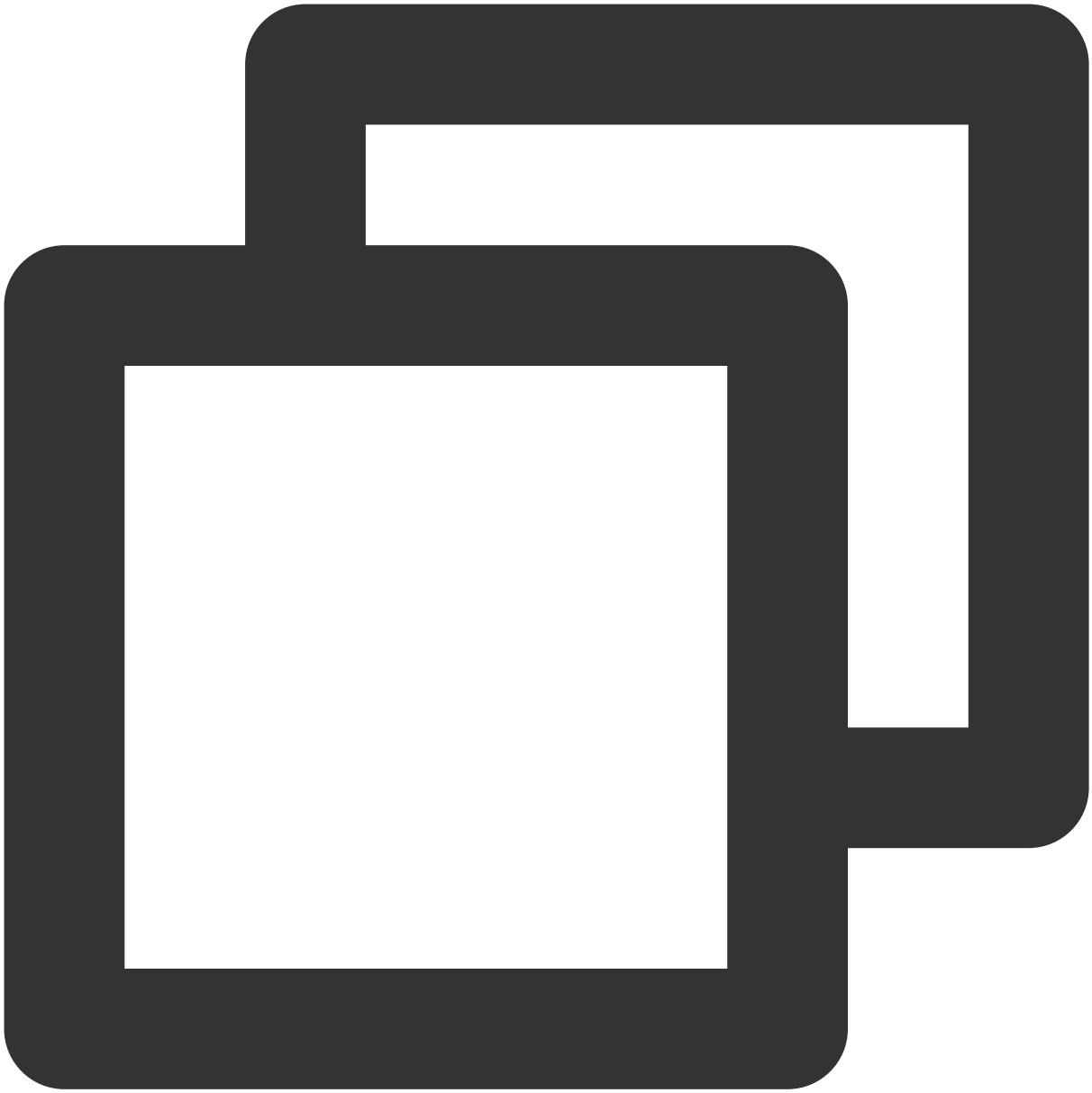
App.vue

Integration (UI Included)

Integration Without UI

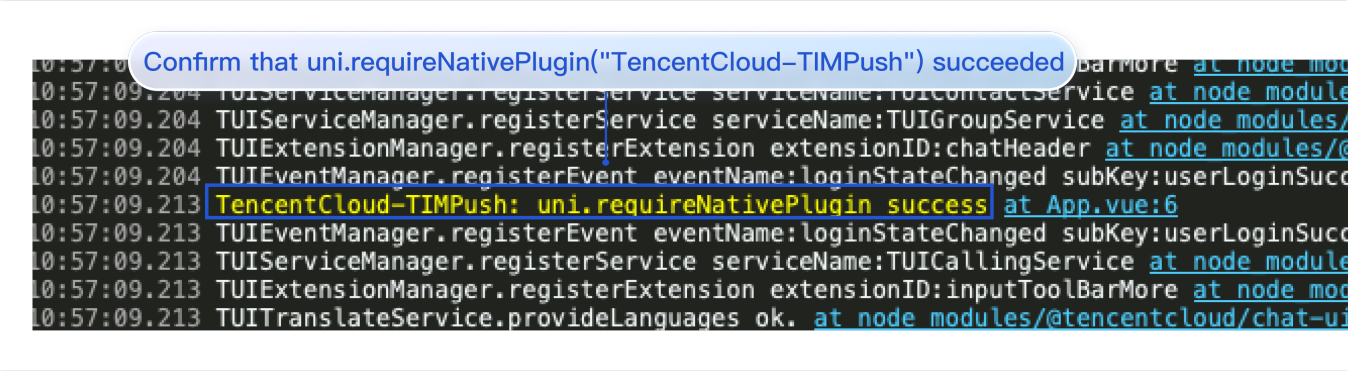
In App.vue, import TencentCloud-TIMPush, and mount it to uni.

In App.vue, import timpush-configs.json, and mount it to uni.

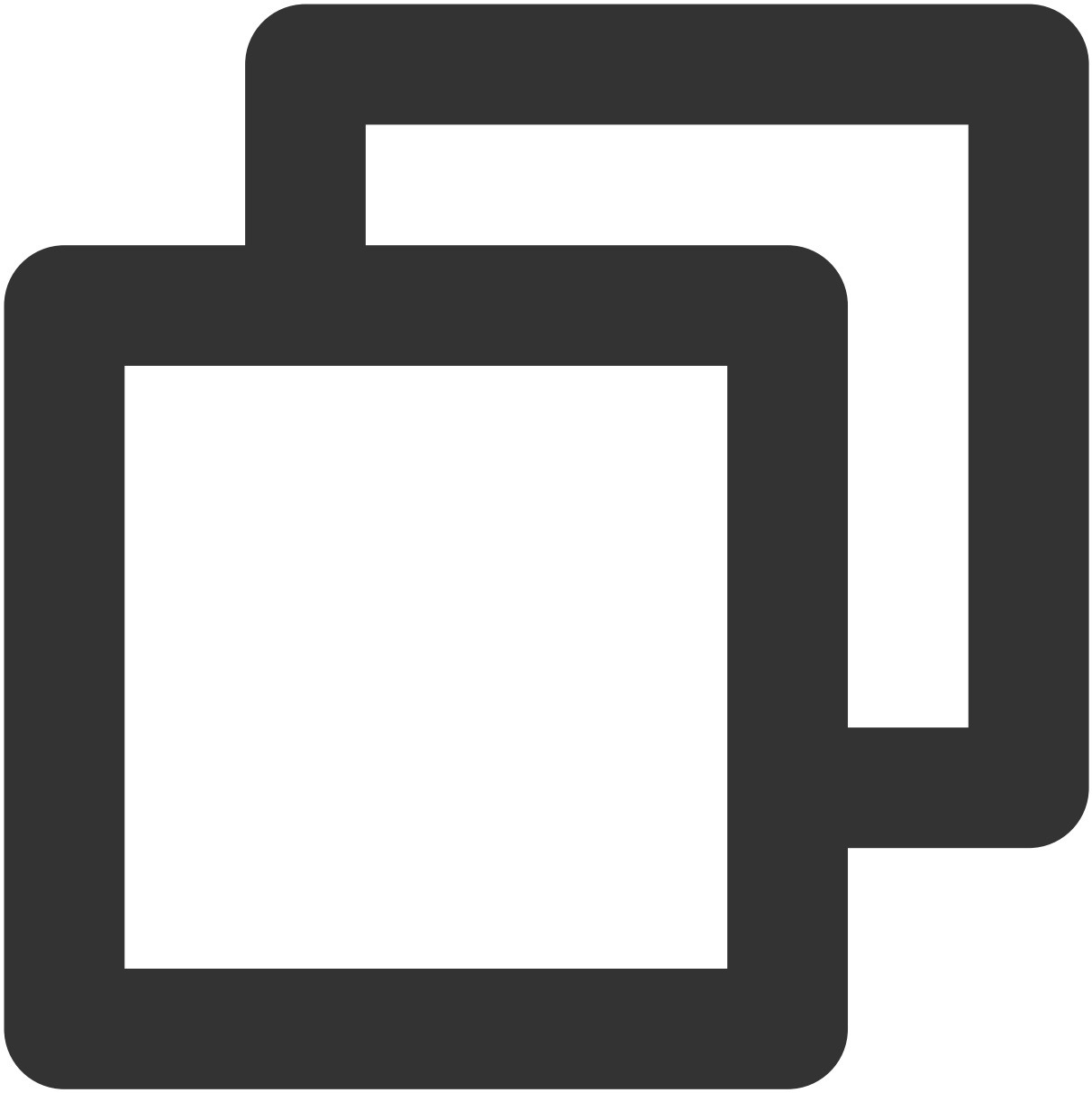


```
// #ifdef APP-PLUS
import TIMPushConfigs from "./timpush-configs.json";
const TIMPush = uni.requireNativePlugin("TencentCloud-TIMPush");
console.warn(`TencentCloud-TIMPush: uni.requireNativePlugin ${!!TIMPush ? 'success'
uni.$TIMPush = TIMPush;
uni.$TIMPushConfigs = TIMPushConfigs || {};
// #endif
```

Ensure uni.requireNativePlugin successfully imports TencentCloud-TIMPush, as shown in the figure:



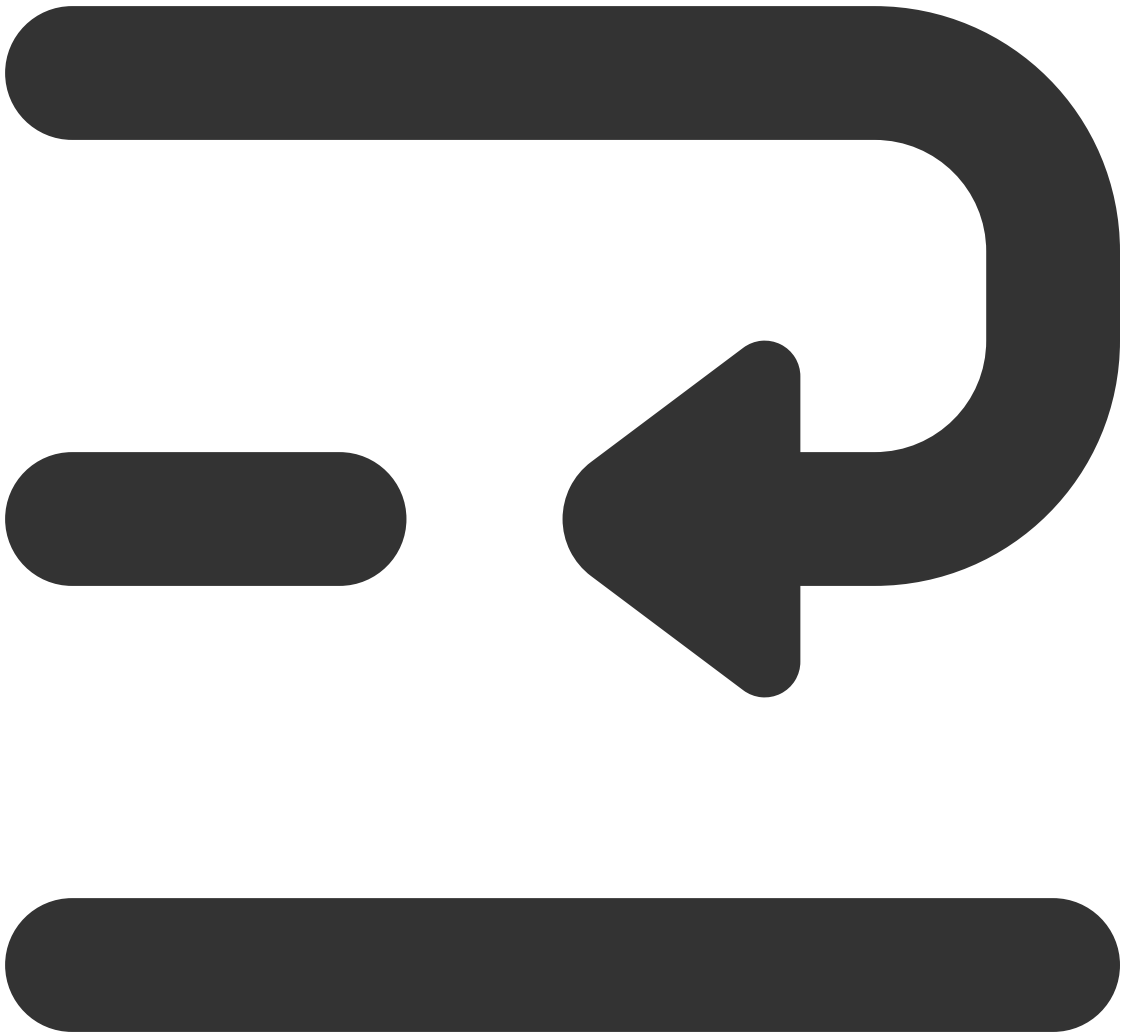
When logging into uikit, register TencentCloud-TIMPush into uikit.

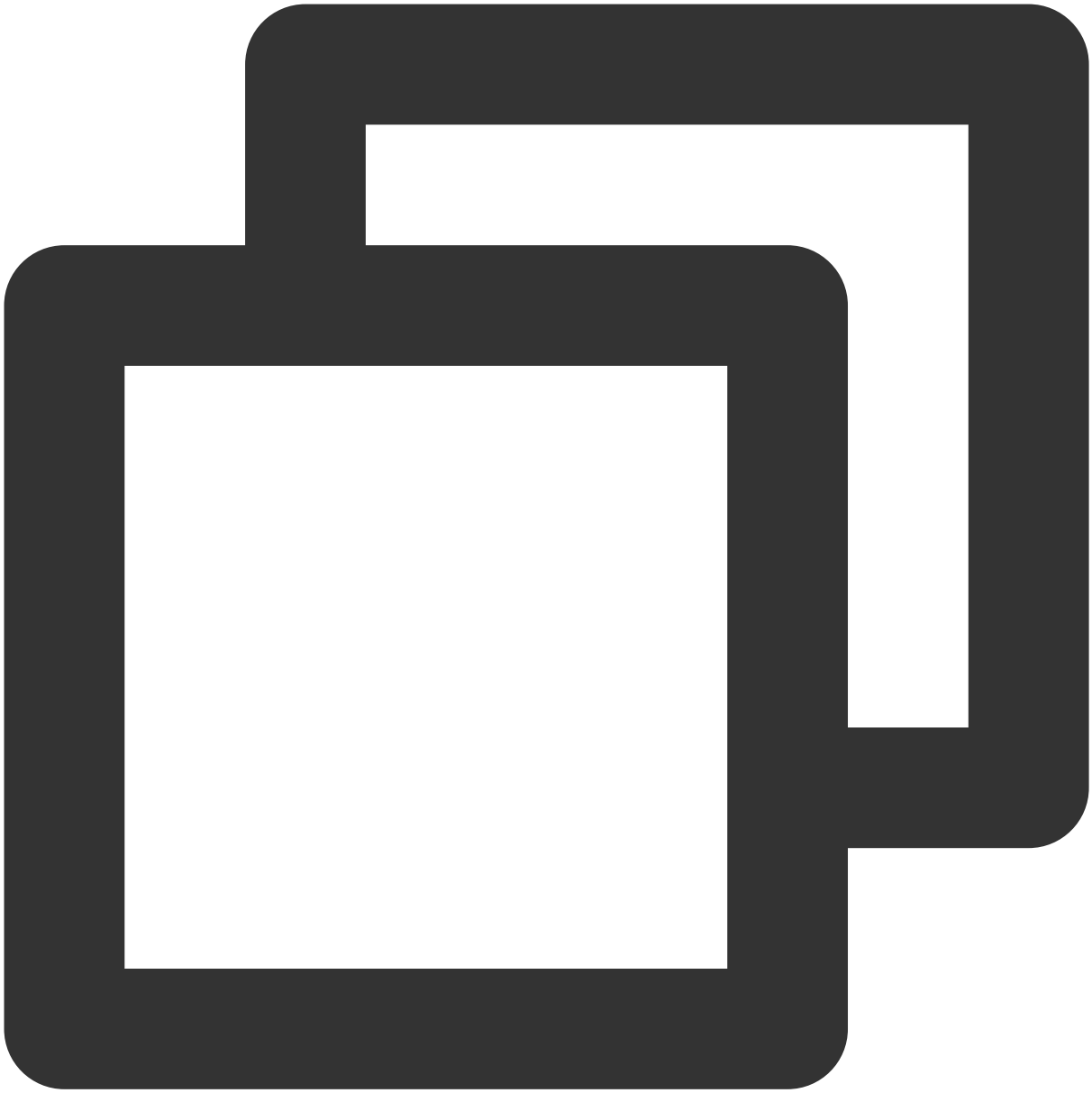


```
import { TUILogin } from "@tencentcloud/tui-core";
TUILogin.login({
    SDKAppID: 0, // Replace 0 with the SDKAppID of your IM app during integration.
    userID: '',
    // UserSig is the cipher for users to sign in to Instant Messaging, essentially
    // This method is only suitable for running Demo locally and debugging function
    userSig: '',
    // Should you require to send image, voice, video, file and other rich media me
    useUploadPlugin: true,
    framework: `vue${vueVersion}` // Current development uses framework vue2 / vue3
```

```
    TIMPush: uni.$TIMPush, // APP registers Push Plugin
    pushConfig: {
        androidConfig: uni.$TIMPushConfigs, // Android push configuration, pass nul
        iOSConfig: {
          "iOSBusinessID": "" // iOS push configuration, pass null if not needed.
        }
    }
})
```

Before logging into chat, register TencentCloud-TIMPush in the chat.





```
import TencentCloudChat from '@tencentcloud/chat';
const chat = TencentCloudChat.create({SDKAppID: 0}) // Replace 0 with the SDKAppID
chat.registerPlugin({
  'tim-push': uni.$TIMPush,
  pushConfig: {
    androidConfig: uni.$TIMPushConfigs, // Android push configuration, pass null if
    iOSConfig: {
      "iOSBusinessID": "xxx" // iOS push configuration, pass null if not needed.
     }
  }
})
```

```
chat.login({
    userID: '', // User ID.
    userSig: '' // The password for logging into Chat, which is essentially the cip
})
```

Step 5. Create a Custom Base

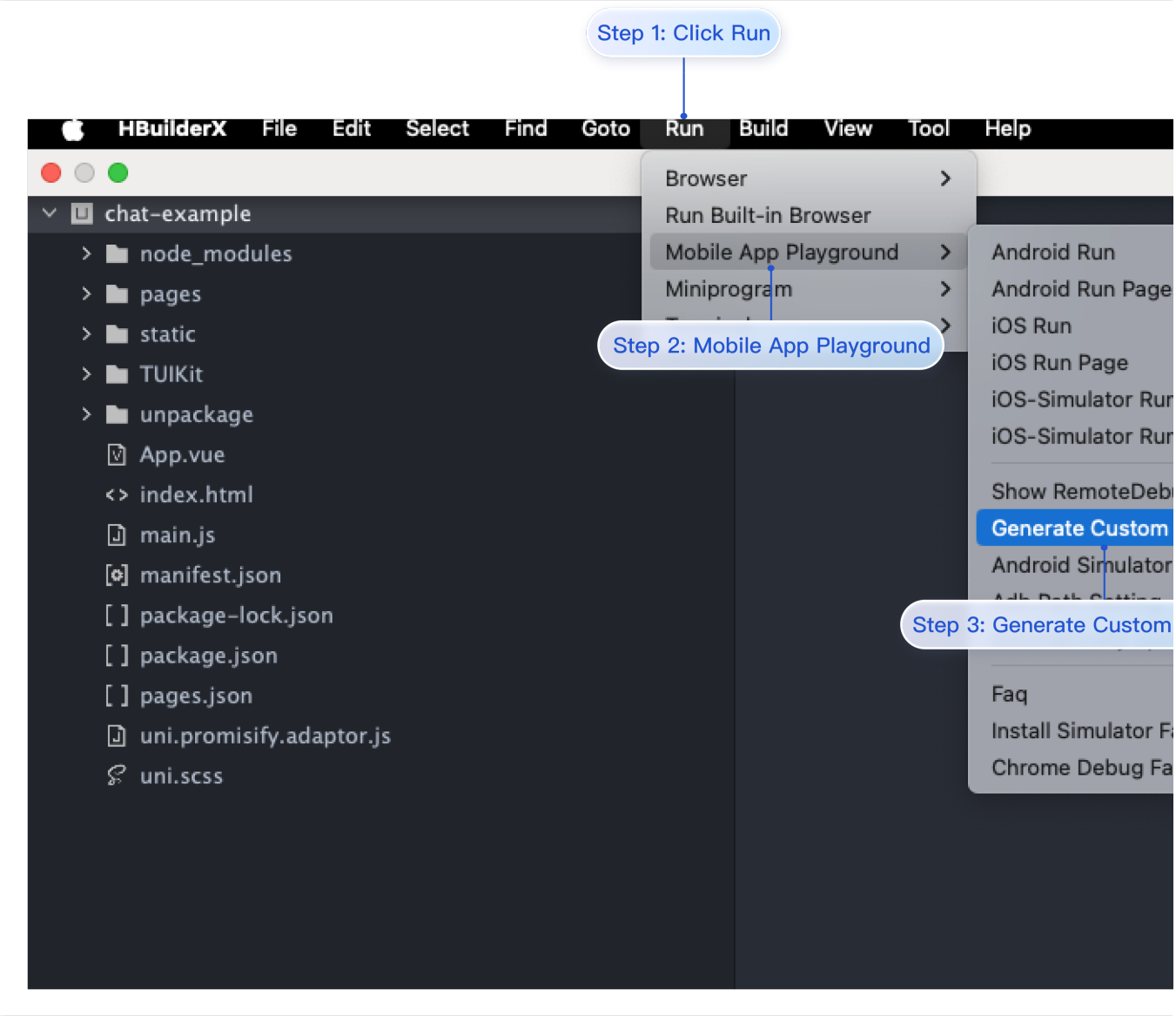
click HBuilderX's **Run**> **Run on Mobile or Emulator**> **Make a Custom Debugging Base**Create a Custom Base.

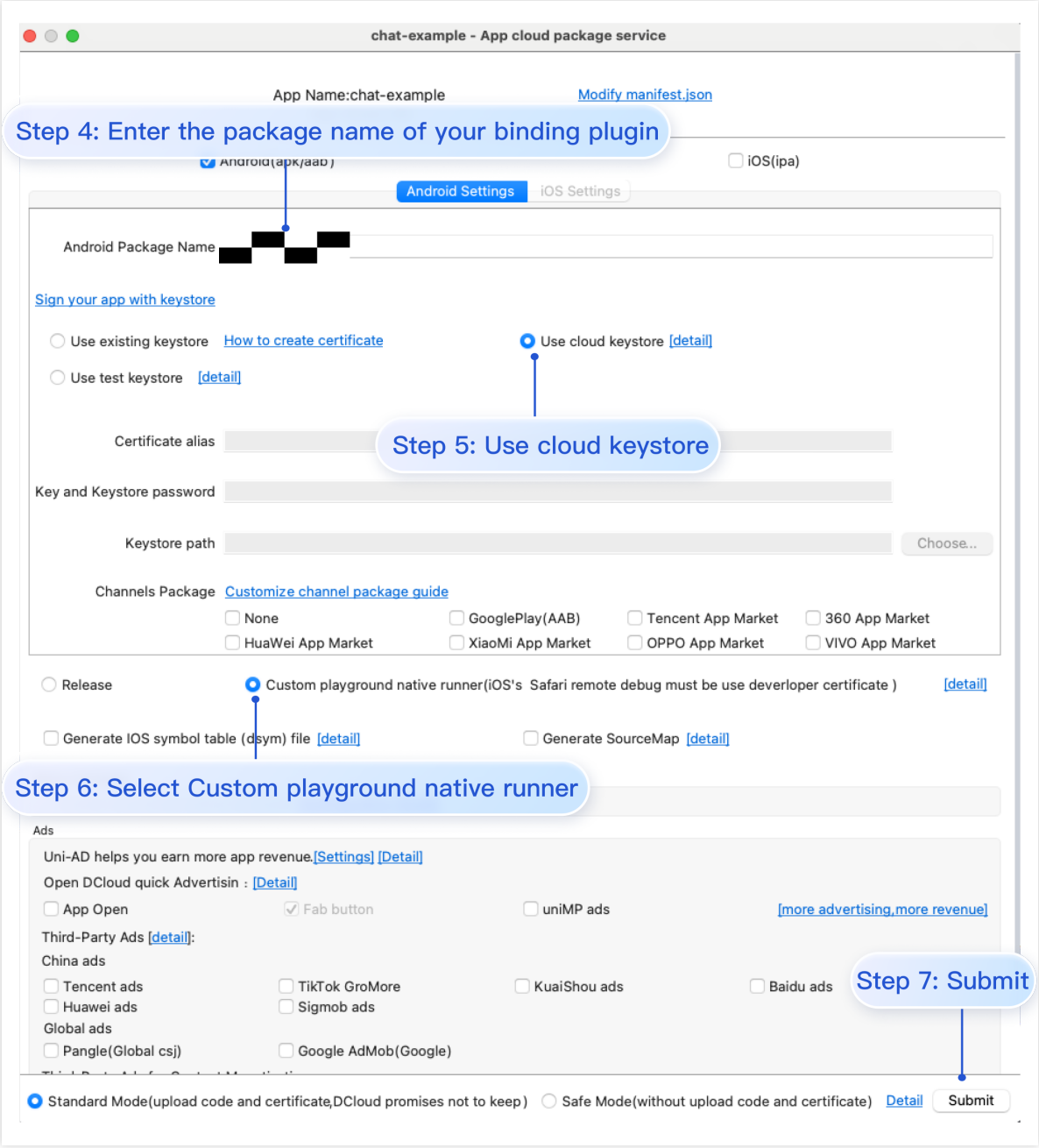
Note:

Configure Native Plugins, it is necessary to package a Custom Base for testing.

When creating a base, the Android package name is the application package name bound during the activation of cloud packaging for plugins.

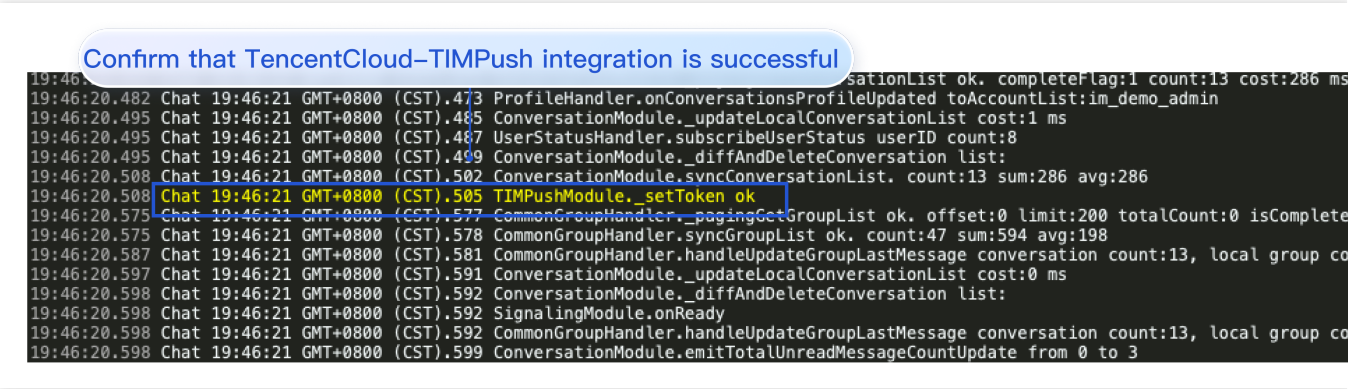
Use cloud certificates.





Step 6. Run and log in to the project, confirm the integration of TencentCloud-TIMPush is successful.

During runtime, choose to run with a Custom Base. In HBuilder's log, confirm the printing of `TIMPushModule._setToken ok`, indicating successful integration of TencentCloud-TIMPush, as shown in the picture:

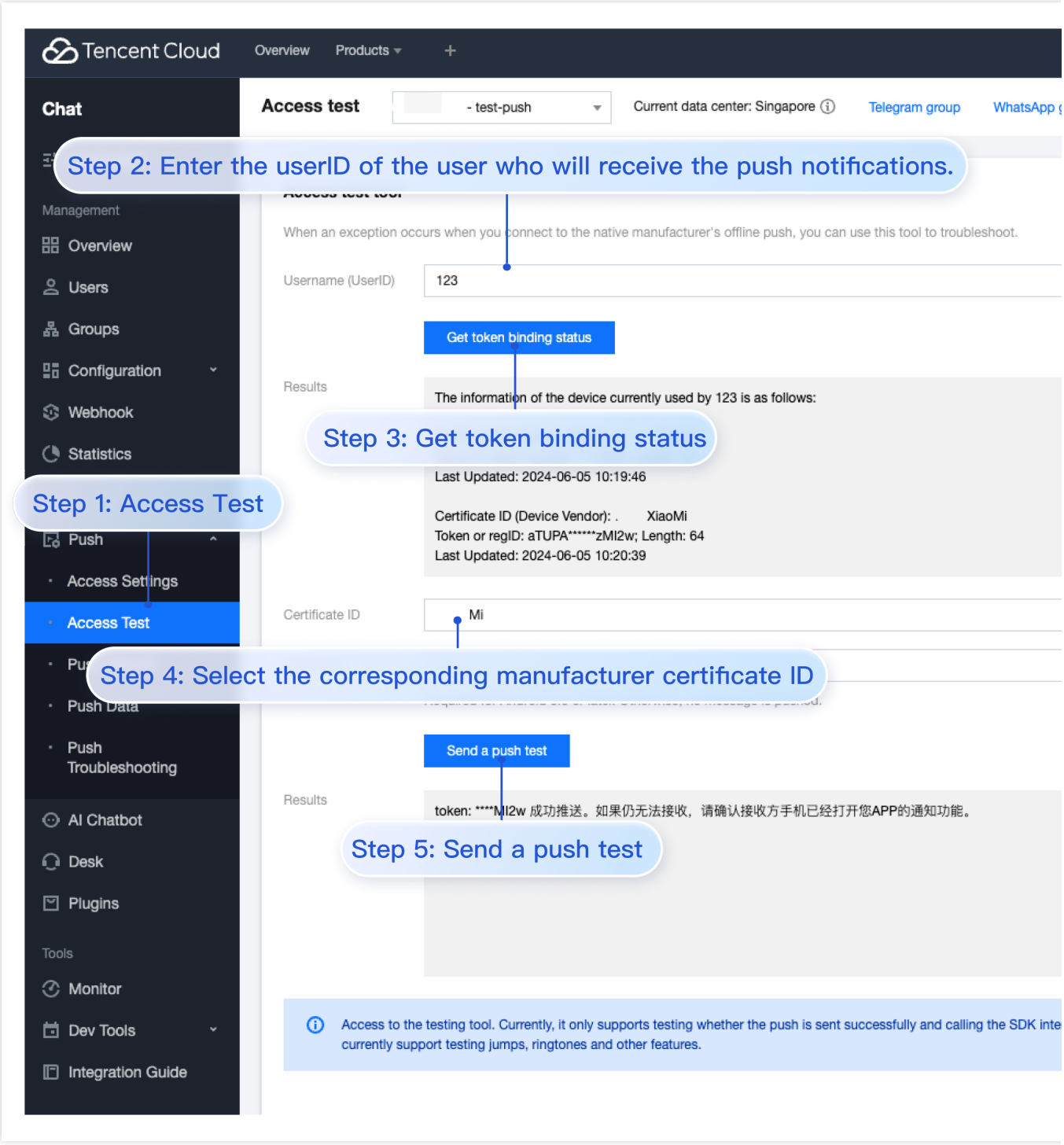


Step 7. Receive your first push notification.

Go to the IM Console Push > Push Test to send your first push notification. As shown in the figure:

Note:

1. The receiver's application must be in the background or the process terminated.
2. Live broadcast groups do not support offline message push.



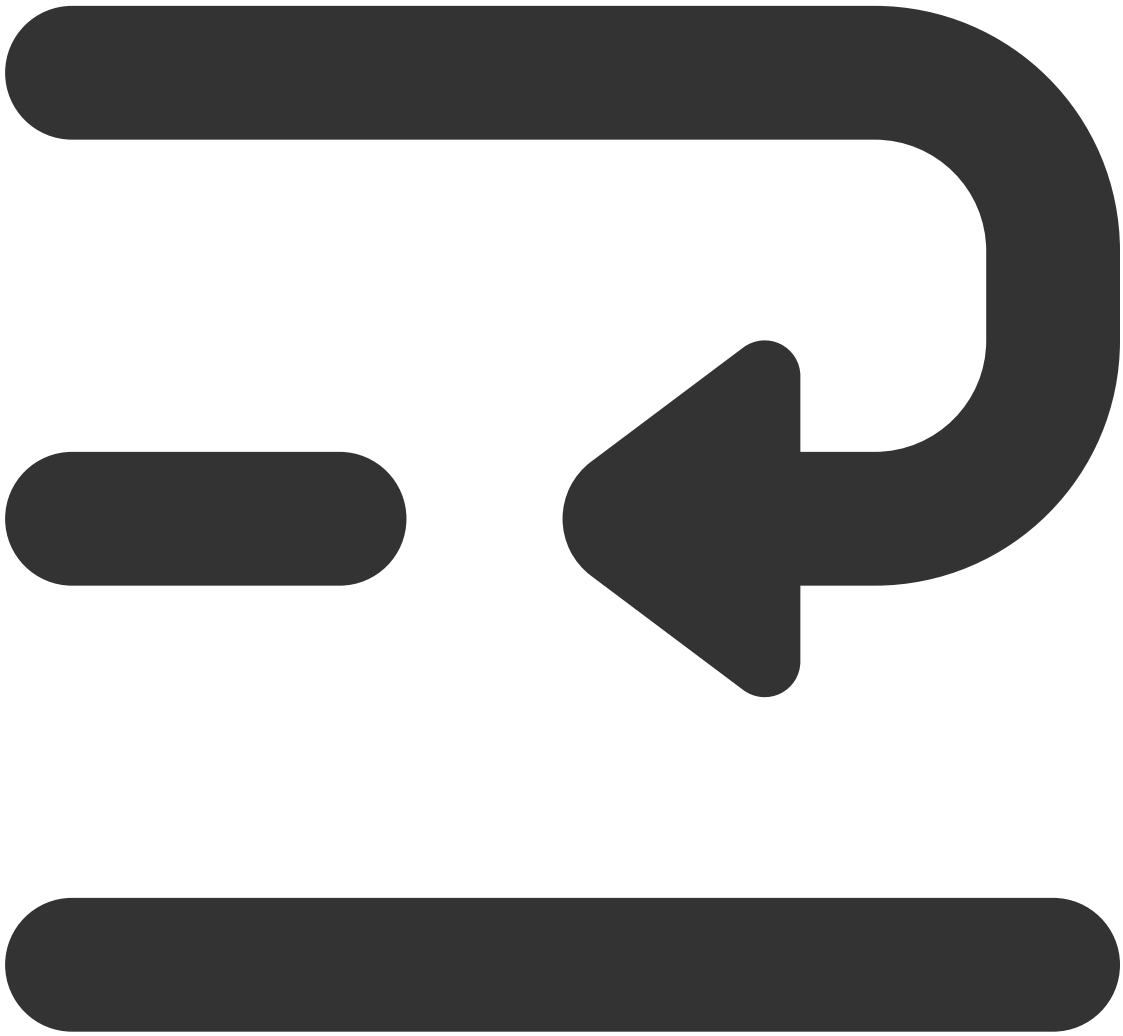
More Advanced Features (Highly Recommended)

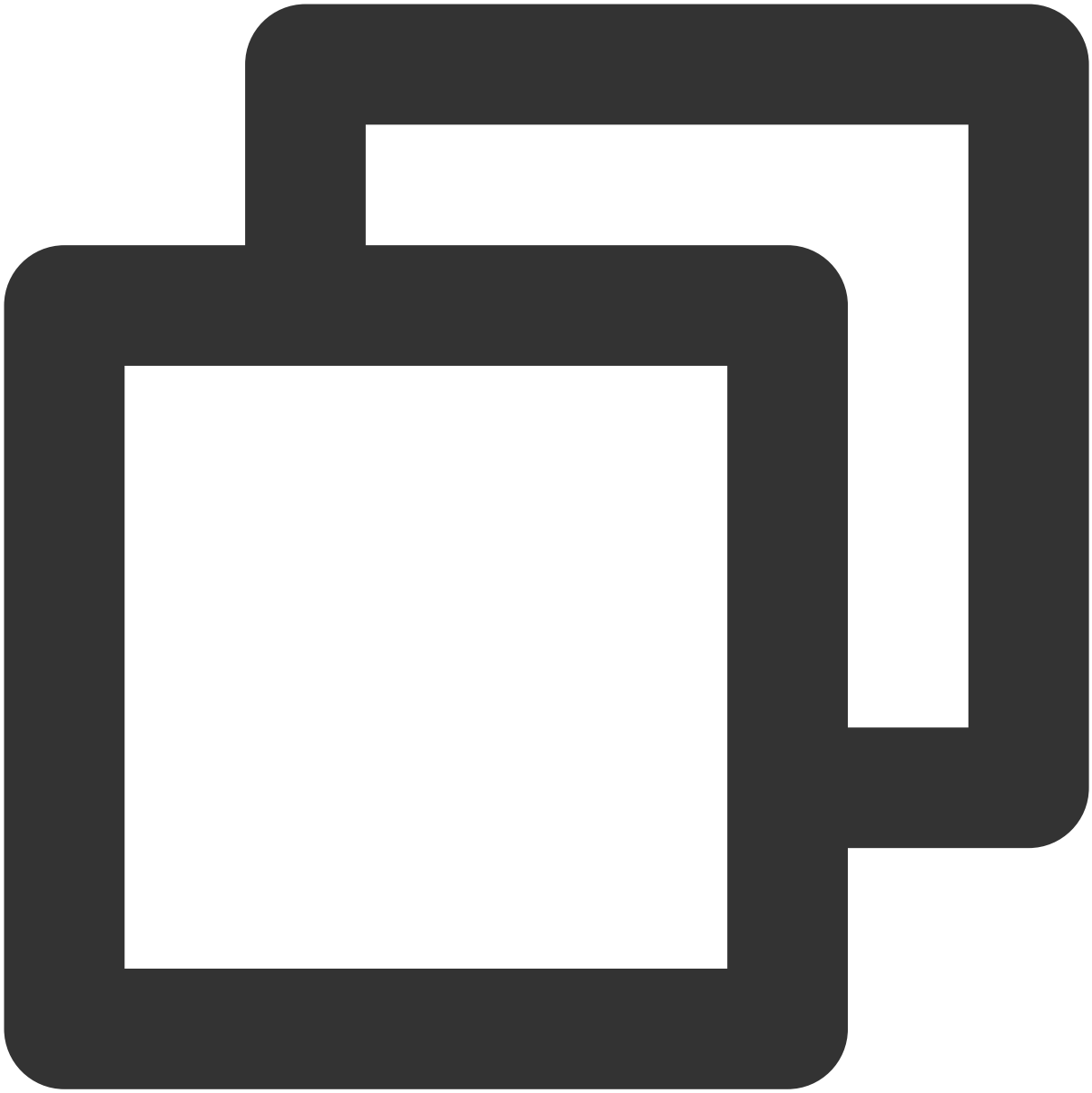
1. Set Push Content

Integration (UI Included)

Integration Without UI

When using TUIChatService to send messages in uikit, set the related parameters for offlinePushInfo. For example, when sending a standard text message, the code is as follows:



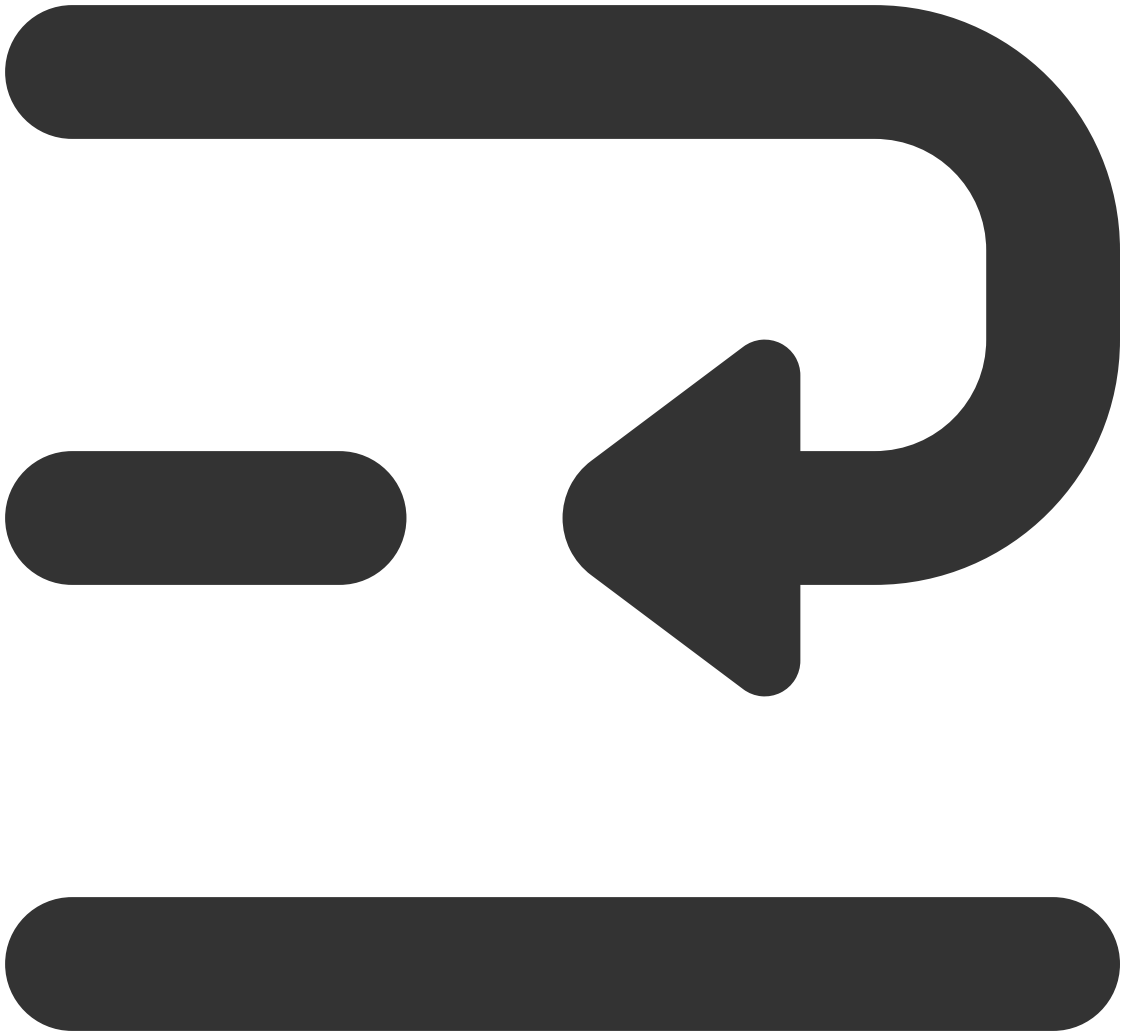


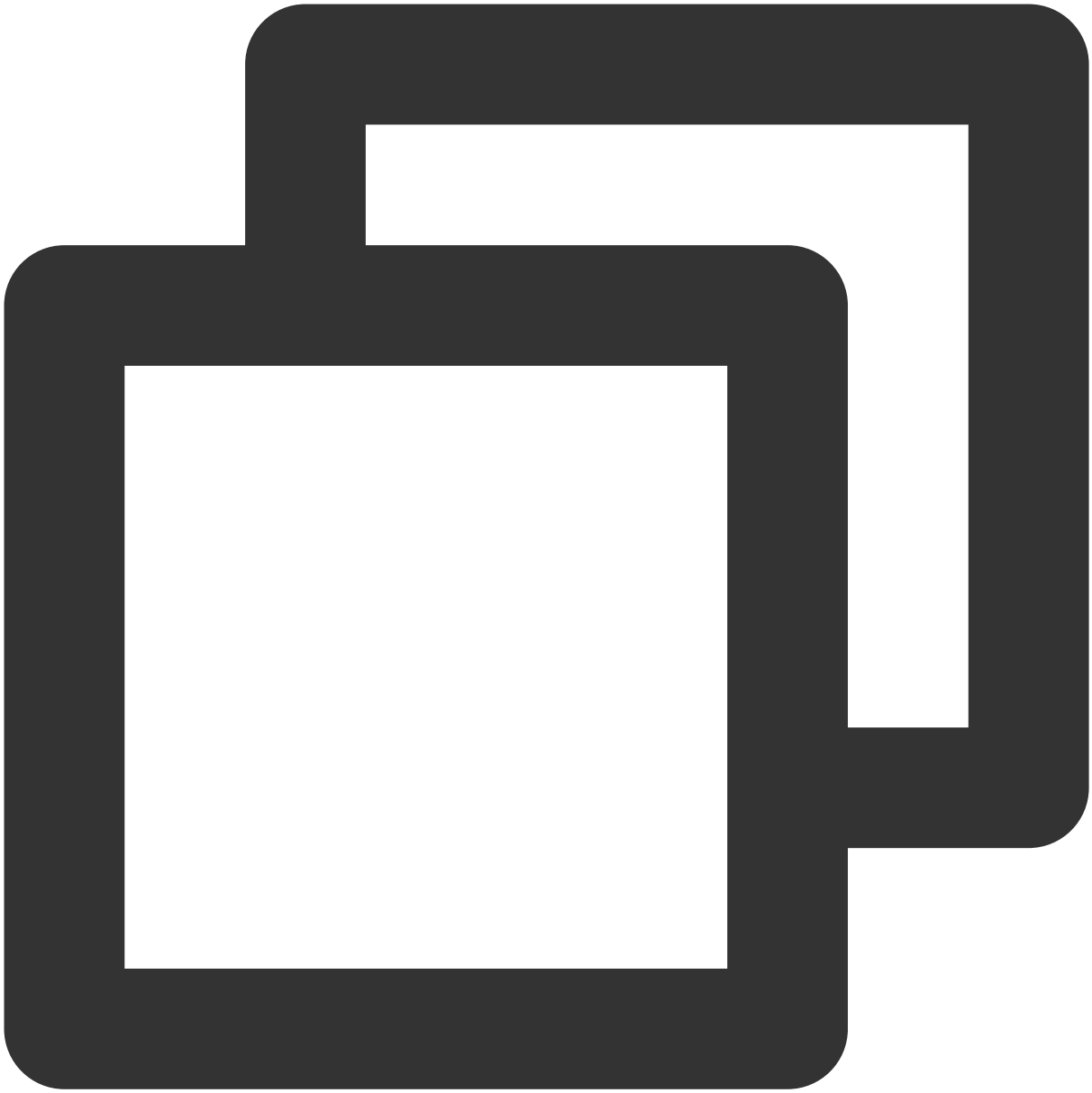
```
// Send Plain Text Message
let promise = TUIChatService.sendTextMessage(
{
   payload: {   text: 'Hello world!' }
},
{
  // If the recipient is offline, the message will be stored for roaming and an off
   offlinePushInfo: {
     title: '', // Offline Push Title.
     description: '', // Offline Push Content.
     extension: '', // Offline Push Transparent Content
```

```
    }
}
);
promise.catch((error) => {
    // Business side can handle errors by catching exceptions through promise.catch
});
```

Reference document: uikit - TUIChatService message sending documentation

When sending a message in chat, set the related fields for offlinePushInfo as follows:





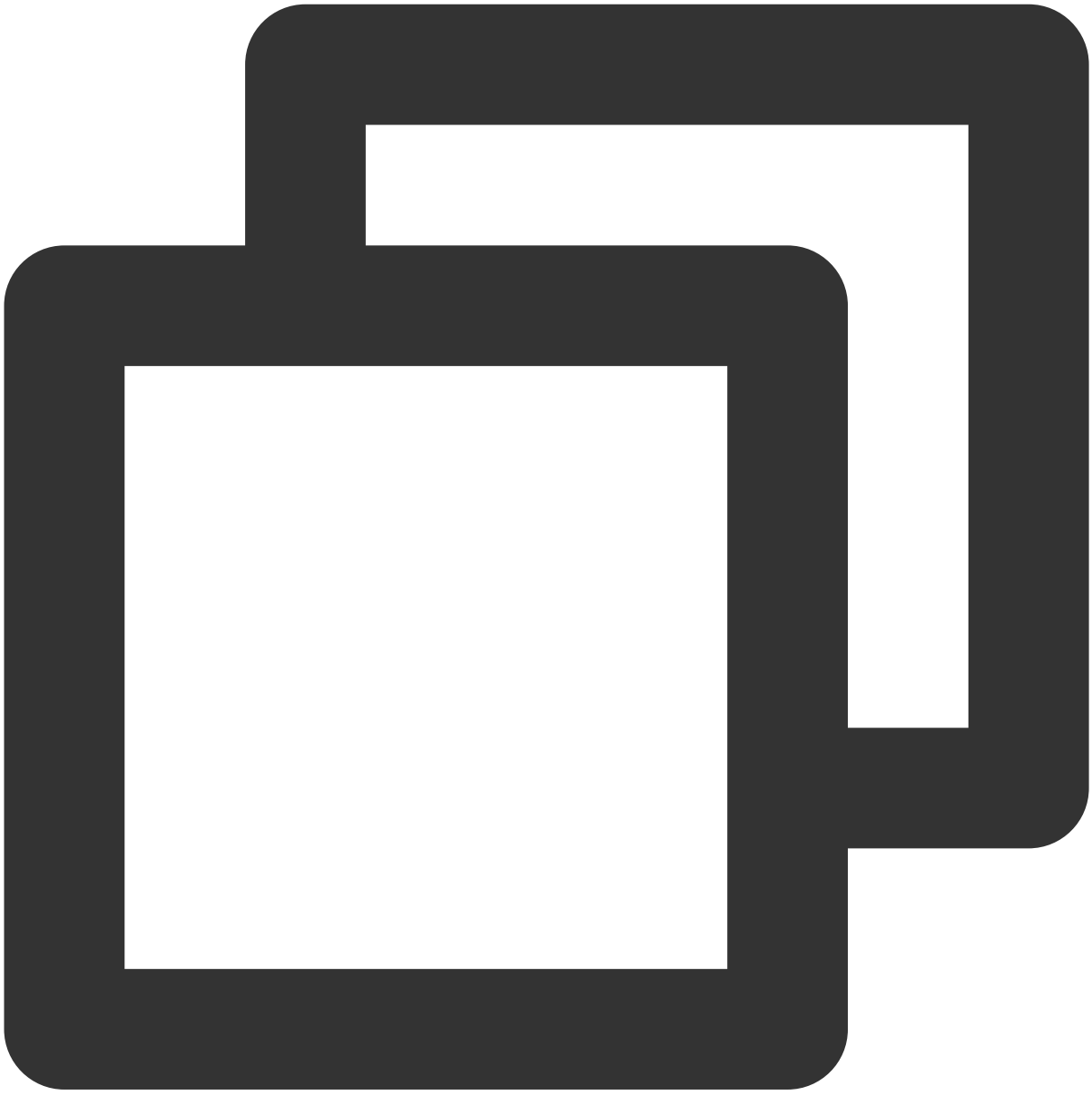
```
// Message Sending Options
chat.sendMessage(message, {
  // If the recipient is offline, the message will be stored for roaming and an off
  offlinePushInfo: {
     title: '', // Offline Push Title.
     description: '', // Offline Push Content.
     extension: '', // Offline Push Transparent Content
  }
});
```

Reference Documentation:

Chat SDK - sendMessage Documentation

2. Retrieving Click-through Transmission Content

Retrieve the transmission content in the `App.vue` file, and configure the specified jump page.



```
onLaunch: function () {
   // #ifdef APP-PLUS
   // In App.vue, listen in the lifecycle hook onLaunch
   if (uni.$TIMPush) {
    uni.$TIMPush.setOfflinePushListener((data) => {
        // The content within the pass-through entity, excluding the pushed Message
```

```
            console.log('setOfflinePushListener', data);
            // Redirect to a specified interface within the application
            uni.navigateTo({
                url: "/pages/xxx/xxx",
            });
        });
    }
    // #endif
}
```

Note:

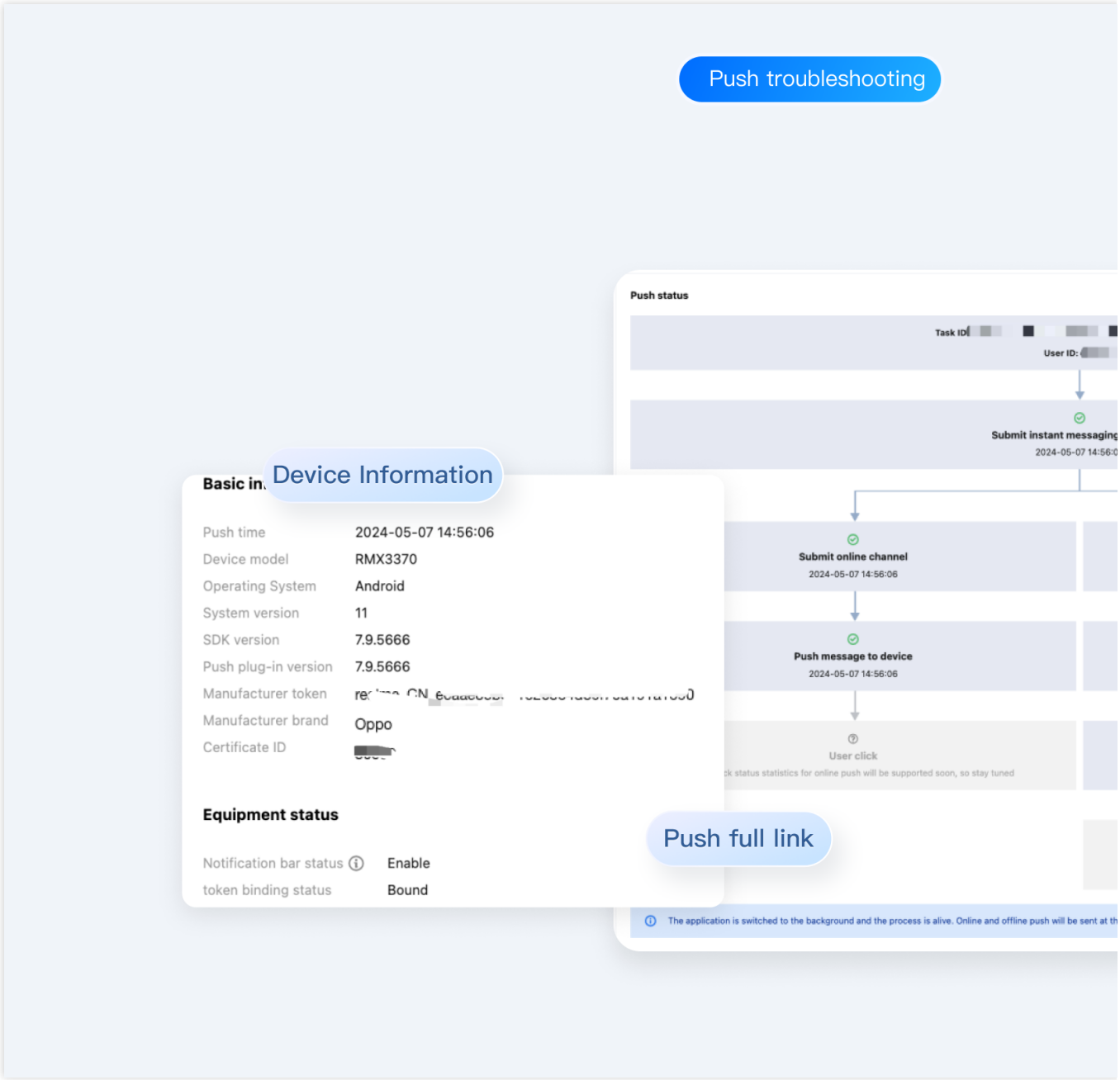
Here you can retrieve the pass-through content configured during the push notification.

Here you can configure the application redirect interface.

During cross-platform interoperability, ensure that `extension` remains consistent, `extension` must include the `entity` field.

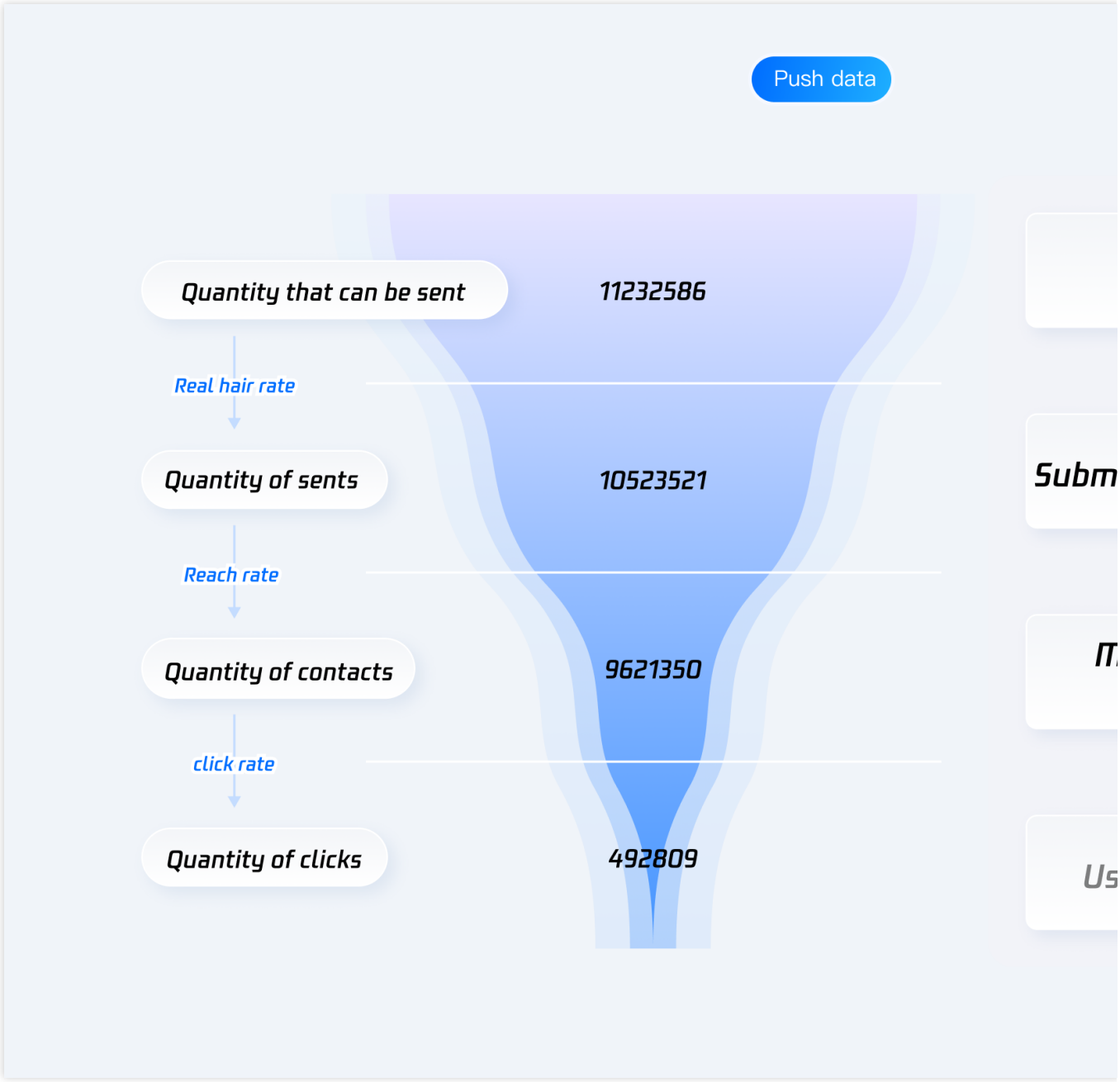
Anomaly Troubleshooting

If users do not receive offline push notifications when the App is offline, they can use the Troubleshooting Tools for end-to-end troubleshooting of push details.



Statistics Collection

Query all Statistics for a user, organize and analyze daily push metric data of various types, and generate recent **Sent > Reached > Clicked** funnel conversion charts, with support for viewing by manufacturer channel.



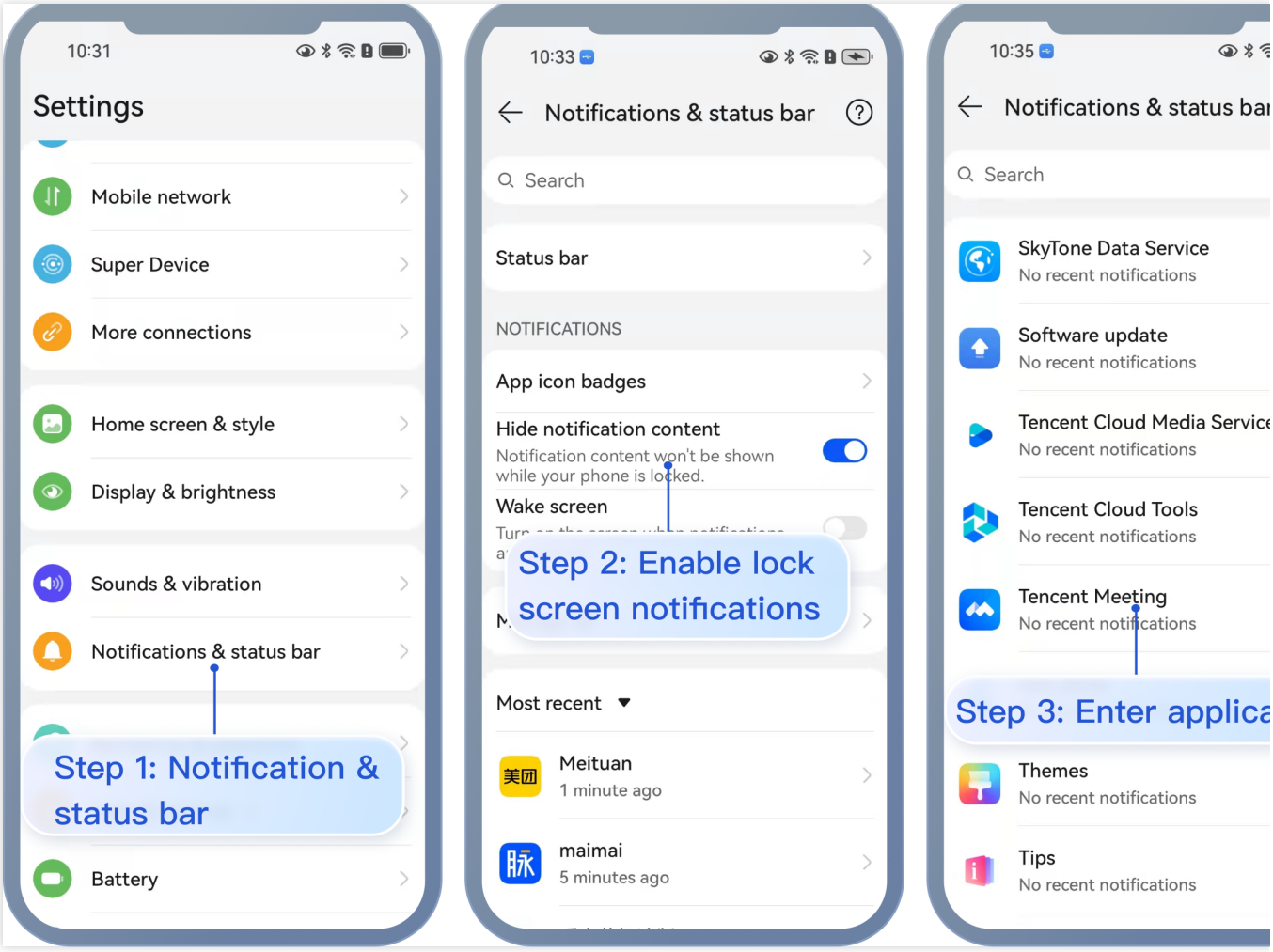
Device Notification Bar Settings

The direct manifestation of a push is a notification bar prompt, which, like other notifications, is affected by device notification-related settings, taking Huawei as an example:

"Settings - Notifications - Notifications (Lock Screen) - Hide or Do not Disturb" will affect the display of offline push notifications when the screen is locked.

"Settings - Notifications - Advanced Settings - Show Notification Icons (Status Bar)" will affect the showing of the offline push notification icon in the status bar.

"Settings - Notifications - Application Notifications - Allow Notifications" will directly affect the display of offline push notifications.
"Settings - Notifications - Application Notifications - Notification Sound" and "Settings - Notifications - Application Notifications - Notification Mute" will affect the offline push notification sound.



Vendor's push restrictions

1. All vendors in China have adopted message classification mechanisms, and different push policies are assigned for different types of messages. To make the push timely and reliable, you need to set the push message type of your app as the system message or important message with a high priority based on the vendor's rules. Otherwise, offline push messages are affected by the vendor's push message classification and may vary from your expectations.
2. Additionally, some manufacturers impose limits on the daily number of pushes for an application. You can check the daily push quantity limits in the Manufacturer Console. If offline push messaging is not timely or is occasionally not received, consider the following:

Huawei	viv	OPPO	Mi	Meizu	FCM
Push messages are divided into Service and Communication Category and Information Marketing Category, with different push effectiveness and policies. Additionally, message classification is also related to self-categorization benefits: If there is no self-help message classification permission, the vendor will perform secondary intelligent message classification on push messages. If you have applied for self-categorization benefits, push messages will be classified according to your own defined categories for pushing. For more details,	Push messages are divided into system messages and operational messages, with different push effectiveness and policies. System messages will also undergo a secondary correction by the vendor's intelligent classification. If a message is identified not as a system message, it will automatically be corrected to an operational message. If there is a misjudgment, you can provide feedback via email. Additionally, the number of push messages is restricted by	Push messages are divided into private messages and public trust messages, with different push effectiveness and policies. Private messages are for users who are significantly attentive to and wish to receive information promptly. Privileges for the private message channel need to be applied for via email. The number of pushes through the public trust channel is limited. For more details, please see Manufacturer Description 1 or Manufacturer Description 2.	Push messages are divided into important messages and general messages, with different push effectiveness and policies. The important message type is reserved exclusively for instant messaging, personal follow-up dynamic alerts, personal matters reminders, personal order status changes, personal financial reminders, personal status changes, personal resource changes, and personal device alerts. These eight types of messages can be activated through an	The number of push messages is limited. For more details, please see Manufacturer Description.	The frequency of push uplink messages is limited. For details, please see Manufacturer Description.

please see Manufacturer Description.	a daily limit, which is determined by the app's subscription statistics with the vendor. For more details, please see Manufacturer Description 1 or Manufacturer Description 2.		application in the manufacturer's console. The number of general message type pushes is limited. For more details, please see Manufacturer Description 1 or Manufacturer Description 2.		

FAQs

1. TencentCloud-TIMPush and uniPush2 are in conflict and cannot be shared. How should this be addressed?

Reason: TencentCloud-TIMPush does not support sharing with other offline push channels.

Solution: It is recommended to use only TencentCloud-TIMPush.

2. Error in push: "TIMPushModule._getDeviceToken failed. error: { "code": -1, "msg": "huawei ApiException: com.huawei.hms.common.ApiException: 907135000: arguments invalid"}".

Reason: The Huawei agconnect-services.json file has not been imported.

Solution: Check if the Huawei agconnect-services.json file has been imported.

Note:

Huawei Push requires the agconnect-services.json file downloaded from the official website to be placed in the `nativeResources/android/assets` directory. For details, see uniapp.

3. Error in push: "TIMPushModule._getDeviceToken failed. error:{"code":-1, "msg": "callback is not String"}".

Reason: Huawei version is too low, EMUI version needs to be greater than 10.

Solution: Upgrade or switch to a Huawei with EMUI version > 10 for push.

4. Error in push: "TIMPushModule._getDeviceToken failed. error:{"code":22022, "msg": "xiaomi error code: 22022"}".

Reason: The application package name is invalid.
Solution: Go to the Xiaomi Push Platform to check if the application's package name, appId, and appKey match.

5. Can't receive push notifications on OPPO mobile phones?

Cause: The notification bar display feature is disabled by default for applications installed on the OPPO device.
Solution: Enable the notification bar display feature.

6. Followed the integration process but still can't receive push notifications?

Cause:
Device status anomaly, IM console settings not configured, initialization not registered, etc.
Push is dependent on vendor capabilities; some simple characters may be filtered by the vendor and not be transparently pushed.
If push messages are not sent promptly or are occasionally not received, you need to check the vendor's push restrictions.
Solution:
Investigation can be conducted using the Troubleshooting Tools.
Check the content to be sent, avoid using simple characters.
Check the manufacturer's push restrictions, details can be seen in Manufacturer Push Restrictions.

7. Why aren't regular messages receiving offline push notifications on iOS?

Reason: The inconsistency between the app's running environment and the certificate environment.
Solution:
Check if the app's running environment matches the certificate environment. If they do not match, offline push notifications will not be received.
Check whether the app and certificate environment is set to production. If it's in the development environment, requesting a deviceToken from Apple might fail, but this issue hasn't been observed in the production environment, please switch to the production environment for testing.

8. In the iOS development environment, registration occasionally does not return a deviceToken or indicates an APNs request token failure?

Cause: This issue arises due to instability in the APNs service.
Solution:
Insert a SIM card into the phone and use the 4G network.
Uninstall and reinstall the application, restart the application, or shut down and restart the phone.
Use a package for the production environment.
Use another iPhone.

9. Why is there no token in iOS?

Cause:

Emulators do not generate tokens.

Real Device, you need to enable push permissions on the phone.

Real Device, you need to add push notification's entitlement.

Solution: Use a real device to enable push permissions, and add push notification's entitlement.

10. Can tim-js-sdk use TIMPush?

Cause: Does not support TencentCloud-TIMPush;

Solution: Upgrade @tencentcloud/chatAccess, @tencentcloud/chat is downward compatible, and will not affect the business implemented by tim-js-sdk.

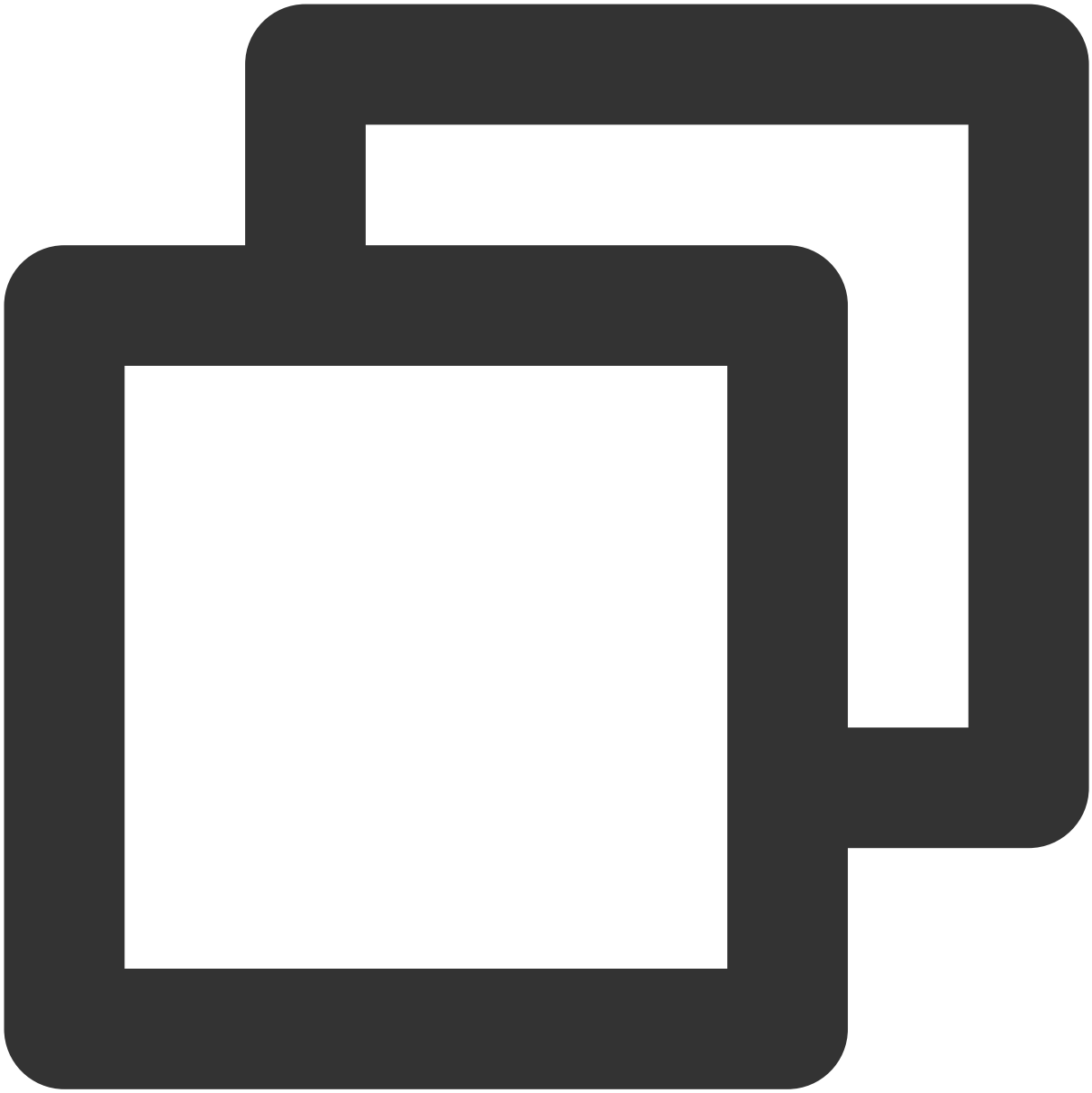
Flutter

Last updated：2024-06-13 10:21:45

Operation step

Step 1: Integrate message push plugin

This plugin is known as `tencent_cloud_chat_push` on pub.dev. You can include it in your pubspec.yaml dependency directory, or run the following command for automatic installation.



```
flutter pub add tencent_cloud_chat_push
```

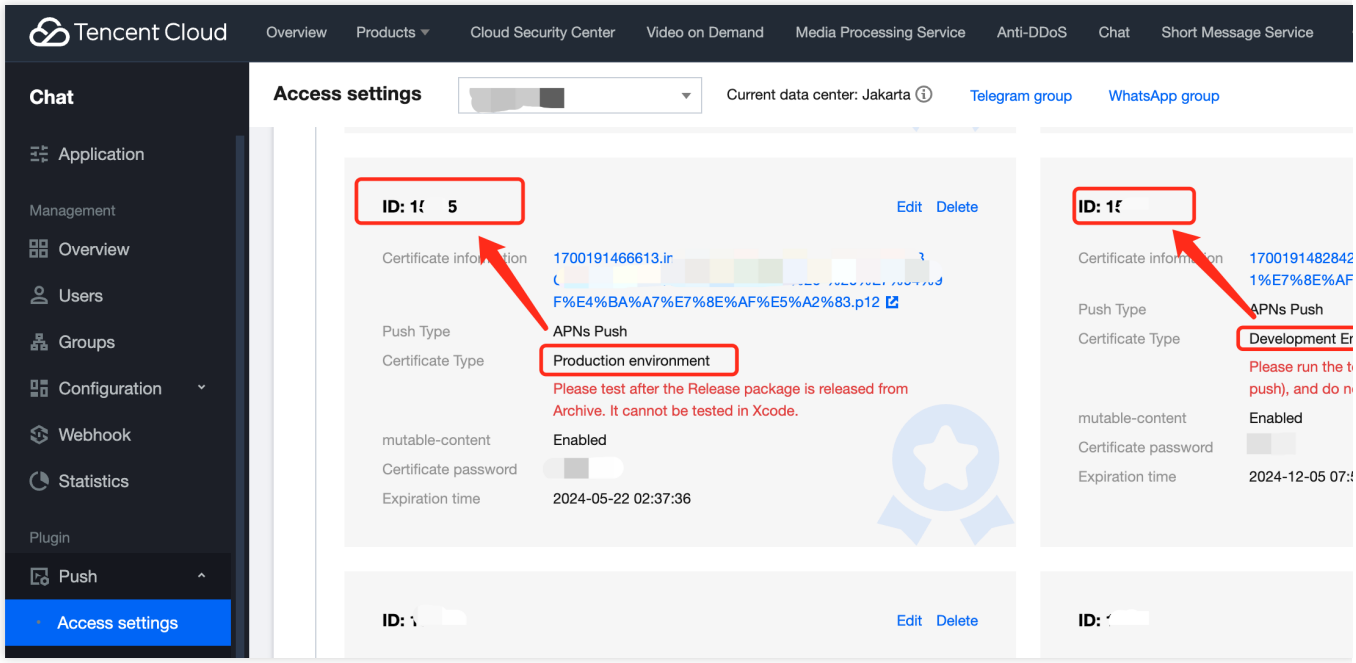
Step 2: Push parameter configuration

iOS

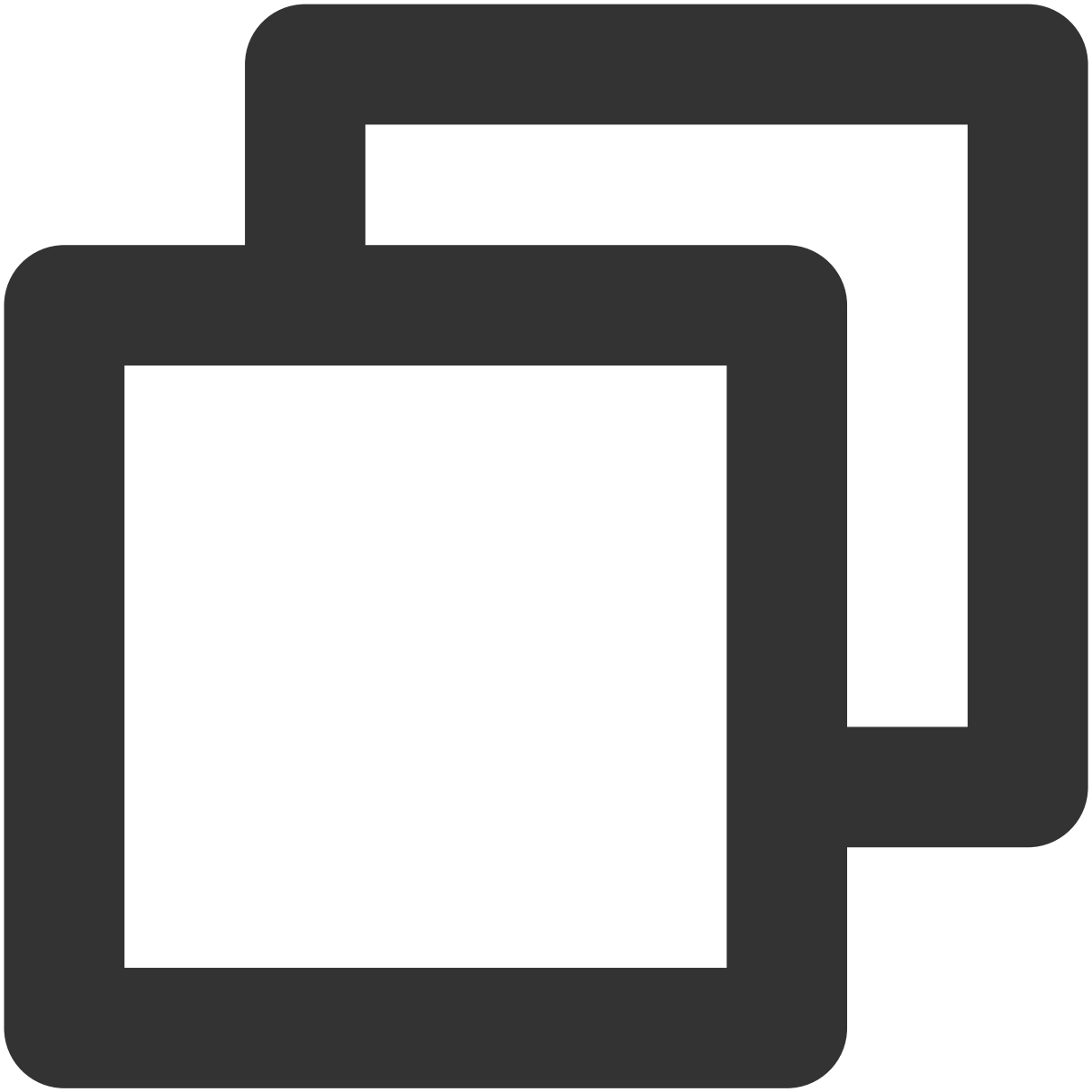
Android

Please upload the iOS APNs push certificate obtained during the vendor configuration step to the IM console.

The IM console will assign a certificate ID for you, as shown in the figure below:



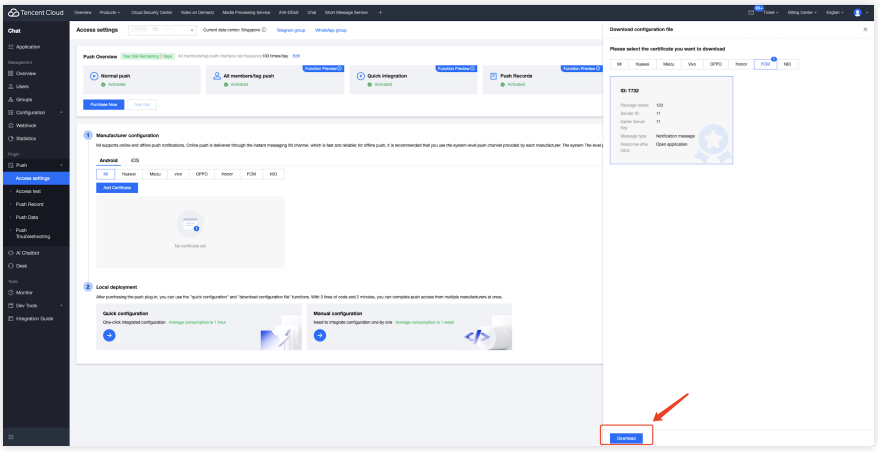
As early as possible after your application starts, call the `TencentCloudChatPush().setApnsCertificateID` method to pass in the certificate ID.

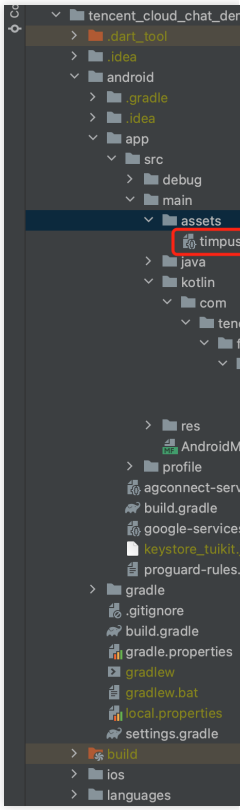


```
TencentCloudChatPush().setApnsCertificateID(apnsCertificateID: Certificate ID);
```

After completing the manufacturer push information on the console, download and add the configuration file to the project. Add the downloaded timpush-configs.json file to the `android/app/src/main/assets` directory. If the directory does not exist, please create it manually.

1. Choose to download the configuration file timpush-configs.json	1. Add to the project





Step 3: Client Code Configuration

In this step, you'll need to write some native code, such as: Swift, Java, XML, etc.

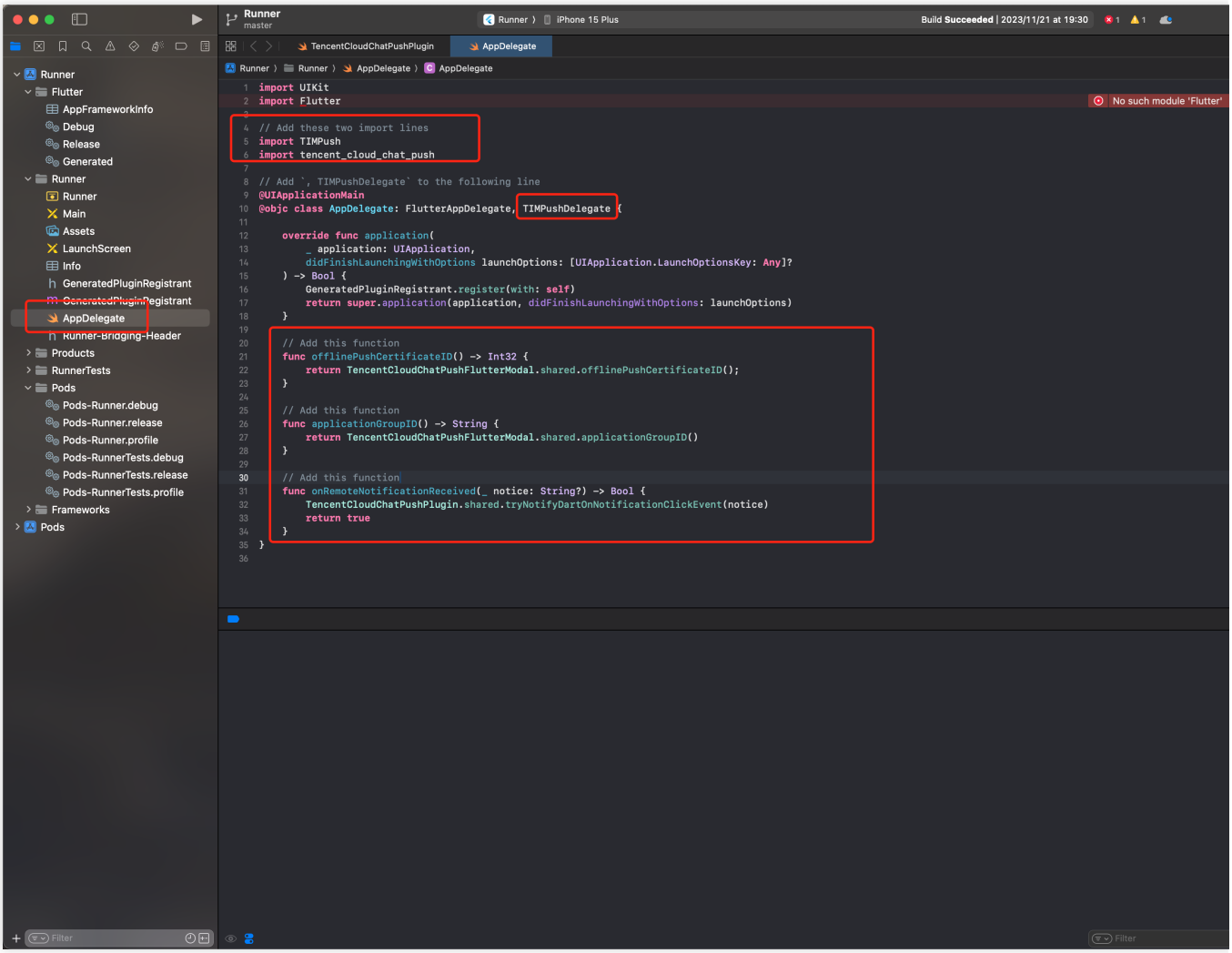
Please don't worry, just follow the instructions and copy the code we provide into the specified file.

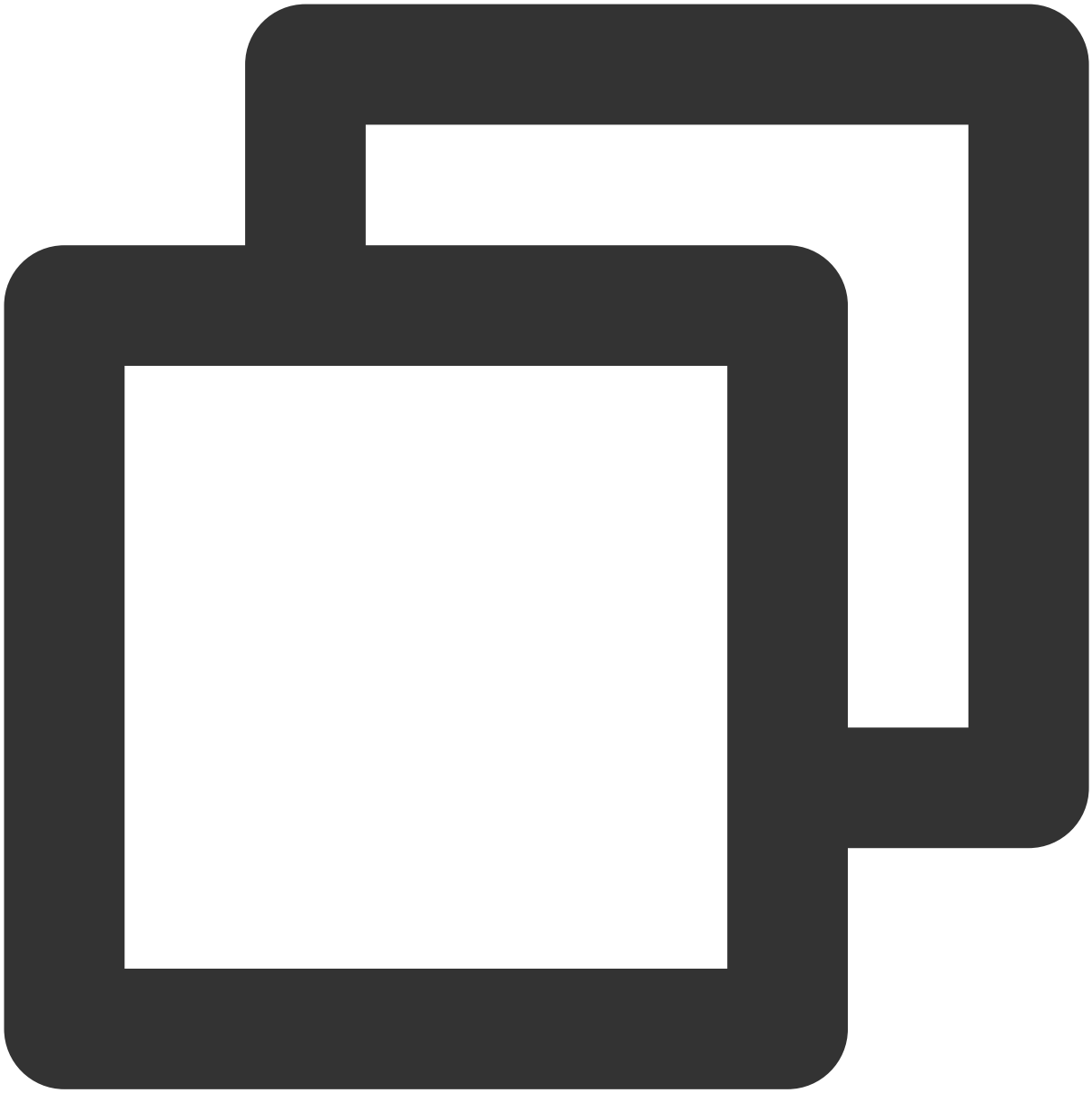
iOS

Android

You can use Xcode for editing, or you can directly edit in Visual Studio Code or Android Studio.

Open the `ios/Runner/AppDelegate.swift` file, paste the following outlined code into it, as shown in the figure. The code is attached after the picture.





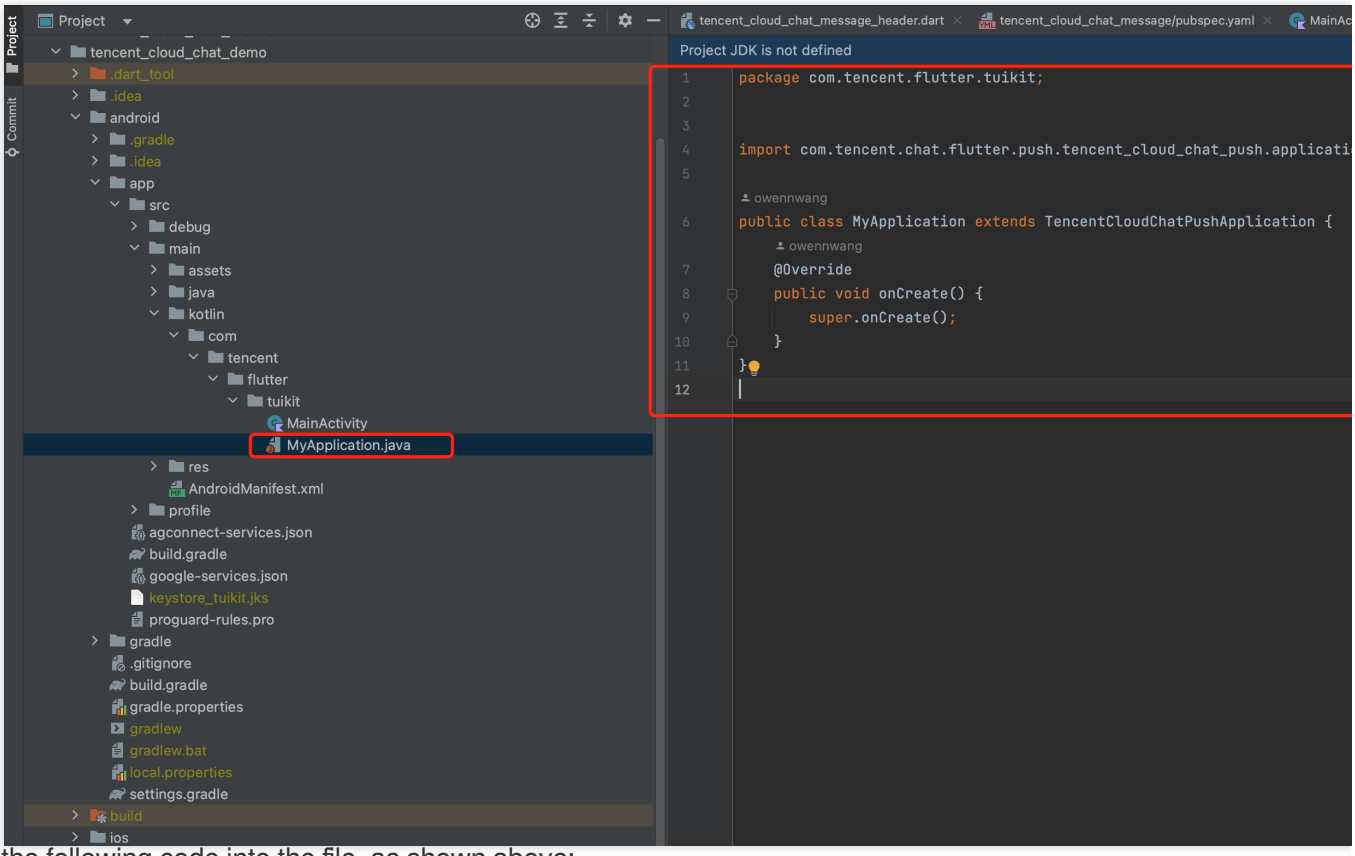
```
import UIKit
import Flutter

// Add these two import lines
import TIMPush
import tencent_cloud_chat_push

// Add `, TIMPushDelegate` to the following line
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate, TIMPushDelegate {
    override func application(
```

```
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKe
    ) -> Bool {
        GeneratedPluginRegistrant.register(with: self)
        return super.application(application, didFinishLaunchingWithOptions: launch
    }

    // Add this function
    func offlinePushCertificateID() -> Int32 {
        return TencentCloudChatPushFlutterModal.shared.offlinePushCertificateID();
    }

    // Add this function
    func applicationGroupID() -> String {
        return TencentCloudChatPushFlutterModal.shared.applicationGroupID()
    }

    // Add this function
    func onRemoteNotificationReceived(_ notice: String?) -> Bool {
        TencentCloudChatPushPlugin.shared.tryNotifyDartOnNotificationClickEvent(not
        return true
    }
}
```

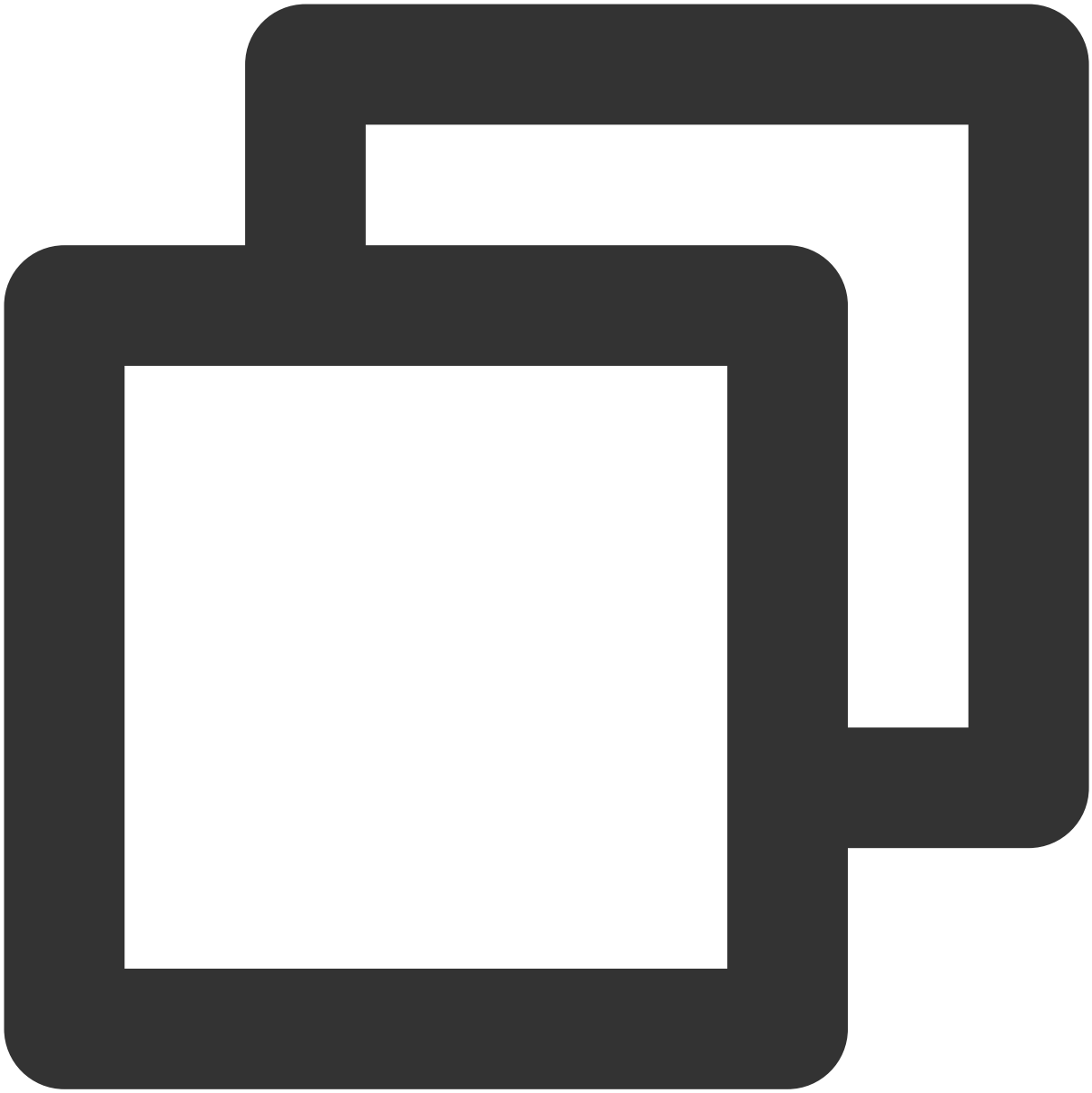
It is recommended to use Android Studio to complete this part of the editing.

In the same directory as `MainActivity` under your project's Android path, create a new Application file class, for example, it can be named `MyApplication.java` .

If you have already defined an Application class, you can directly reuse it without having to create it again.



Paste the following code into the file, as shown above:

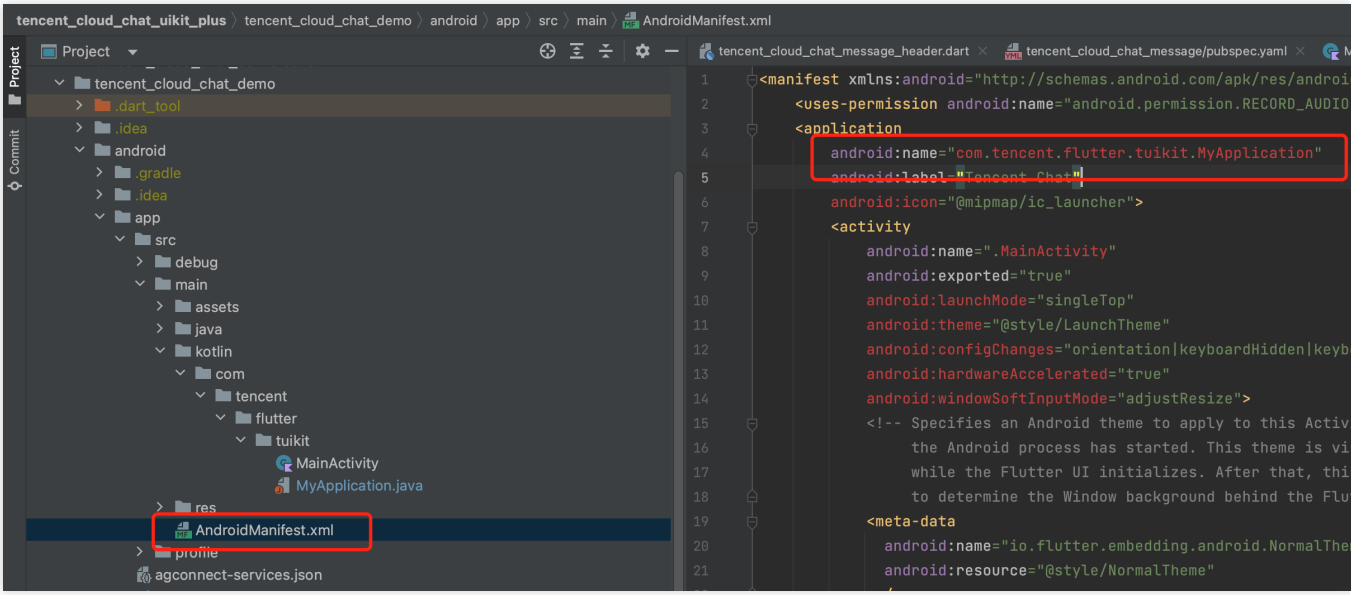


```
Replace 'package' with your own, usually Android Studio will generate it automatica

import com.tencent.chat.flutter.push.tencent_cloud_chat_push.application.TencentClo

public class MyApplication extends TencentCloudChatPushApplication {
    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

Note:

If you have already created your own Application for other purposes, please directly `extend TencentCloudChatPushApplication` and ensure in the `onCreate()` function, you call `super.onCreate();`.

Open the `android/app/src/main/AndroidManifest.xml` file, add a specified `android:name` parameter for the `<application>` tag, pointing to the newly made custom `Application` class. As shown in the figure:



Step 4: Client OEM Configuration

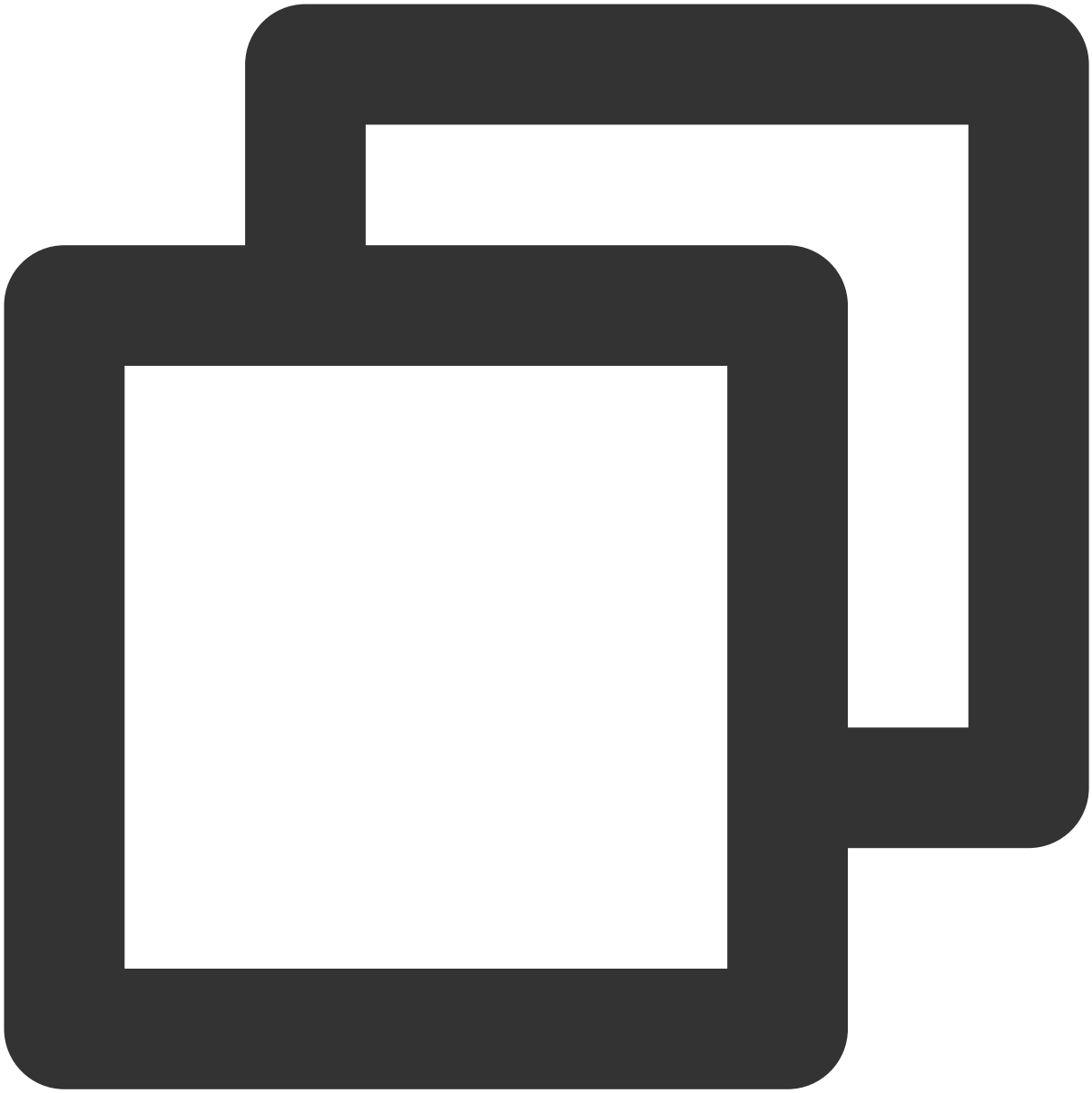
iOS

Android

No need to proceed with this step on the iOS side.

Open the android/app/build.gradle file, at the end, add a dependencies configuration, and as required, include all or part of the following vendor push packages. Only by including the corresponding vendor's push package can you enable that vendor's Native Push Capability.

The version numbers mentioned below should be consistent with the version number of this Flutter Push Plugin (tencent_cloud_chat_push).



```
dependencies {
     // Huawei
     implementation 'com.tencent.timpush:huawei:${Push Plugin version number}'

     // XiaoMi
     implementation 'com.tencent.timpush:xiaomi:${Push Plugin version number}'

     // vivo
     implementation 'com.tencent.timpush:vivo:${Push Plugin version number}'

     // Honor
```

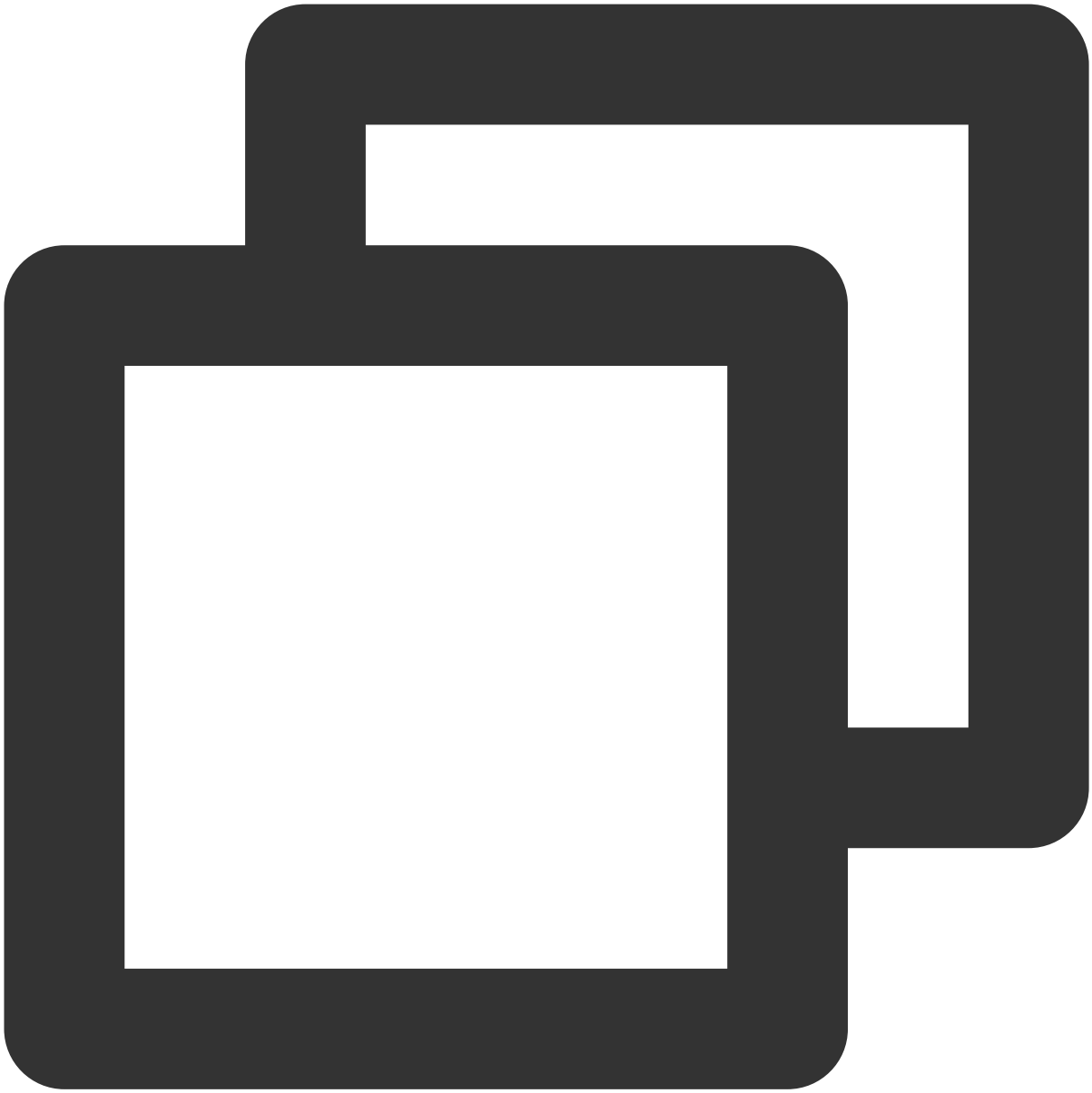
```
    implementation 'com.tencent.timpush:honor:${Push Plugin version number}'

    // Meizu
    implementation 'com.tencent.timpush:meizu:${Push Plugin version number}'

    // Google Firebase Cloud Messaging (Google FCM)
    implementation 'com.tencent.timpush:fcm:${Push Plugin version number}'

    // Choose one of the two below for OPPO
    // For the China Region, choose to integrate this package
    implementation 'com.tencent.timpush:oppo:${Push Plugin version number}'
    // For other regions, choose to integrate this package
    implementation 'com.tencent.timpush:oppo-intl:${Push Plugin version number}'
}
```

Vivo and Honor Adaptation

According to Vivo and Honor Vendor Access Guide, it is necessary to add APPID and APPKEY to the Manifest File.

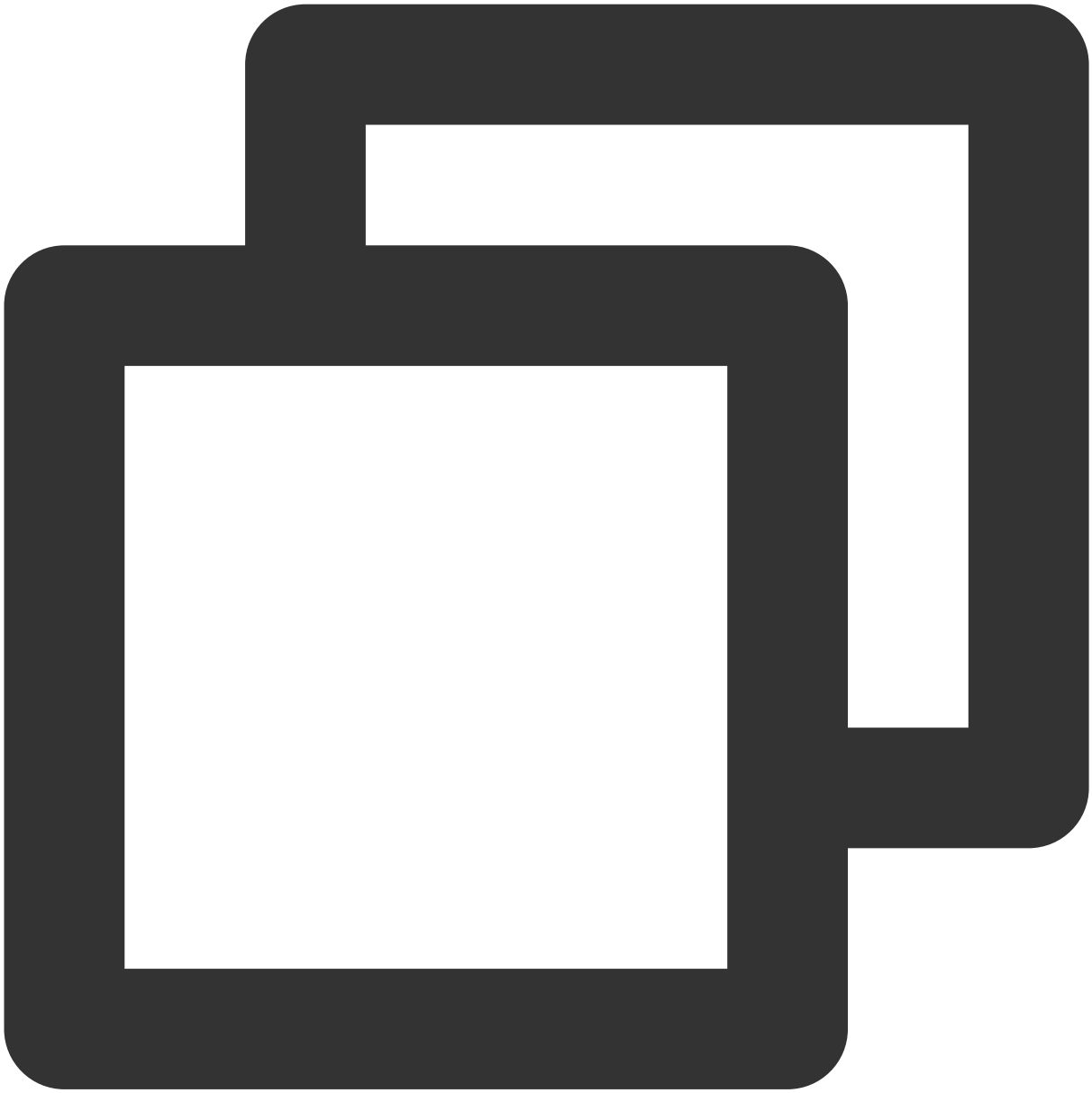
Method 1

Method 2



```
// android/app/build.gradle

android {
    ...
    defaultConfig {
        ...
        manifestPlaceholders = [
            "VIVO_APPKEY" : "`APPKEY` of the certificate assigned to your applicati
            "VIVO_APPID" : "`APPID` of the certificate assigned to your application
            "HONOR_APPID" : "`APPID` of the certificate assigned to your applicatio
        ]
```

```
    }
}
```



```
// android/app/src/main/AndroidManifest.xml

// Vivo begin
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.api_key"
    android:value="`APPKEY` of the certificate assigned to your application" />
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.app_id"
```

```
    android:value="`APPID` of the certificate assigned to your application" />
// Vivo end

// Honor begin
<meta-data tools:replace="android:value"
    android:name="com.hihonor.push.app_id"
    android:value="`APPID` of the certificate assigned to your application" />
// Honor end
```

Adapting to Huawei, Honor, and Google FCM

Follow the vendor's method to integrate the corresponding plugin and JSON configuration file.

Note:

The following adaptation for Honor is only needed for version 7.7.5283 and above.

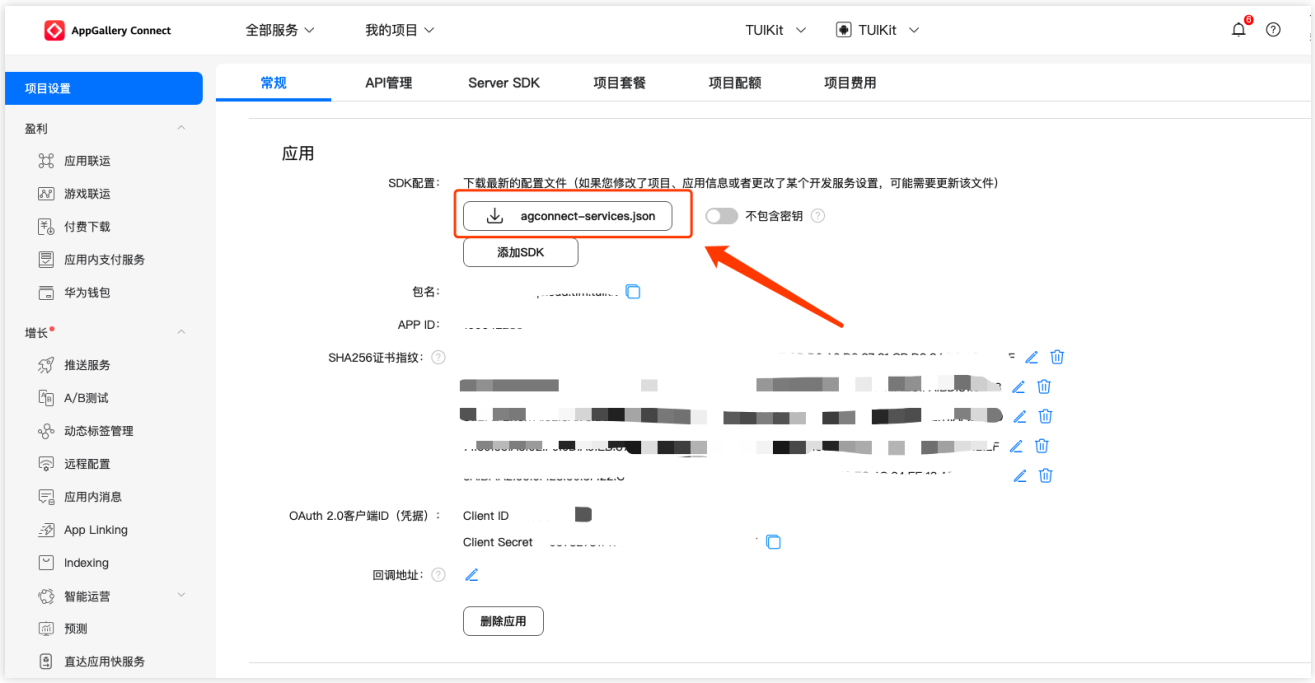
1.1 Download the configuration file and add it to the root directory of the project/Android/app.

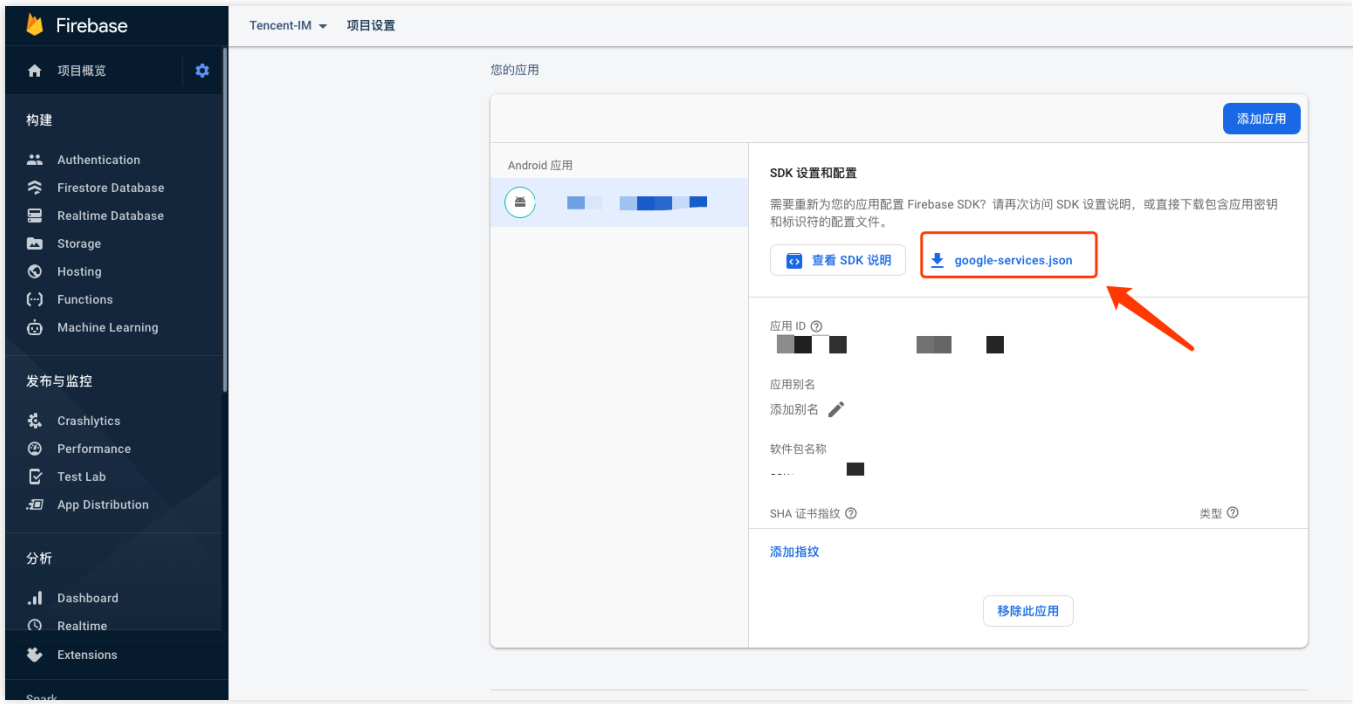
Huawei

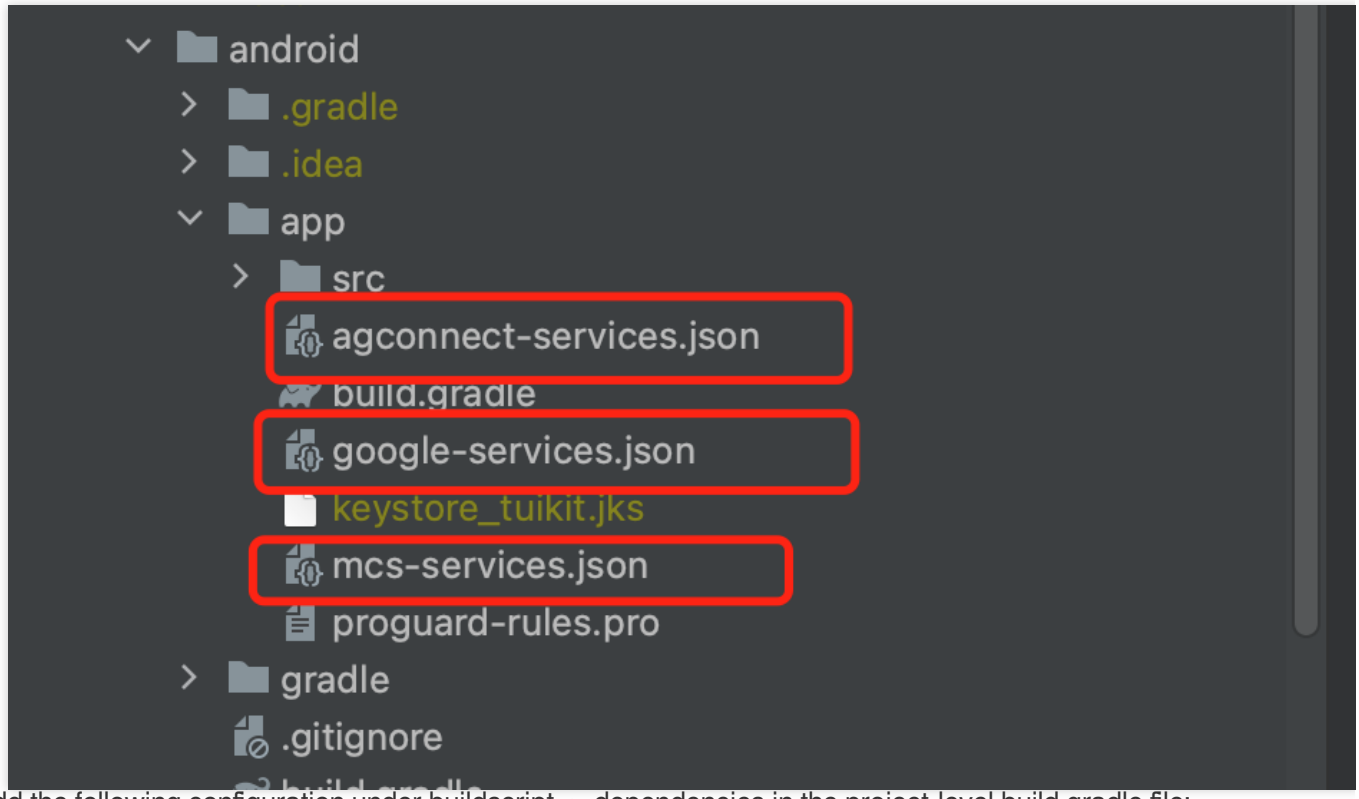
HONOR

Google FCM

Operation Path



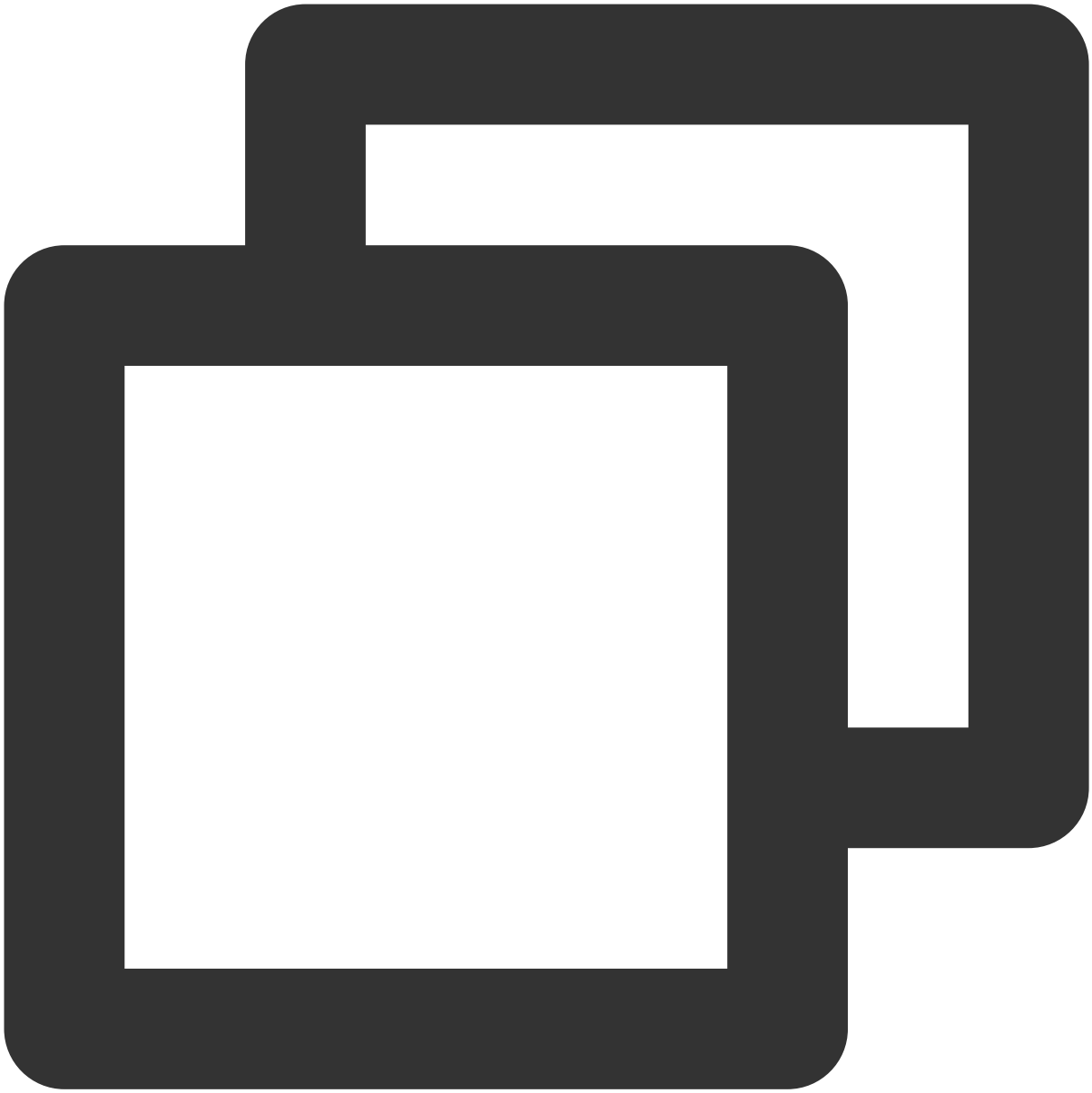




1.2 Add the following configuration under buildscript -> dependencies in the project-level build.gradle file:

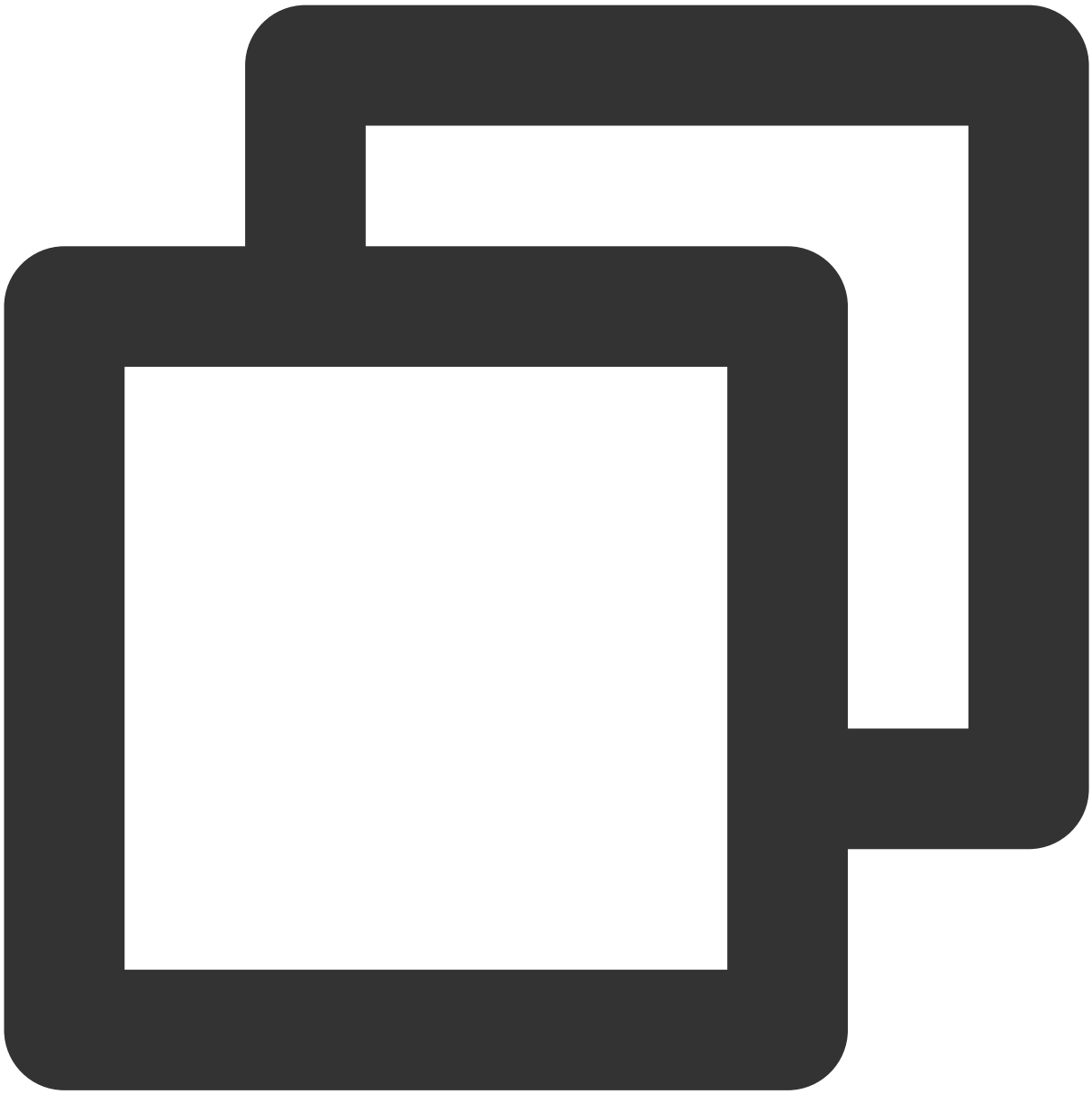
Gradle 7.1 and above

Add the following configuration under buildscript -> dependencies in the project-level build.gradle file:



```
buildscript {
    dependencies {
        ...
        classpath 'com.huawei.agconnect:agcp:1.6.0.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
        classpath 'com.google.gms:google-services:4.4.0'
    }
}
```

Add the following repository configuration under buildscript -> repositories and allprojects -> repositories in the project-level settings.gradle file:



```
pluginManagementbuildscript {
    repositories {
        gradlePluginPortal()
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configure the Maven repository address for the HMS Core SDK.
```

Step 5: Handle the message click callback, and parse the parameters

Please define a function to receive the push message click callback event.

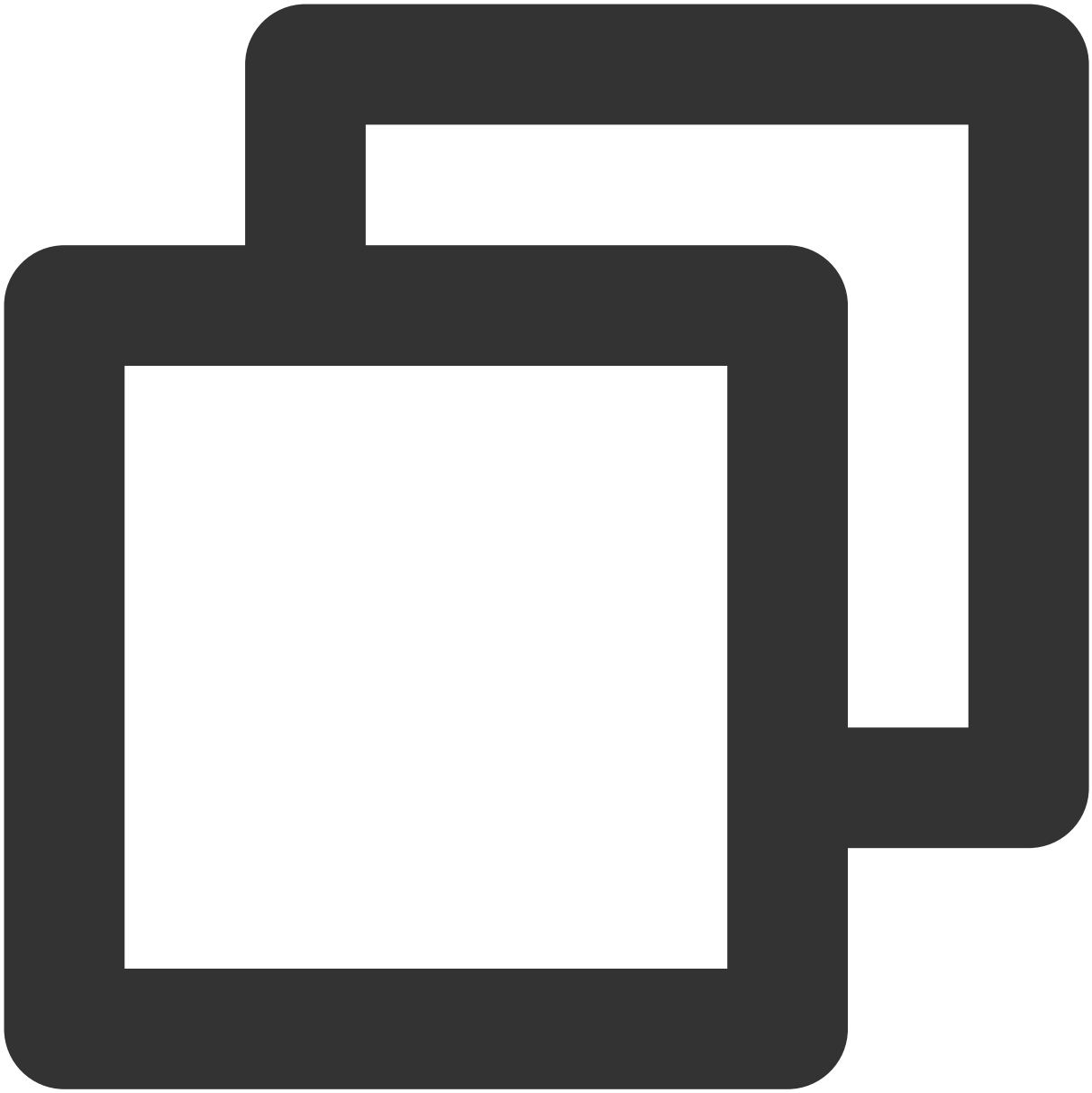
Define the function with the following parameters: `{required String ext, String? userID, String? groupID}` .

Among them, the ext Field carries the complete ext information specified by the sender. If not specified, a default value is assigned. You can navigate to the corresponding page by parsing this field.

The userID and groupID fields are automatically attempted by the plugin to parse the ext Json String, retrieving the single chat partner userID and group chat groupID information. If you have not defined the ext Field yourself, and the ext Field is default specified by the SDK or UIKit, then you can use the default parsing provided here. If parsing fails, it will be null.

You can define a function to receive this callback and navigate to the corresponding session page or your business page accordingly.

Example below:



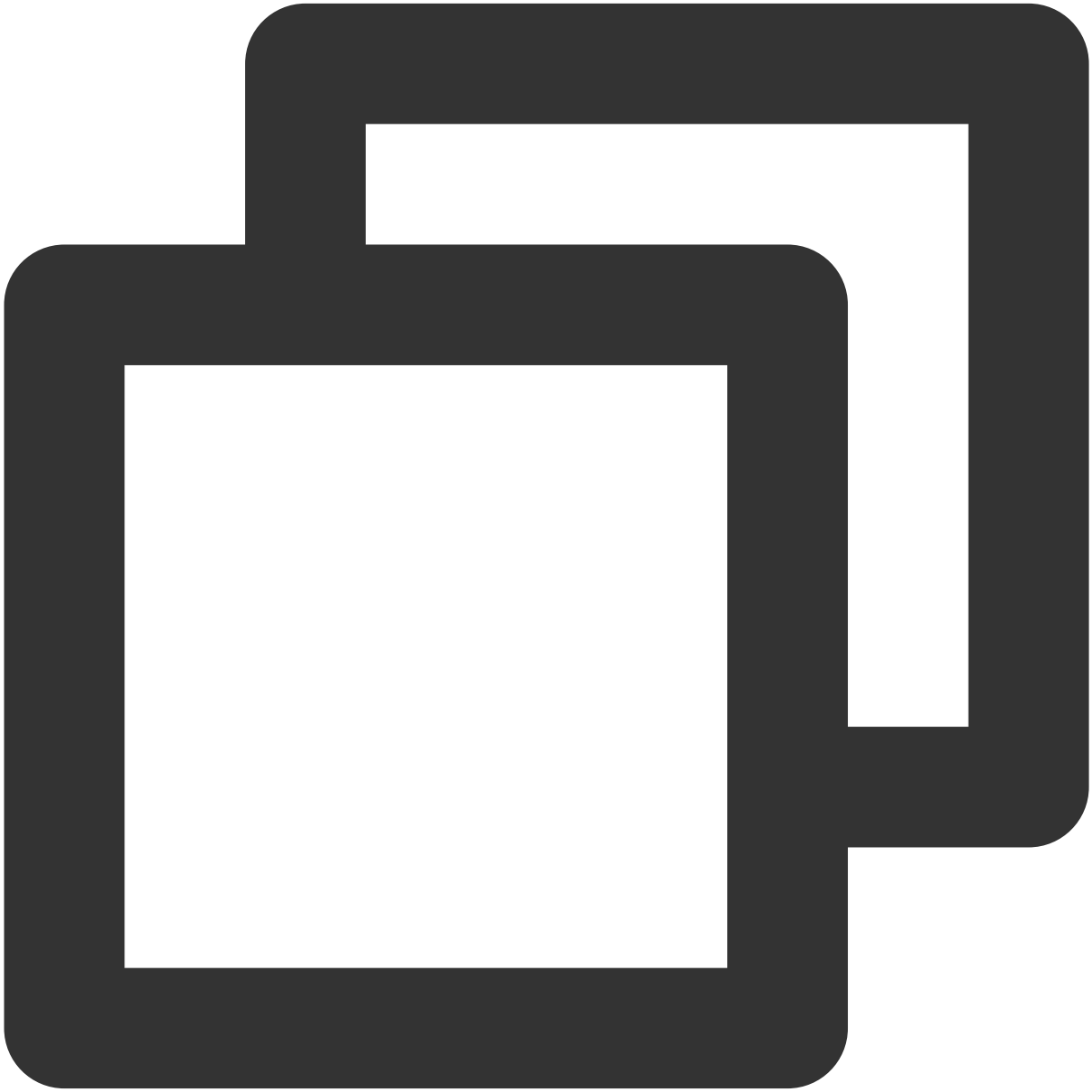
```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
  if (userID != null || groupID != null) {
    // Navigate to the corresponding Message Page based on userID or groupID.
  } else {
    // Write your own parsing method based on the ext Field and navigate to the cor
  }
}
```

Step 6: Register Push Plugin

Please register the push plugin immediately after logging into IM and before using other plugins (such as CallKit).

Invoke the `TencentCloudChatPush().registerPush` method, passing in a callback function defined for clicks.

Furthermore, you have the option to also pass in `apnsCertificateID` for the iOS push certificate ID and `androidPushOEMConfig` for the Android push vendor configuration. These two configurations should have been specified in previous steps, and if no modification is necessary, they do not need to be passed again.

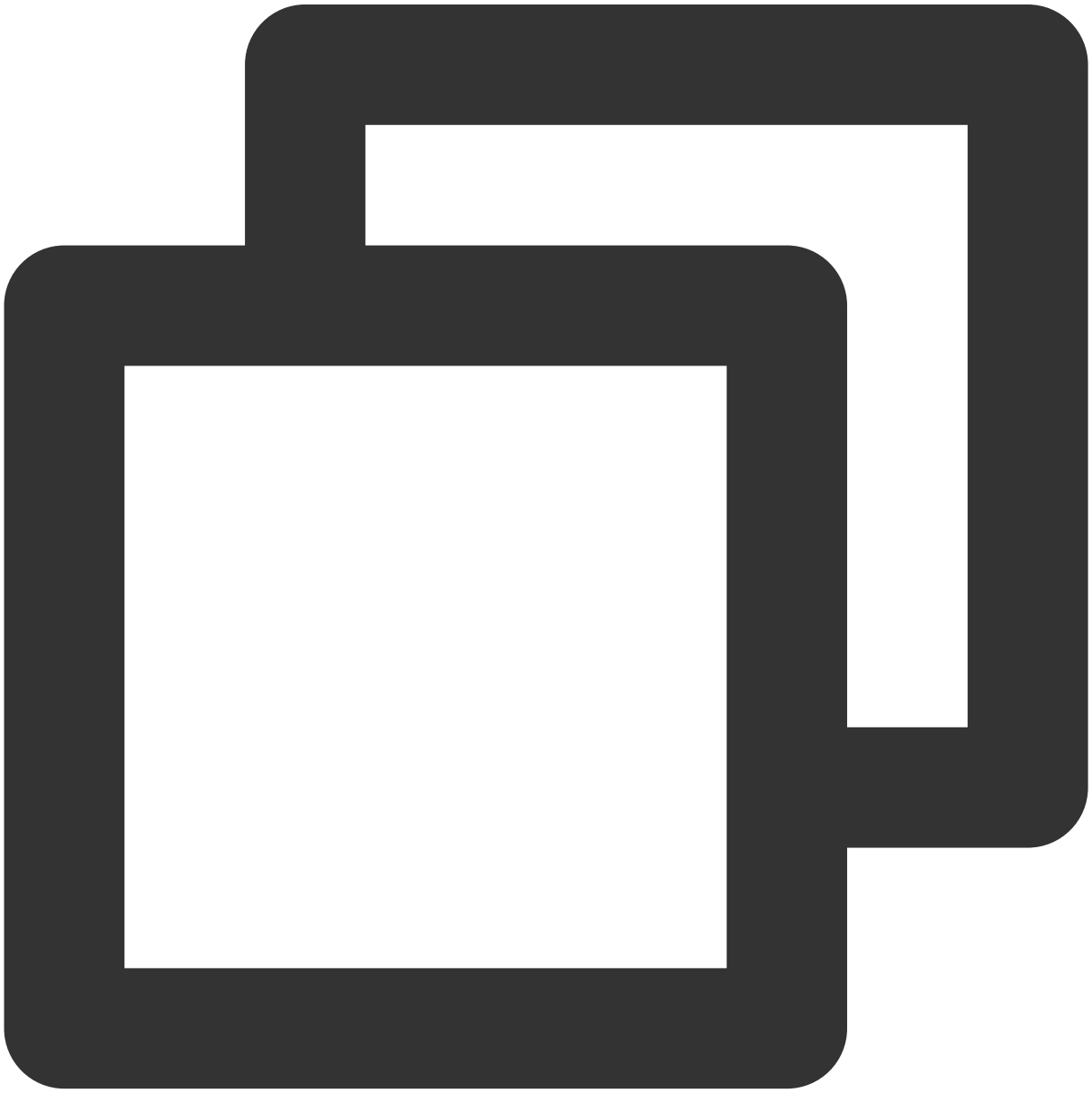


```
TencentCloudChatPush().registerPush(onNotificationClicked: _onNotificationClicked);
```

Note:

If your application requires the use of **push plugin for business message notifications** , and it does not immediately start and log in to the IM module after launching <3>, or if you need to handle business navigation by obtaining message click callbacks before logging in to the IM module <5>, it is recommended that you call the <7>TencentCloudChatPush().registerOnNotificationClickedEvent</7> method as soon as possible to manually mount the message click callback, so that you can promptly obtain the message parameters.

In this scenario, you can execute this function before calling `TencentCloudChatPush().registerPush` and place it as early as possible in the code.



```
TencentCloudChatPush().registerOnNotificationClickedEvent(onNotificationClicked: _o
```

Step 7: Message Push Delivery Statistics

If you need to collect data on delivery, please complete the setup as follows:

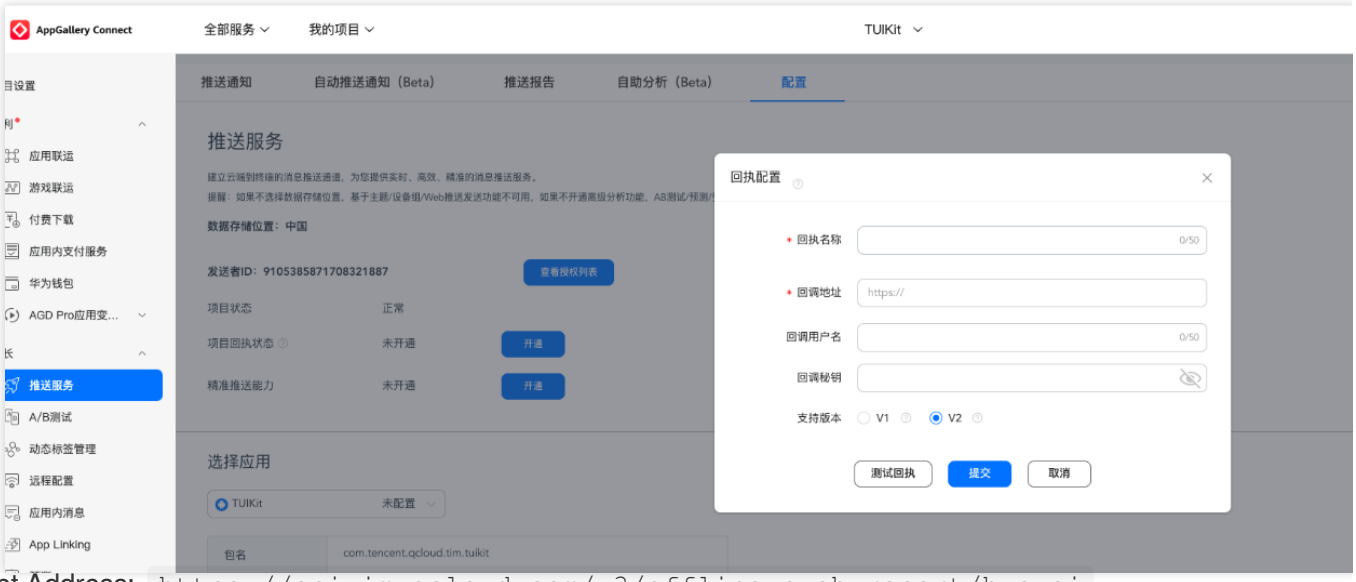
Huawei

HONOR

vivo

Meizu

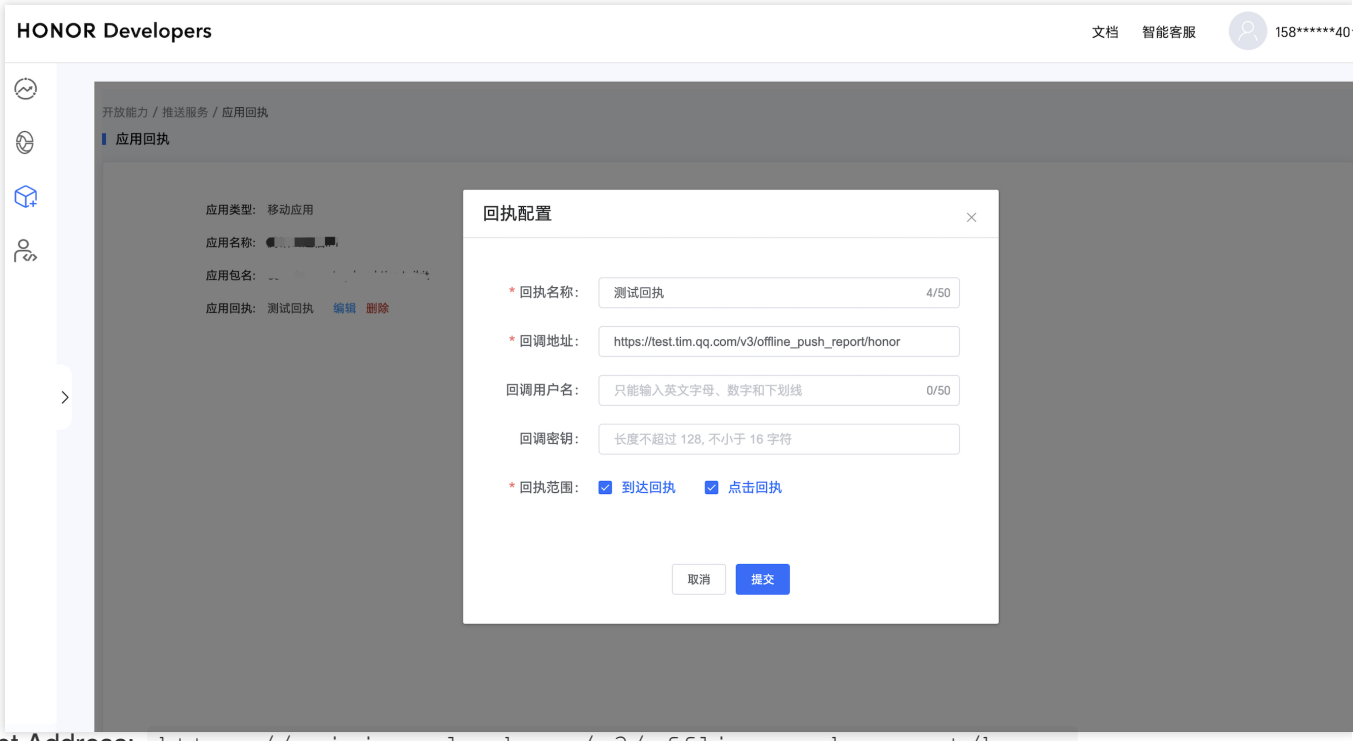
iOS



Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/huawei`

Note:

Huawei Push Certificate ID <= 11344, using Huawei Push v2 version interface does not support reach and click receipt, please regenerate and update the certificate ID.



Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/honor`

Callback Address Configuration	Receipt ID Configuration i
 Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/vivo`	

Enable Receipt Switch	Configure Receipt Addres

Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/meizu`

Note:

After enabling the Receipt Switch, please make sure the Receipt Address is configured correctly. Failing to configure or configuring the wrong address will affect the push feature.

For iOS Push Notification Reach Statistics Configuration, please refer to **Statistics Push Arrival Rate**.

No configuration is needed for other supported manufacturers; FCM does not support the push notification statistics feature.

Congratulations, you have completed the integration of the push plugin. A reminder: After the trial or purchase of the push plugin expires, the push service will automatically stop (including regular message offline push, all-member/Tag push, etc.). To avoid affecting the normal use of your business, please purchase/ renew in advance.

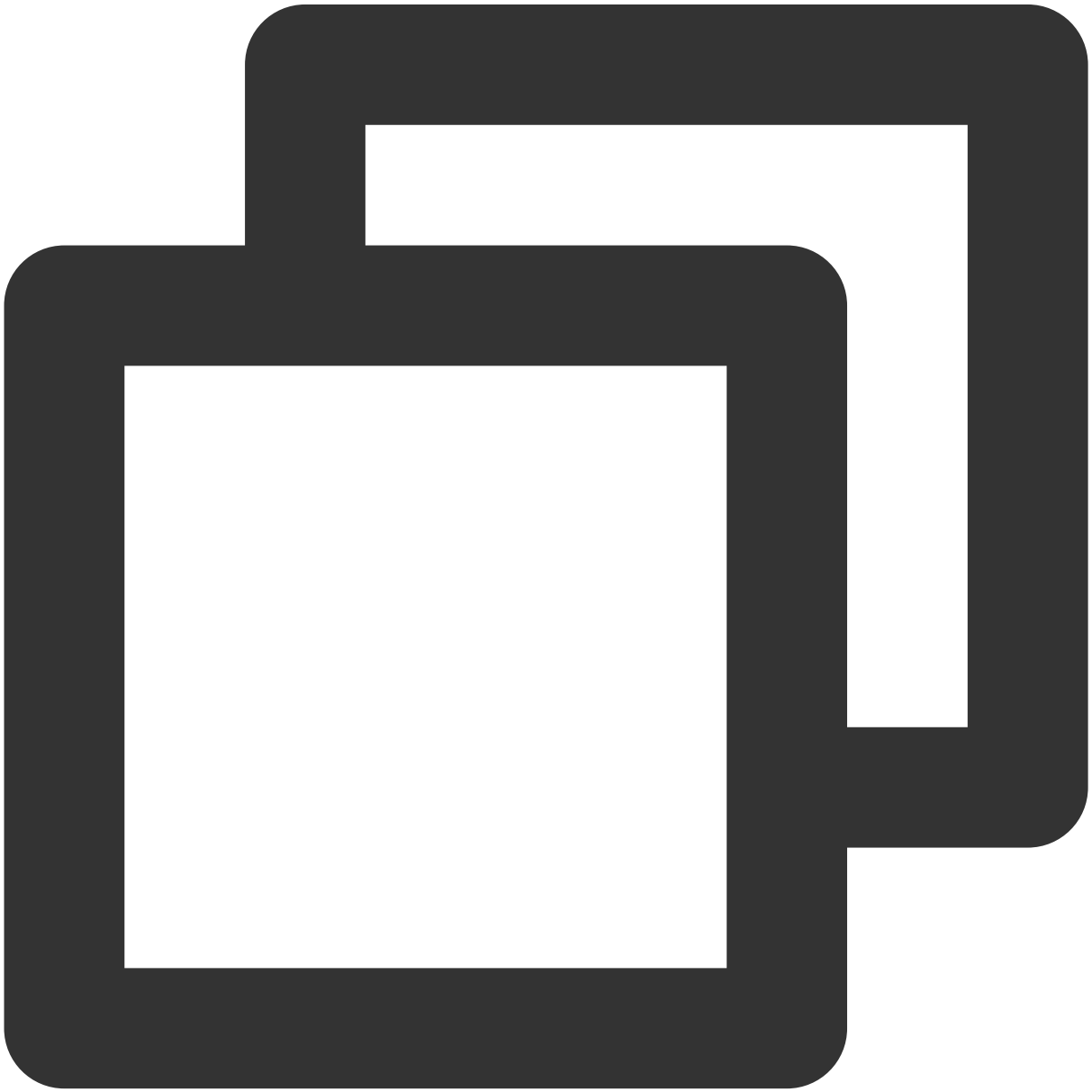
React-Native

Last updated：2024-06-24 15:47:33

Operation step

Step 1: Integrate message push plugin

To install the `react-native-tim-push` package, execute the following code in the terminal.



```
# use yarn
yarn add react-native-tim-push
# use npm
npm install react-native-tim-push
```

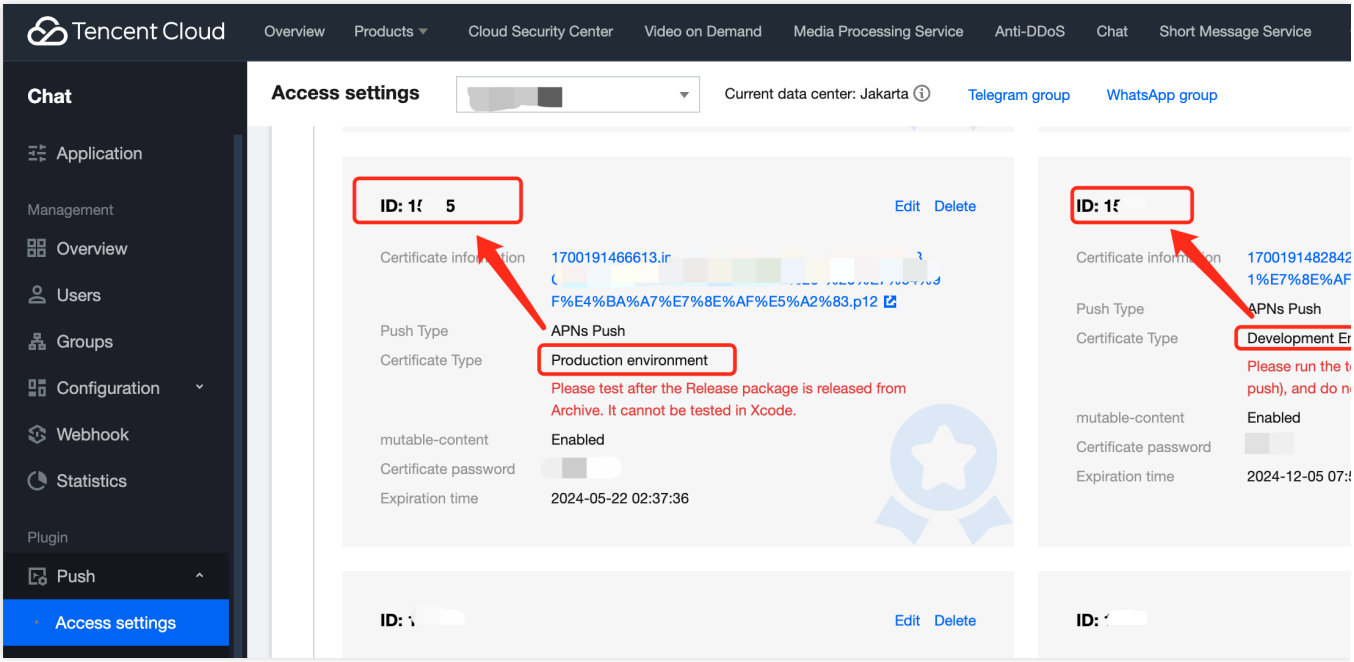
Step 2: Push parameter configuration

iOS

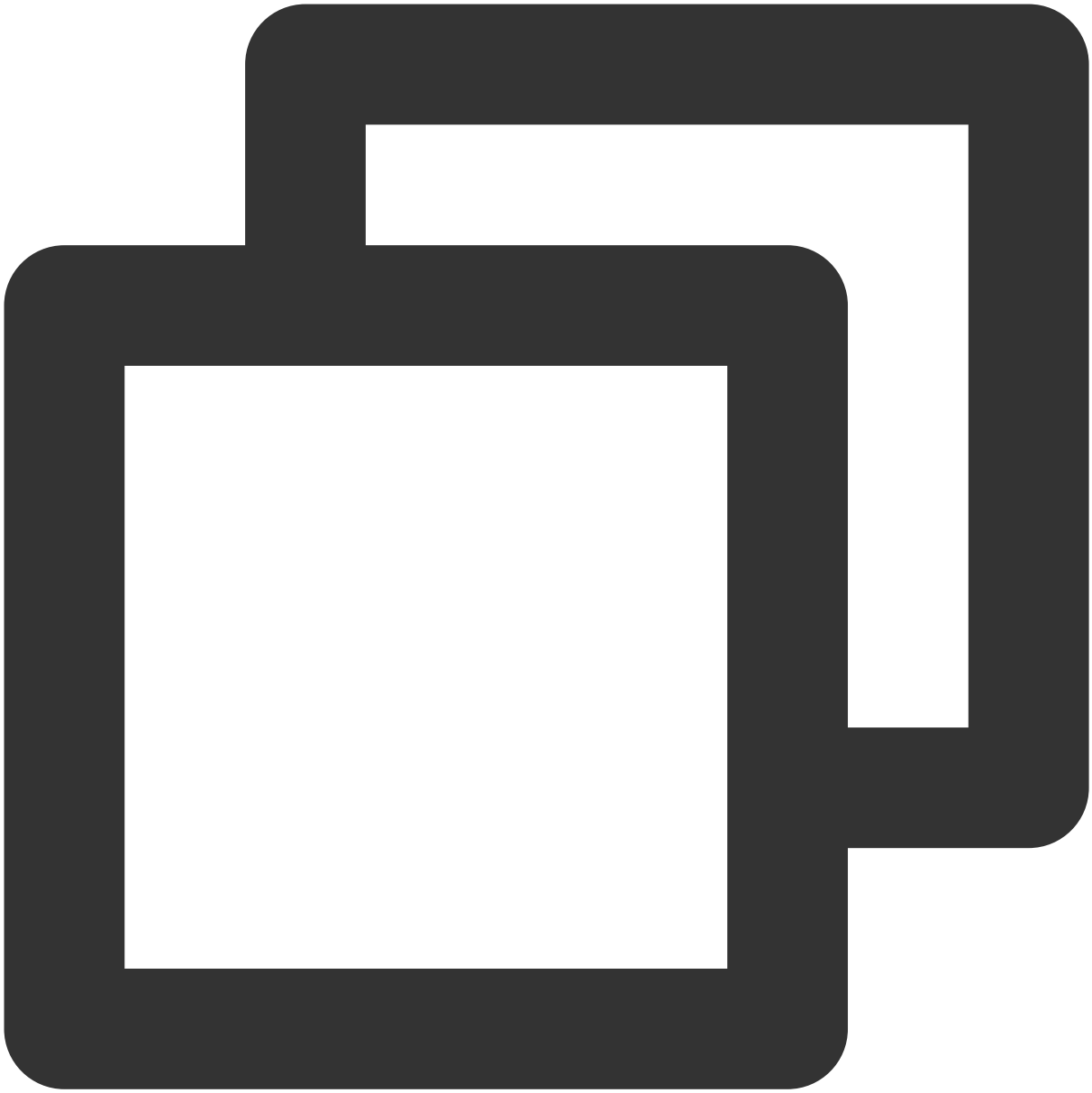
Android

Please upload the iOS APNs push certificate obtained during the vendor configuration step to the IM console.

The IM console will assign a certificate ID for you, as shown in the figure below:



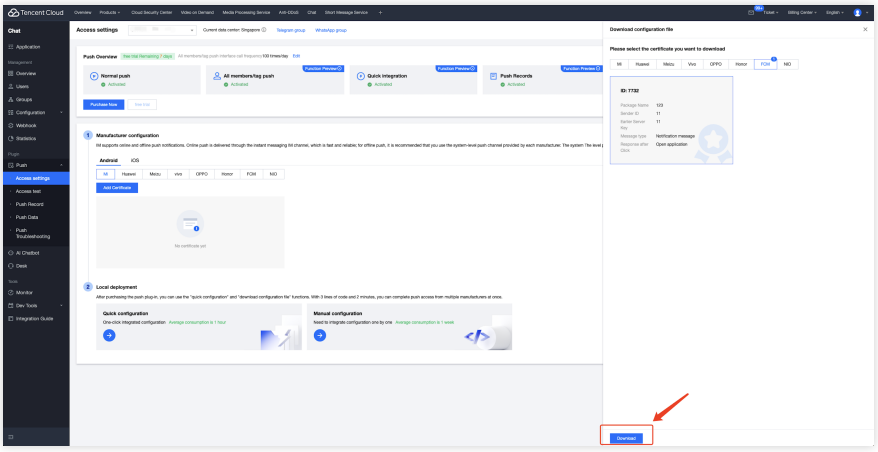
As early as possible after your application starts, call the `TimPushPlugin.getInstance().setApnsCertificateID` method to pass in the certificate ID.

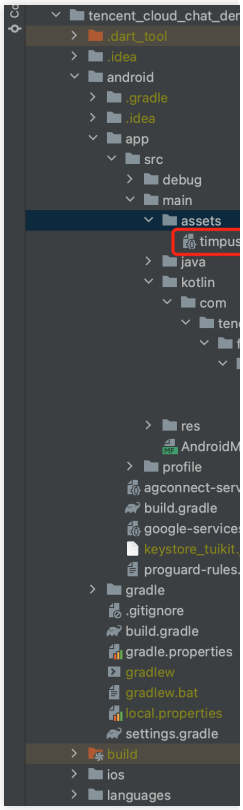


```
import { TimPushPlugin } from 'react-native-tim-push';
const certificateID = 'certificate id';
TimPushPlugin.getInstance().setApnsCertificateID(certificateID);
```

After completing the manufacturer push information on the console, download and add the configuration file to the project. Add the downloaded timpush-configs.json file to the `android/app/src/main/assets` directory. If the directory does not exist, please create it manually.

1. Choose to download the configuration file timpush-configs.json	1. Add to the project





Step 3: Client Code Configuration

In this step, you'll need to write some native code, such as: Swift, Java, etc.

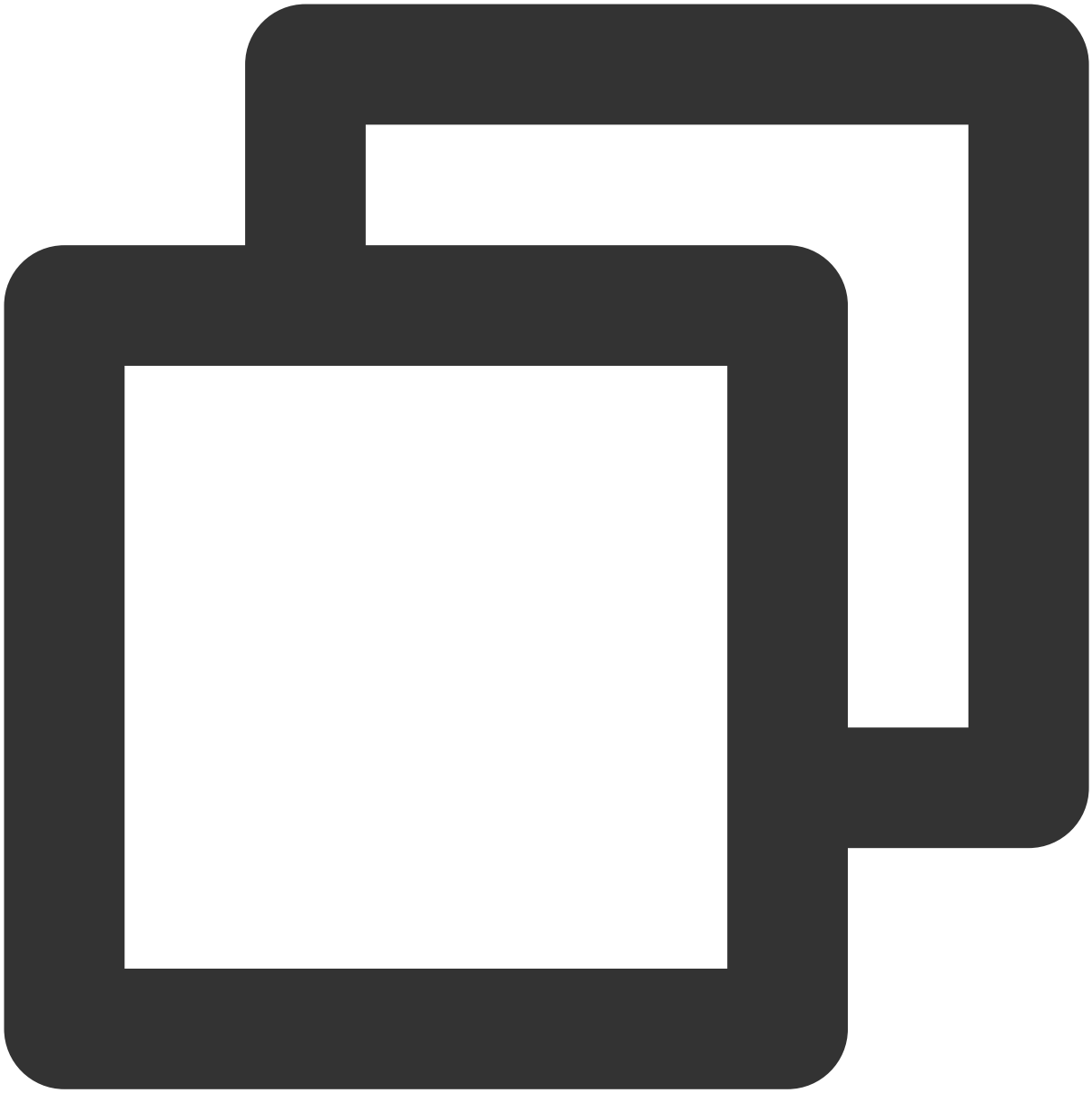
Please don't worry, just follow the instructions and copy the code we provide into the specified file.

iOS

Android

You can use Xcode for editing, or you can directly edit in Visual Studio Code or Android Studio.

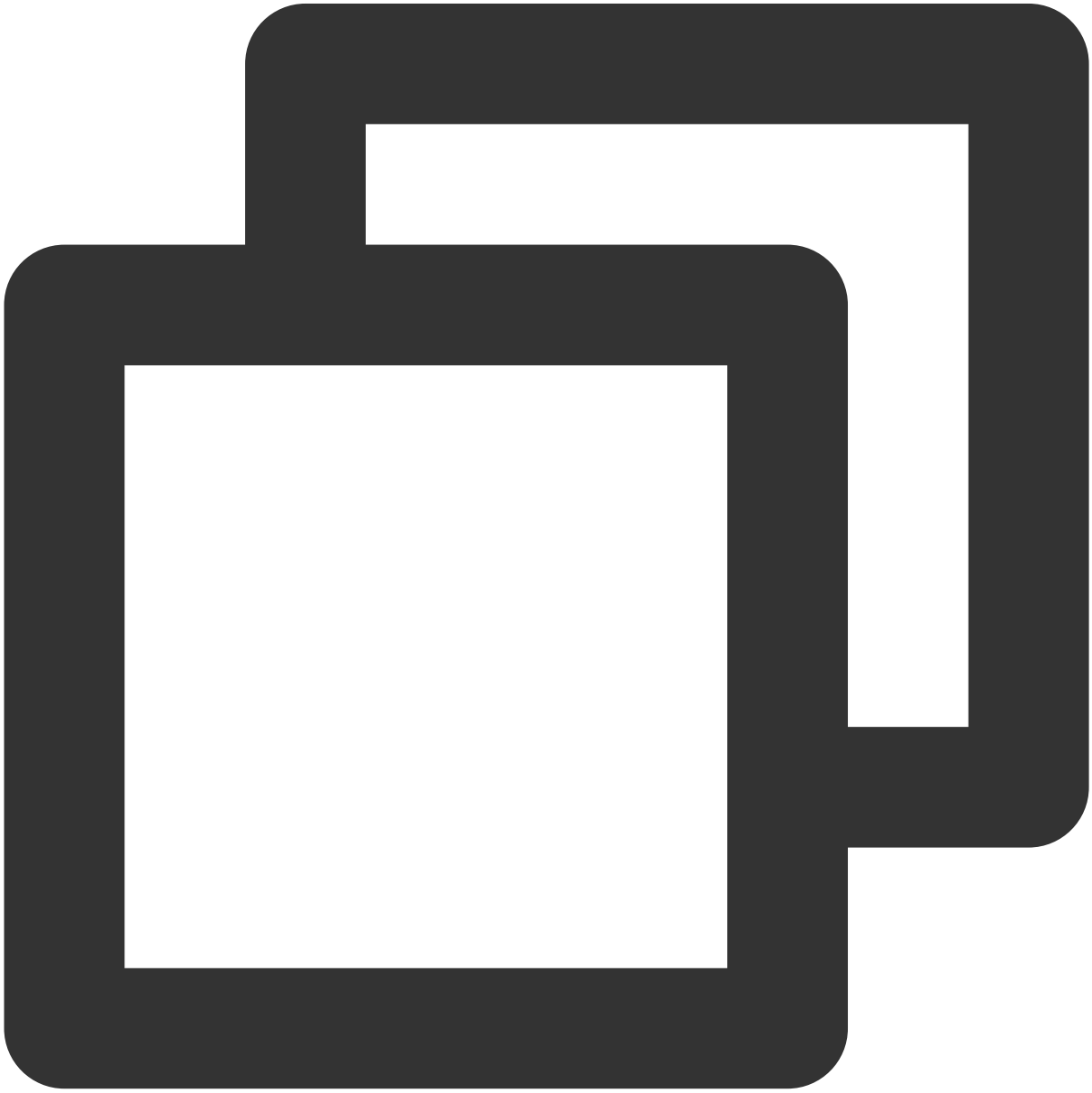
1. Create a file named "TencentIMPush.swift" in the iOS directory and copy the following code into it.



```
import Foundation
import react_native_tim_push


@objc class TencentImPush: NSObject{


  @objc func getOfflinePushCertificatedID() -> Int32 {
    return TencentCloudPushModal.shared.offlinePushCertificateID();
  }
```

```
    @objc func getApplicationGroupID() -> String {
      return TencentCloudPushModal.shared.applicationGroupID();
    }

    @objc func onRemoteNotificationReceived(_ notice: String?) -> Void {
      TencentCloudPushModal.shared.onRemoteNotificationReceived(notice);
    }
}
```

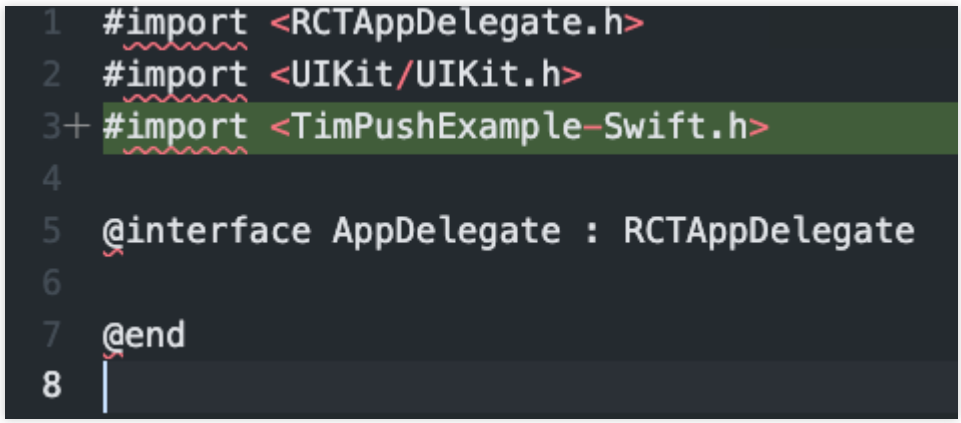
2. Locate the "AppDelegate.h" file and add the following code to it.



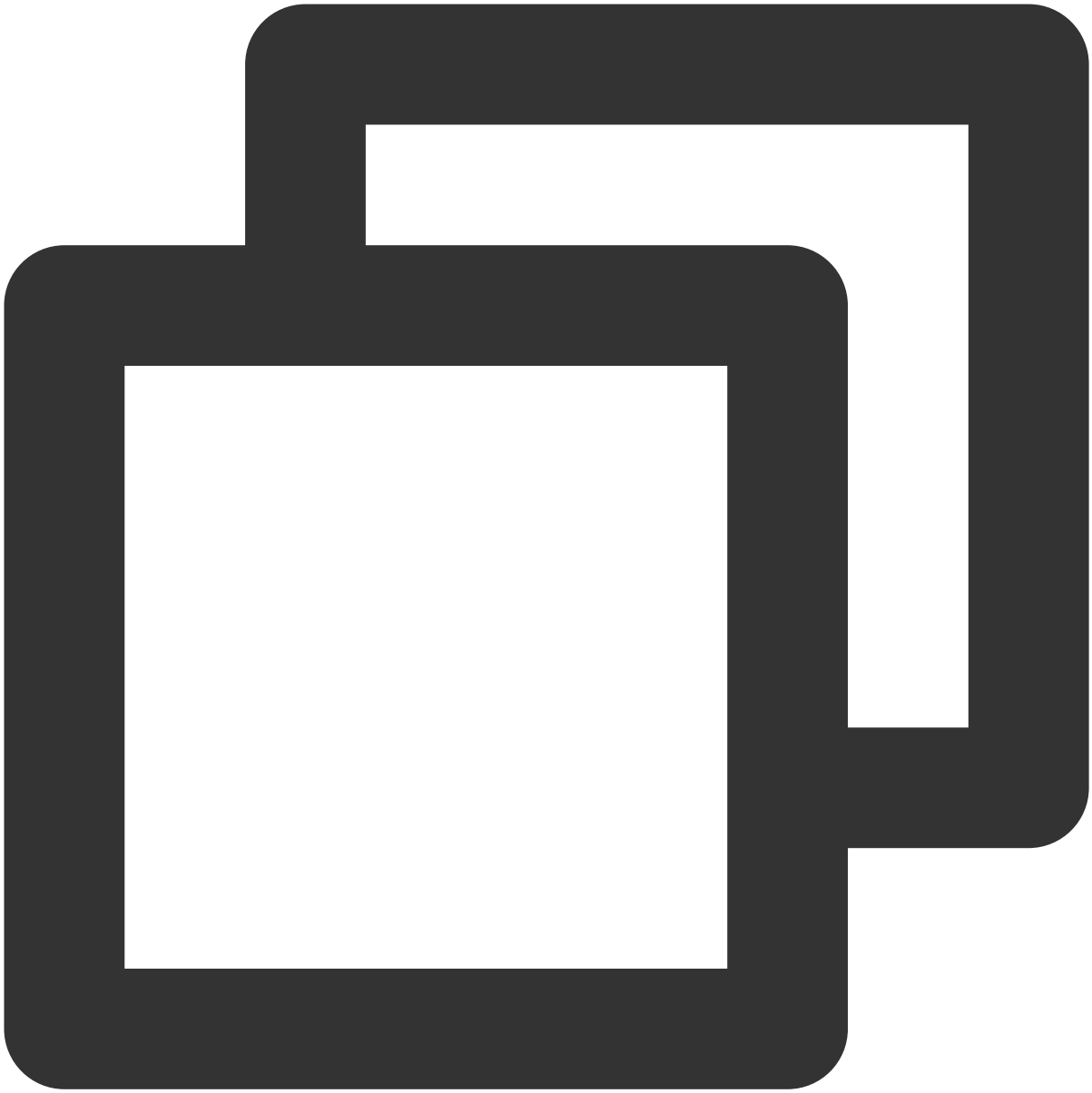
```
...
```

```
#import <Your-Project-Name-Swift.h>
// My project Name is `TimPushExample`. So it should be `#import <TimPushExample-Sw
...
```

After adding the code, it should look like this:



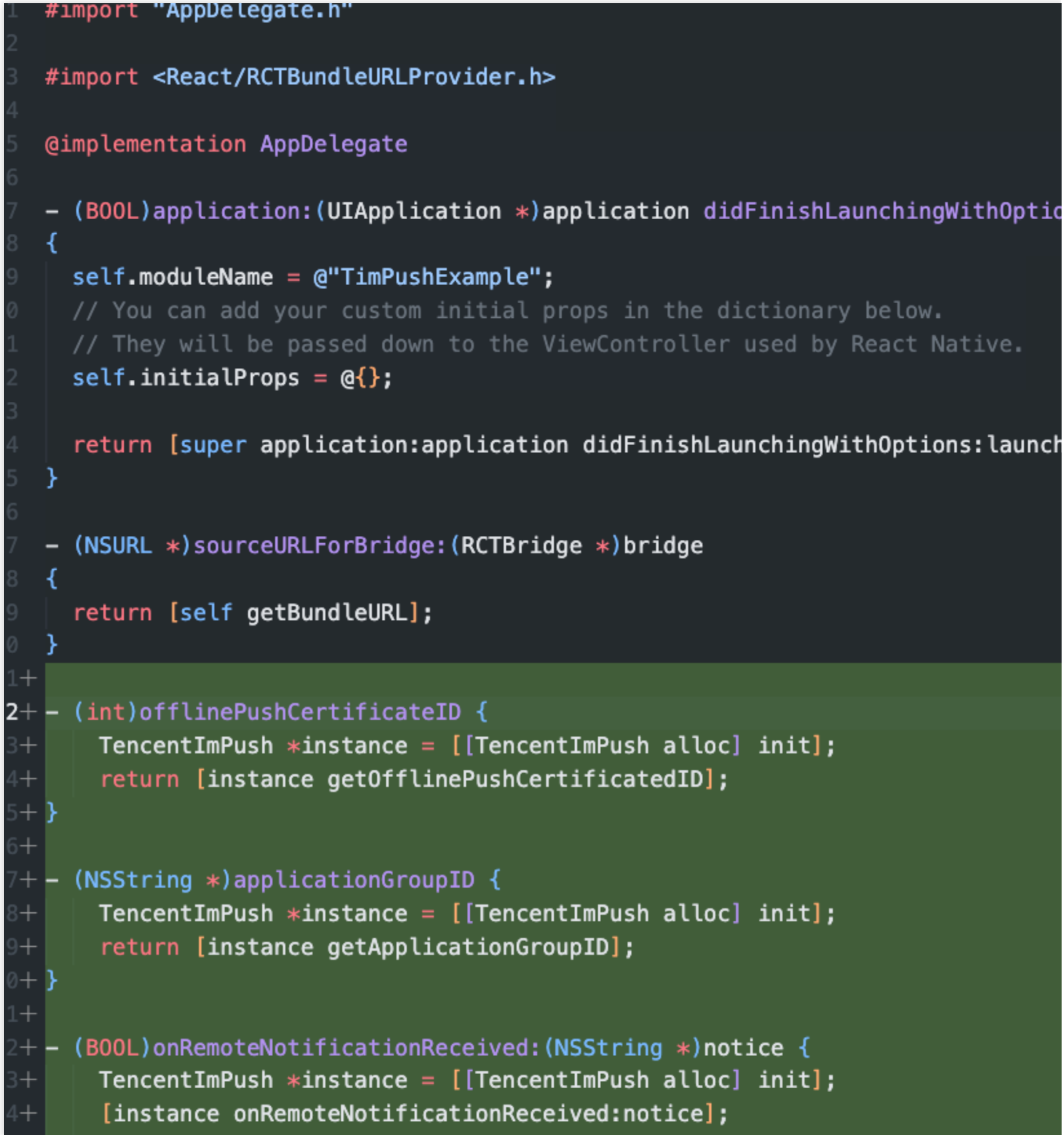
3. Locate the "AppDelegate.mm" file and add the following code to it.



```
- (int)offlinePushCertificateID {
    TencentImPush *instance = [[TencentImPush alloc] init];
    return [instance getOfflinePushCertificatedID];
}


- (NSString *)applicationGroupID {
    TencentImPush *instance = [[TencentImPush alloc] init];
    return [instance getApplicationGroupID];
}
```

```
- (BOOL)onRemoteNotificationReceived:(NSString *)notice {
    TencentImPush *instance = [[TencentImPush alloc] init];
    [instance onRemoteNotificationReceived:notice];
    return YES;
}
```

After adding the code, it should look like this:



```
#import "AppDelegate.h"

#import <React/RCTBundleURLProvider.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptic
{
  self.moduleName = @"TimPushExample";
  // You can add your custom initial props in the dictionary below.
  // They will be passed down to the ViewController used by React Native.
  self.initialProps = @{};

  return [super application:application didFinishLaunchingWithOptions:launch
}

- (NSURL *)sourceURLForBridge:(RCTBridge *)bridge
{
  return [self getBundleURL];
}

+
+ - (int)offlinePushCertificateID {
+     TencentImPush *instance = [[TencentImPush alloc] init];
+     return [instance getOfflinePushCertificatedID];
+ }
+
+ - (NSString *)applicationGroupID {
+     TencentImPush *instance = [[TencentImPush alloc] init];
+     return [instance getApplicationGroupID];
+ }
+
+ - (BOOL)onRemoteNotificationReceived:(NSString *)notice {
+     TencentImPush *instance = [[TencentImPush alloc] init];
+     [instance onRemoteNotificationReceived:notice];
```

```
+    return YES;
+}
+

- (NSURL *)getBundleURL
{
#if DEBUG
  return [[RCTBundleURLProvider sharedSettings] jsBundleURLForBundleRoot:@"i
#else
  return [[NSBundle mainBundle] URLForResource:@"main" withExtension:@"jsbur
#endif
}

@end
```

In the "MainApplication.java" file, import "com.timpush.RNTencentIMPushApplication" and replace "Application" with "RNTencentIMPushApplication" as shown in the following example:

```
import com.facebook.react.flipper.ReactNativeFlipper
import com.facebook.soloader.SoLoader
+import com.timpush.RNTencentIMPushApplication

-class MainApplication : Application(), ReactApplication {
+class MainApplication : RNTencentIMPushApplication(), React

  override val reactNativeHost: ReactNativeHost =
      object : DefaultReactNativeHost(this) {
        override fun getPackages(): List<ReactPackage> =
            PackageList(this).packages.apply {
              // Packages that cannot be autolinked yet can
              // add(MyReactNativePackage())
            }
```

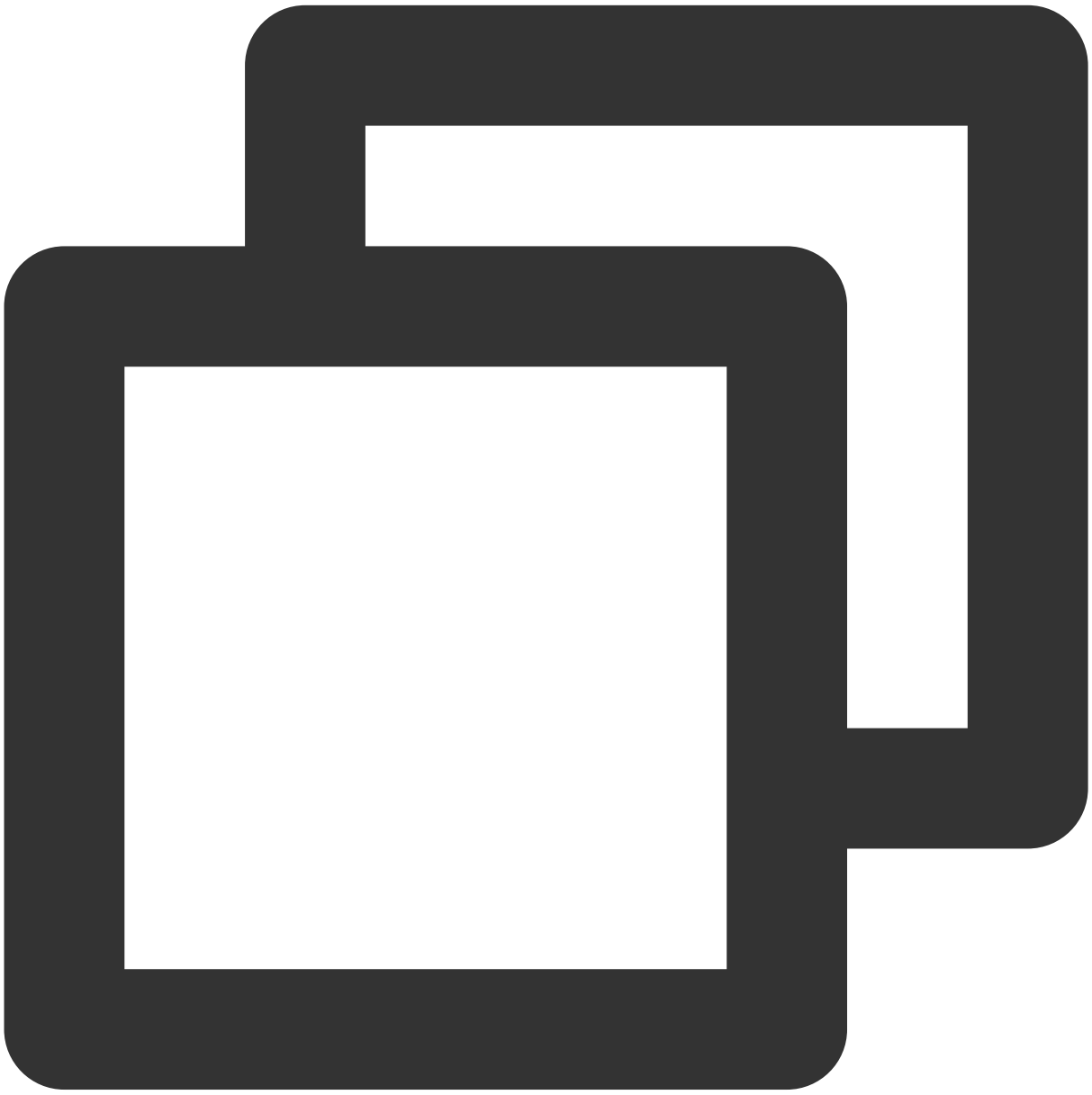
Step 4: Client OEM Configuration

iOS

Android

No need to proceed with this step on the iOS side.

Open the android/app/build.gradle file, at the end, add a dependencies configuration, and as required, include all or part of the following vendor push packages. Only by including the corresponding vendor's push package can you enable that vendor's Native Push Capability.
The version numbers mentioned below should be consistent with the version number of this React Native Push Plugin (react-native-tim-push).



```
dependencies {
     // Huawei
     implementation 'com.tencent.timpush:huawei:${Push Plugin version number}'
```

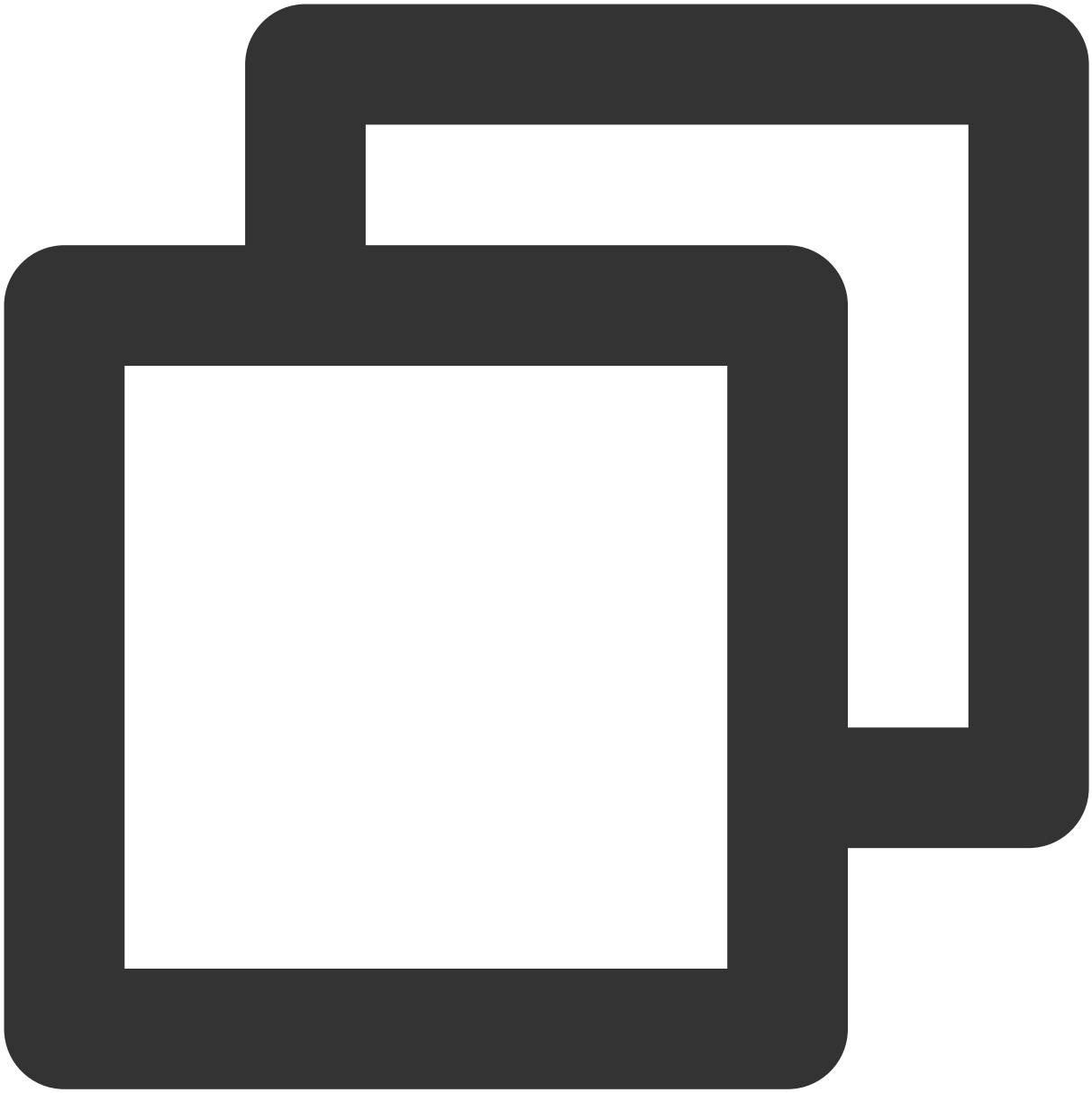
```
    // XiaoMi
    implementation 'com.tencent.timpush:xiaomi:${Push Plugin version number}'

    // vivo
    implementation 'com.tencent.timpush:vivo:${Push Plugin version number}'

    // Honor
    implementation 'com.tencent.timpush:honor:${Push Plugin version number}'

    // Meizu
    implementation 'com.tencent.timpush:meizu:${Push Plugin version number}'

    // Google Firebase Cloud Messaging (Google FCM)
    implementation 'com.tencent.timpush:fcm:${Push Plugin version number}'

    // Choose one of the two below for OPPO
    // For the China Region, choose to integrate this package
    implementation 'com.tencent.timpush:oppo:${Push Plugin version number}'
    // For other regions, choose to integrate this package
    implementation 'com.tencent.timpush:oppo-intl:${Push Plugin version number}'
}
```

Vivo and Honor Adaptation

According to Vivo and Honor Vendor Access Guide, it is necessary to add APPID and APPKEY to the Manifest File.

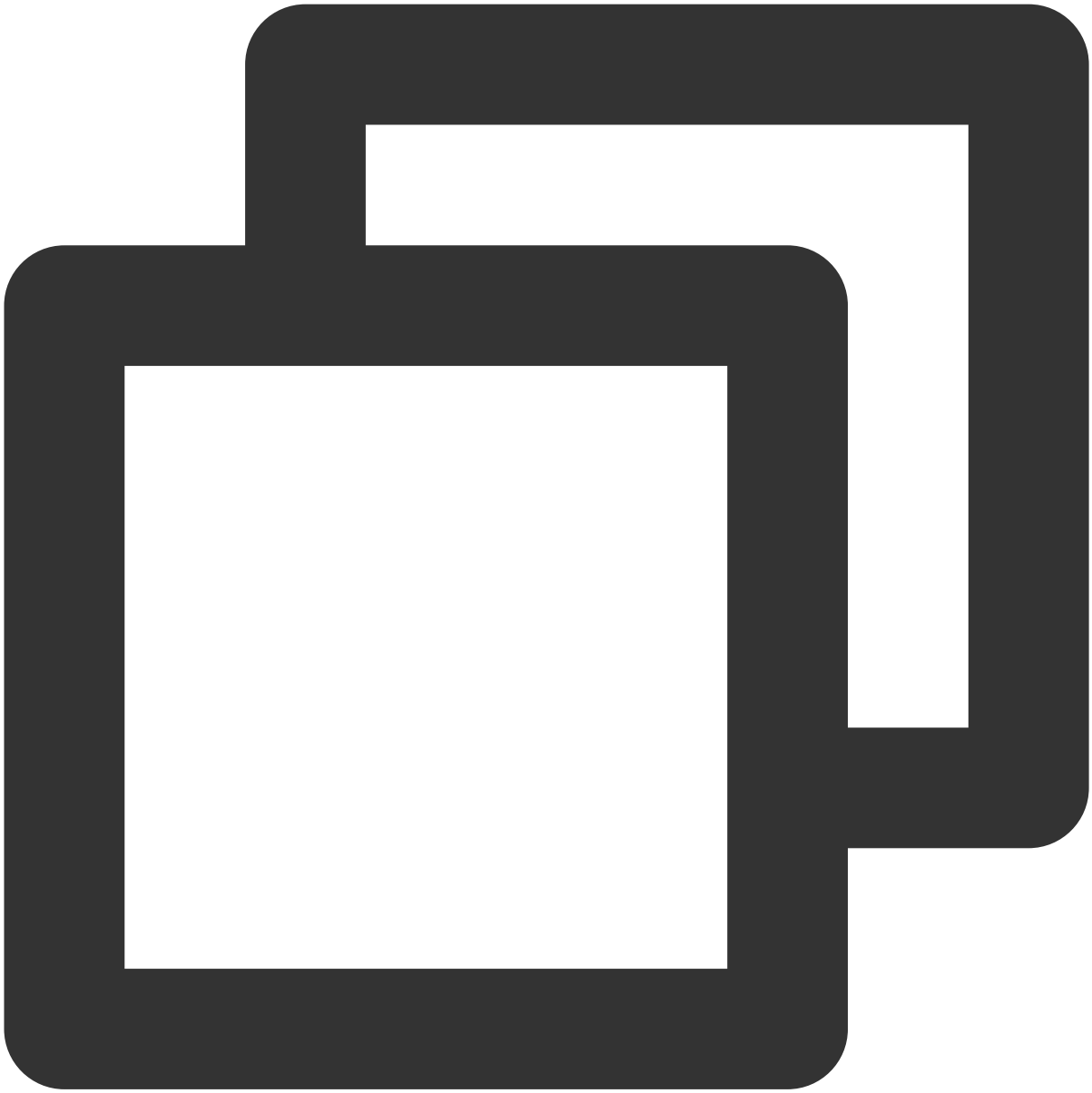
Method 1

Method 2



```
// android/app/build.gradle

android {
    ...
    defaultConfig {
        ...
        manifestPlaceholders = [
            "VIVO_APPKEY" : "`APPKEY` of the certificate assigned to your applicati
            "VIVO_APPID" : "`APPID` of the certificate assigned to your application
            "HONOR_APPID" : "`APPID` of the certificate assigned to your applicatio
        ]
```

```
    }
}
```



```
// android/app/src/main/AndroidManifest.xml

// Vivo begin
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.api_key"
    android:value="`APPKEY` of the certificate assigned to your application" />
<meta-data tools:replace="android:value"
    android:name="com.vivo.push.app_id"
```

```
    android:value="`APPID` of the certificate assigned to your application" />
// Vivo end

// Honor begin
<meta-data tools:replace="android:value"
    android:name="com.hihonor.push.app_id"
    android:value="`APPID` of the certificate assigned to your application" />
// Honor end
```

Adapting to Huawei, Honor, and Google FCM

Follow the vendor's method to integrate the corresponding plugin and JSON configuration file.

Note:

The following adaptation for Honor is only needed for version 7.7.5283 and above.

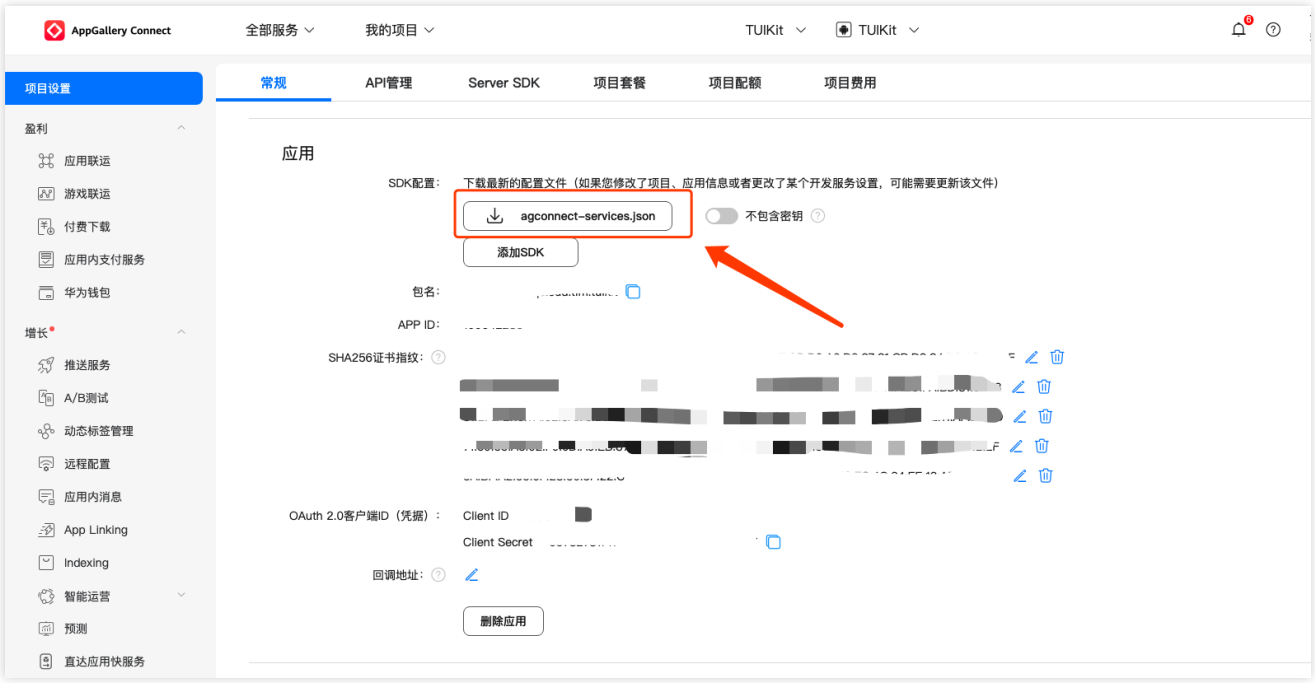
1.1 Download the configuration file and add it to the root directory of the project/Android/app.

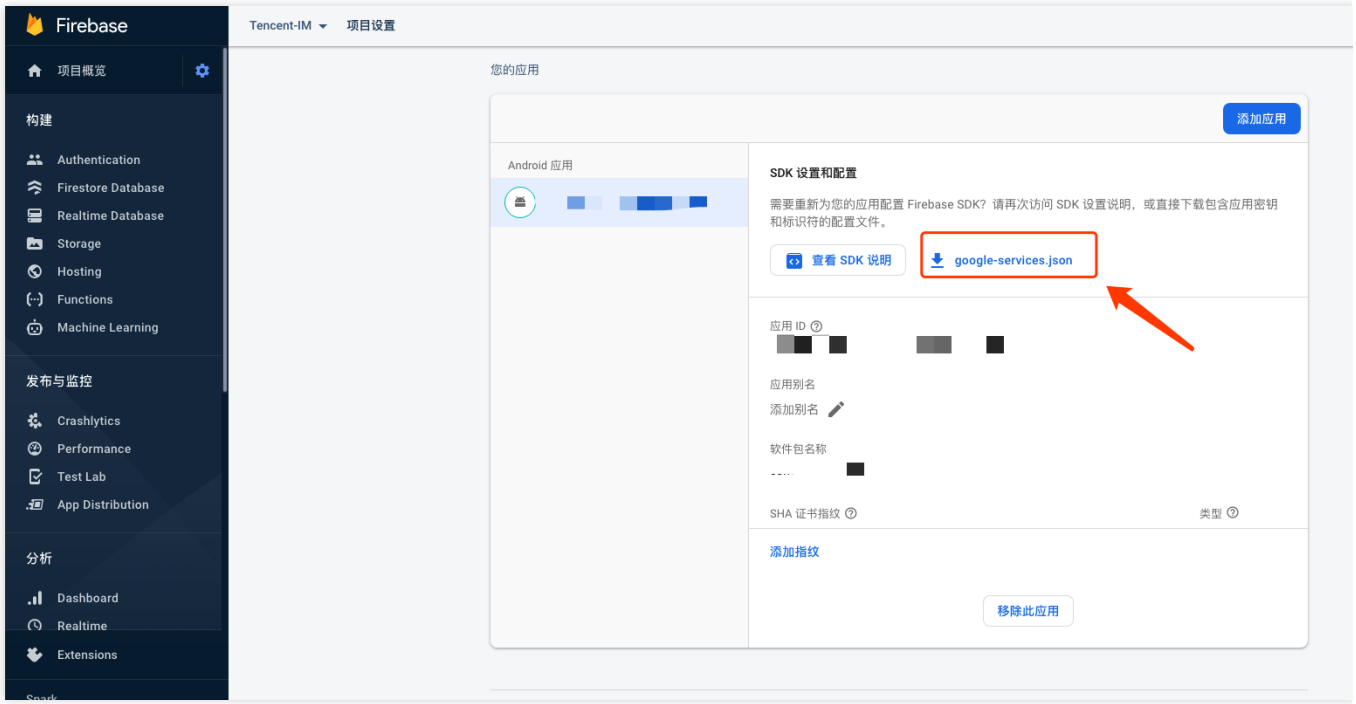
Huawei

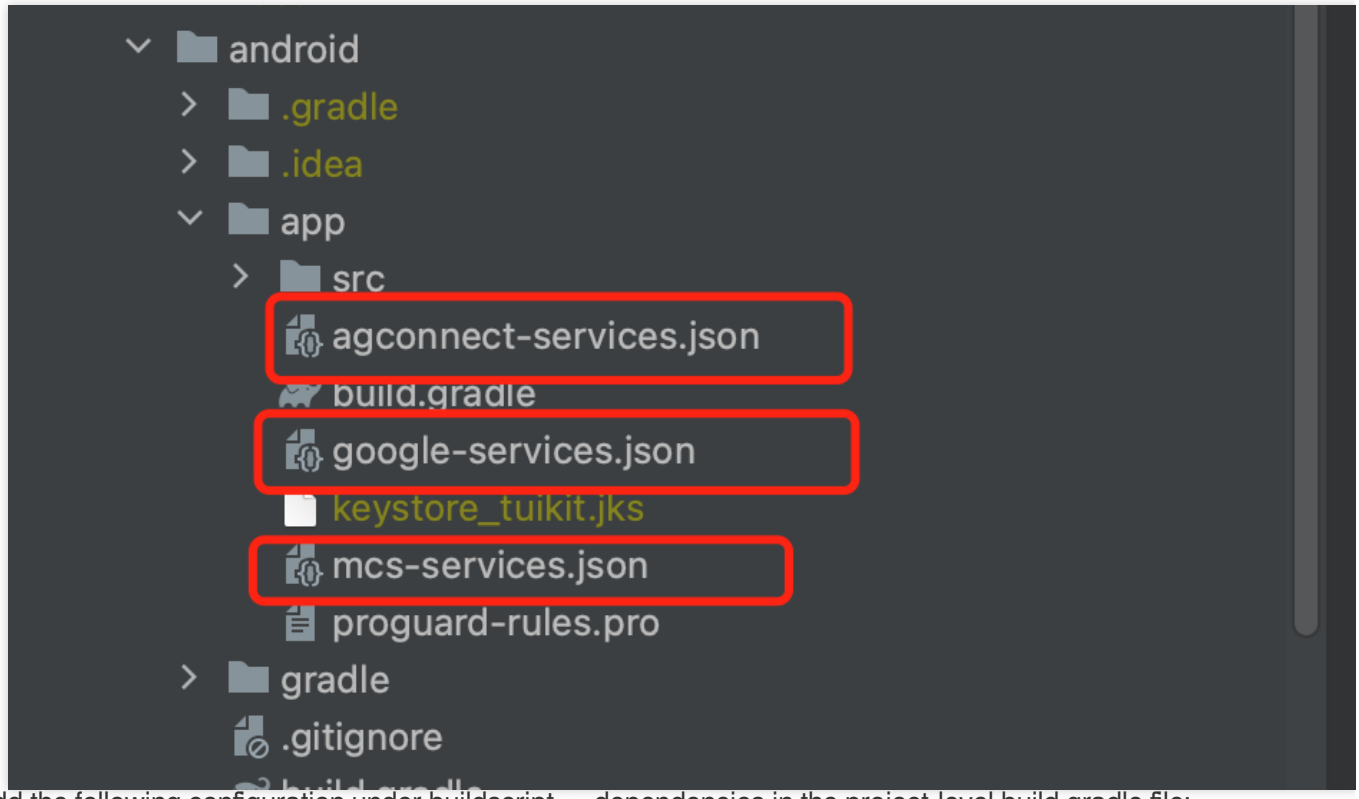
HONOR

Google FCM

Operation Path







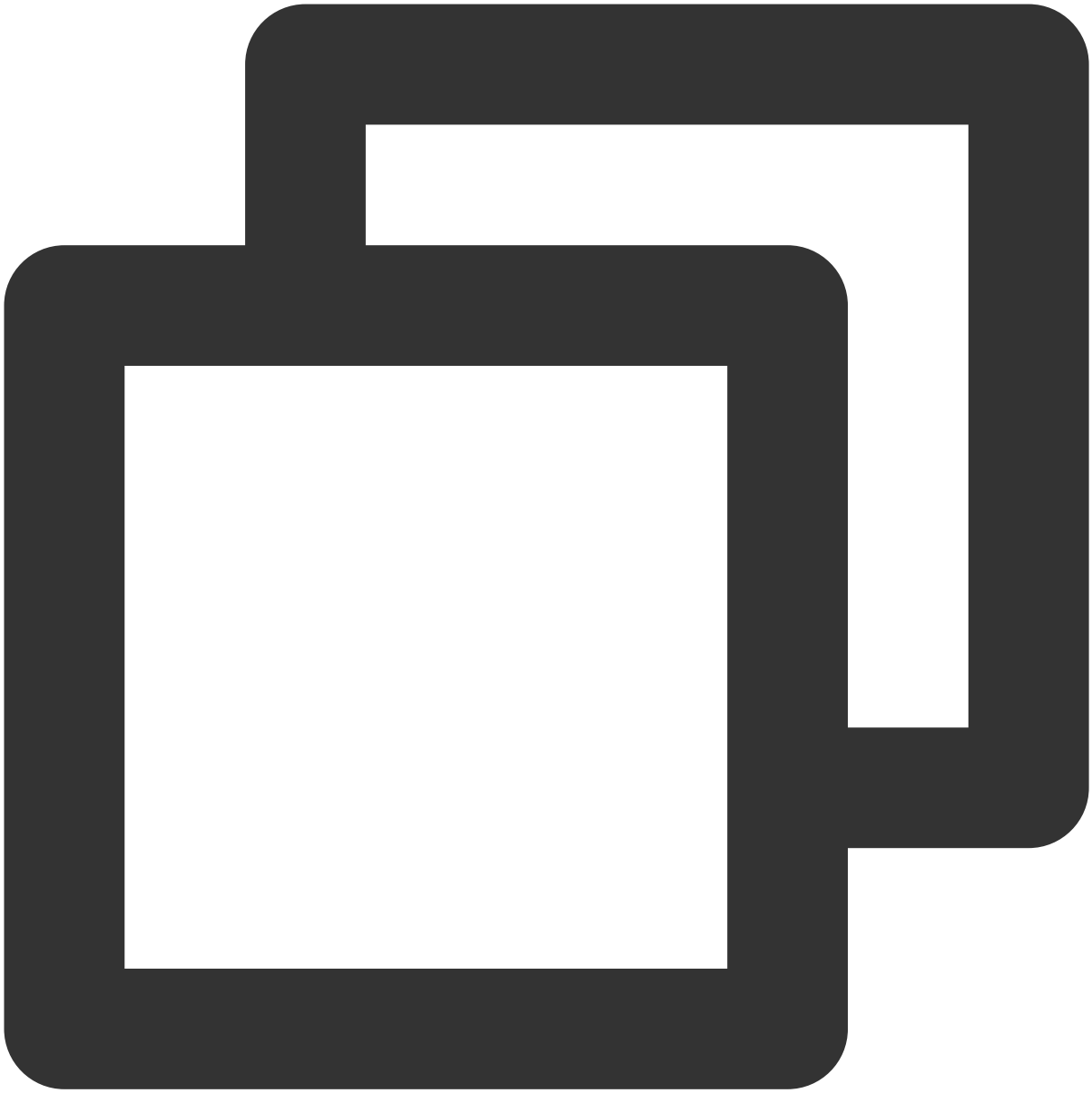
1.2 Add the following configuration under buildscript -> dependencies in the project-level build.gradle file:

For Gradle version 7.1 and above

Gradle version 7.0

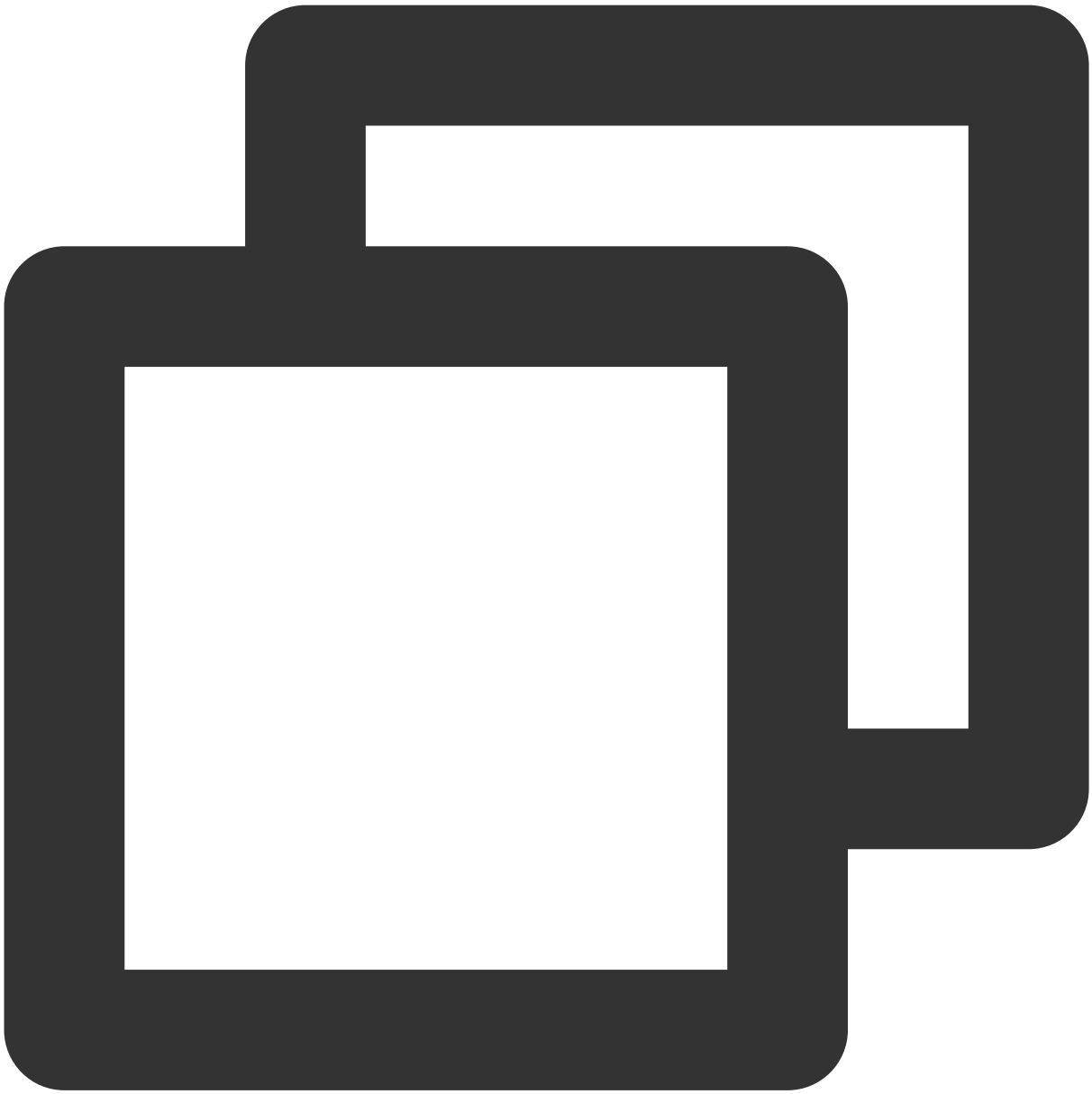
Versions Below Gradle 7.0

Add the following configuration under buildscript -> dependencies in your project-level build.gradle file:



```
buildscript {
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

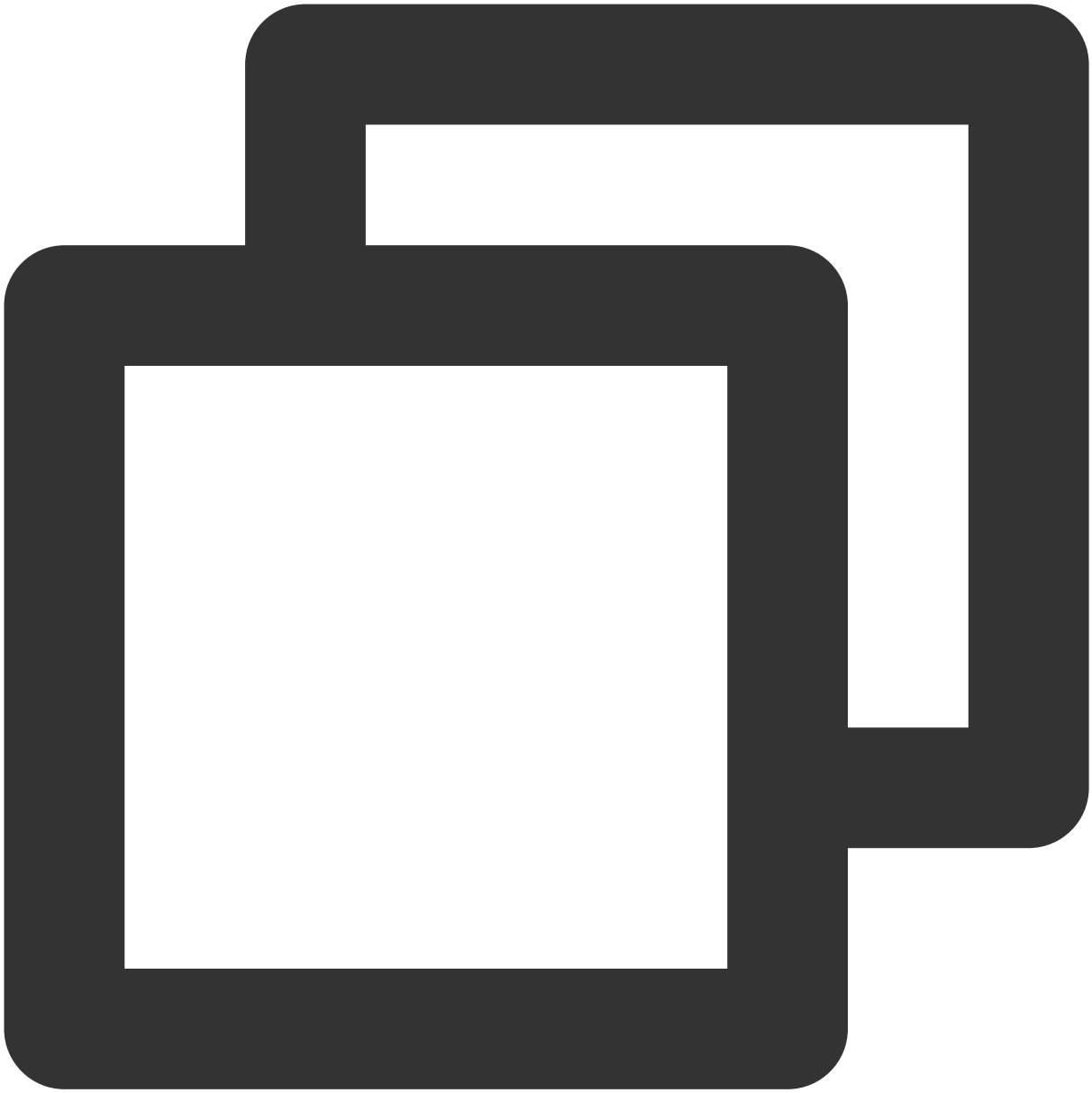
In the project-level settings.gradle file, add the following repository configurations under pluginManagement -> repositories and dependencyResolutionManagement -> repositories:



```
pluginManagement {
    repositories {
        gradlePluginPortal()
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configure the Maven repository address for HMS Core SDK.
        maven {url 'https://developer.huawei.com/repo/'}
        maven {url 'https://developer.hihonor.com/repo'}
```

```
    }
}
dependencyResolutionManagement {
    ...
    repositories {
        mavenCentral()
        maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
        // Configure the Maven repository address for HMS Core SDK.
       maven {url 'https://developer.huawei.com/repo/'}
       maven {url 'https://developer.hihonor.com/repo'}
    }
    }
}
```

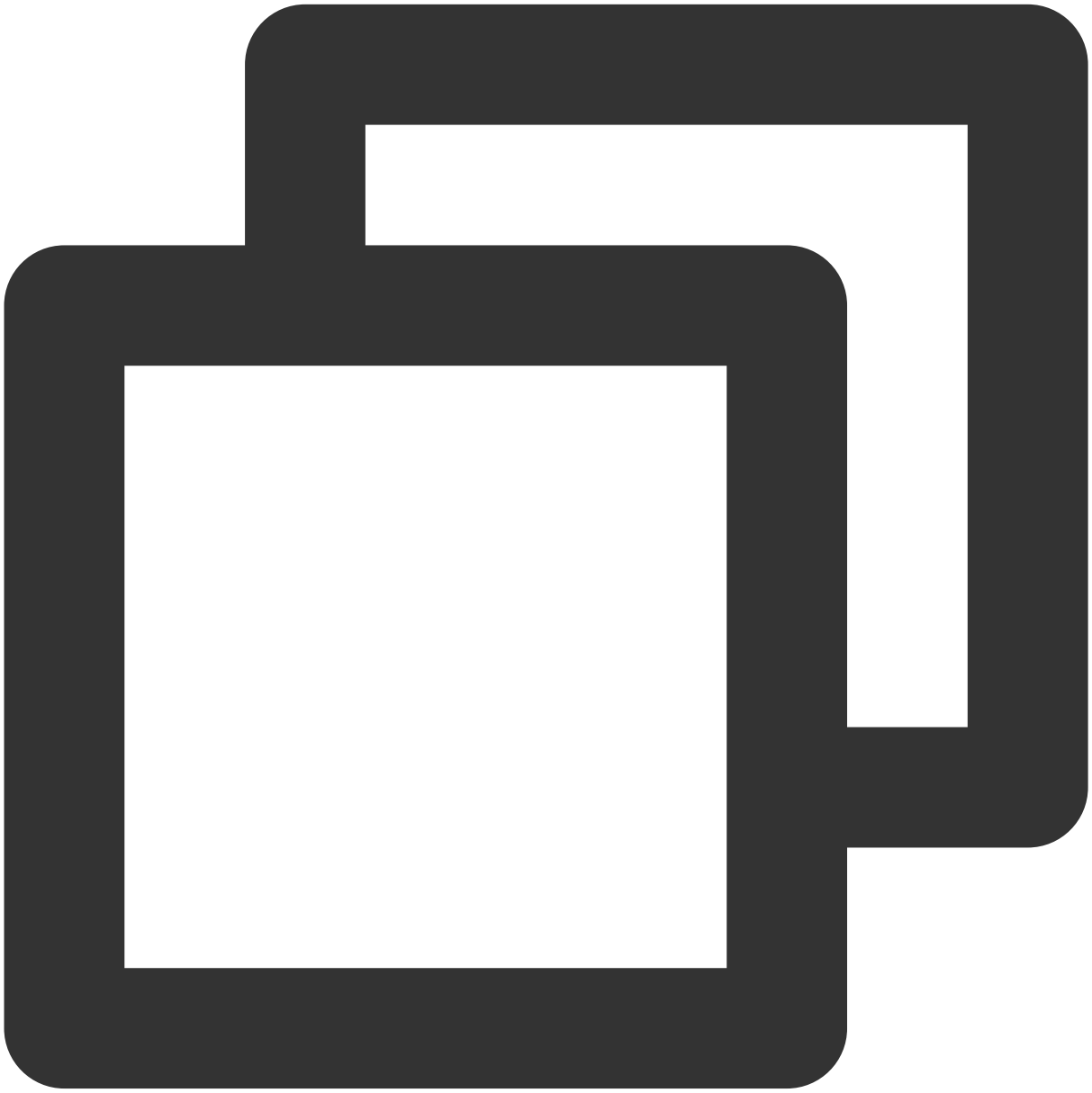
Add the following configuration under buildscript in your project-level build.gradle file:



```
buildscript {
    repositories {
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
```

```
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}
```

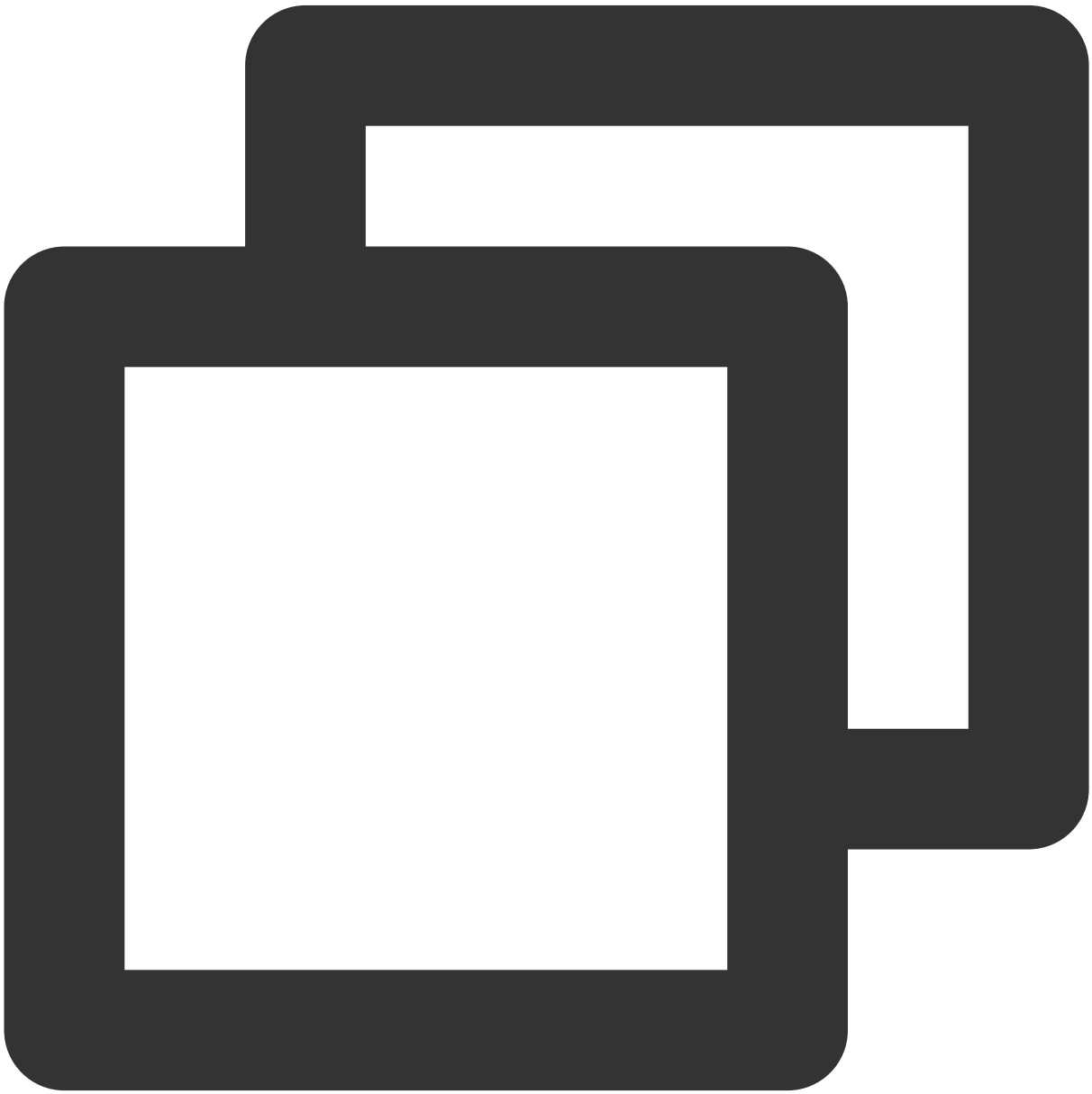
Add the following repository configurations under dependencyResolutionManagement -> repositories in your project-level settings.gradle file:



```
dependencyResolutionManagement {
    ...
    repositories {
```

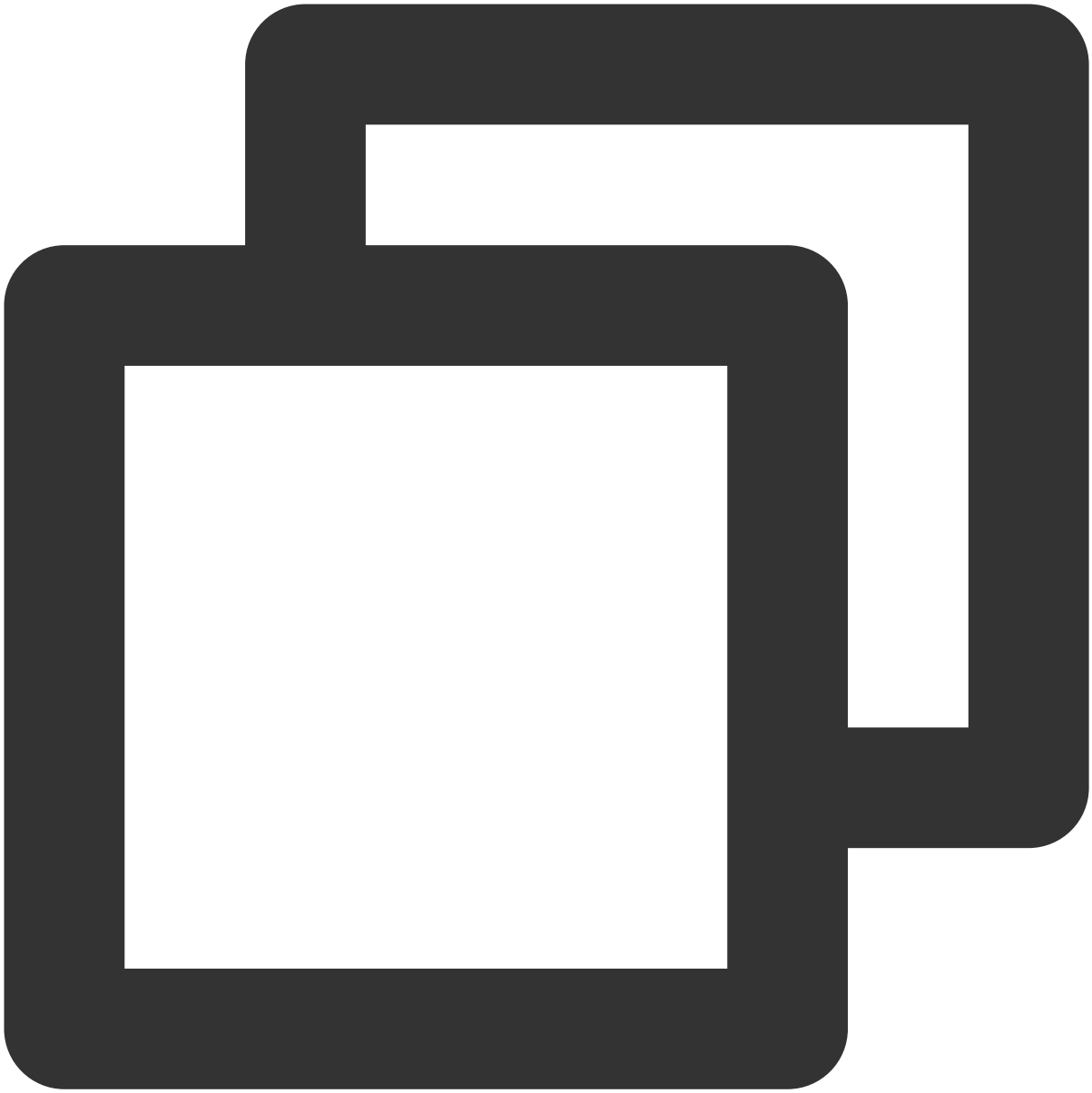
```
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

Add the following configuration under buildscript and allprojects in the project-level build.gradle file:



```
buildscript {
    repositories {
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
    dependencies {
        ...
        classpath 'com.google.gms:google-services:4.3.15'
        classpath 'com.huawei.agconnect:agcp:1.4.1.300'
        classpath 'com.hihonor.mcs:asplugin:2.0.1.300'
    }
}

allprojects {
    repositories {
        mavenCentral()
maven { url "https://mirrors.tencent.com/nexus/repository/maven-public/" }
// Configure the Maven repository address for HMS Core SDK.
maven {url 'https://developer.huawei.com/repo/'}
maven {url 'https://developer.hihonor.com/repo'}
    }
}
```

1.3 Add the following configuration in the app-level build.gradle file:



```
apply plugin: 'com.google.gms.google-services'
apply plugin: 'com.huawei.agconnect'
apply plugin: 'com.hihonor.mcs.asplugin'
```

Step 5: Handle the message click callback, and parse the parameters

Please define a function to receive the push message click callback event.

Define the function with the following parameters: `(ext: string, userID?: string, groupID?: string): void;` .

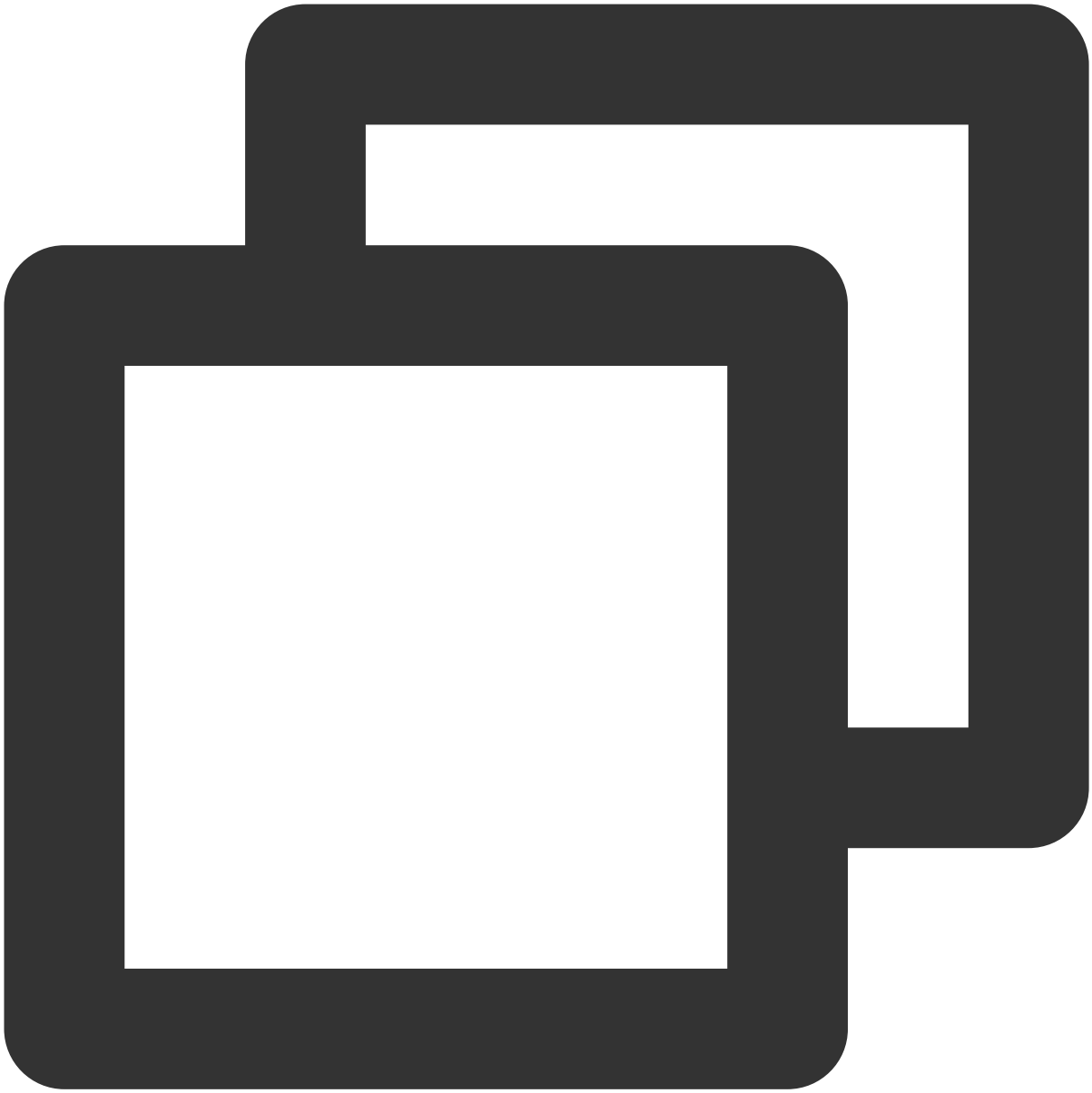
Among them, the ext Field carries the complete ext information specified by the sender. If not specified, a default value is assigned. You can navigate to the corresponding page by parsing this field.
The userID and groupID fields are automatically attempted by the plugin to parse the ext Json String, retrieving the single chat partner userID and group chat groupID information. If you have not defined the ext Field yourself, and the ext Field is default specified by the SDK or UIKit, then you can use the default parsing provided here. If parsing fails, it will be null.
You can define a function to receive this callback and navigate to the corresponding session page or your business page accordingly.

Step 6: Register Push Plugin

Please register the push plugin immediately after logging into IM and before using other plugins (such as CallKit).

Invoke the `TimPushPlugin.getInstance().registerPush` method, passing in a callback function defined for clicks.

Furthermore, you have the option to also pass in `apnsCertificateID` for the iOS push certificate ID and `androidPushOEMConfig` for the Android push vendor configuration. These two configurations should have been specified in previous steps, and if no modification is necessary, they do not need to be passed again.

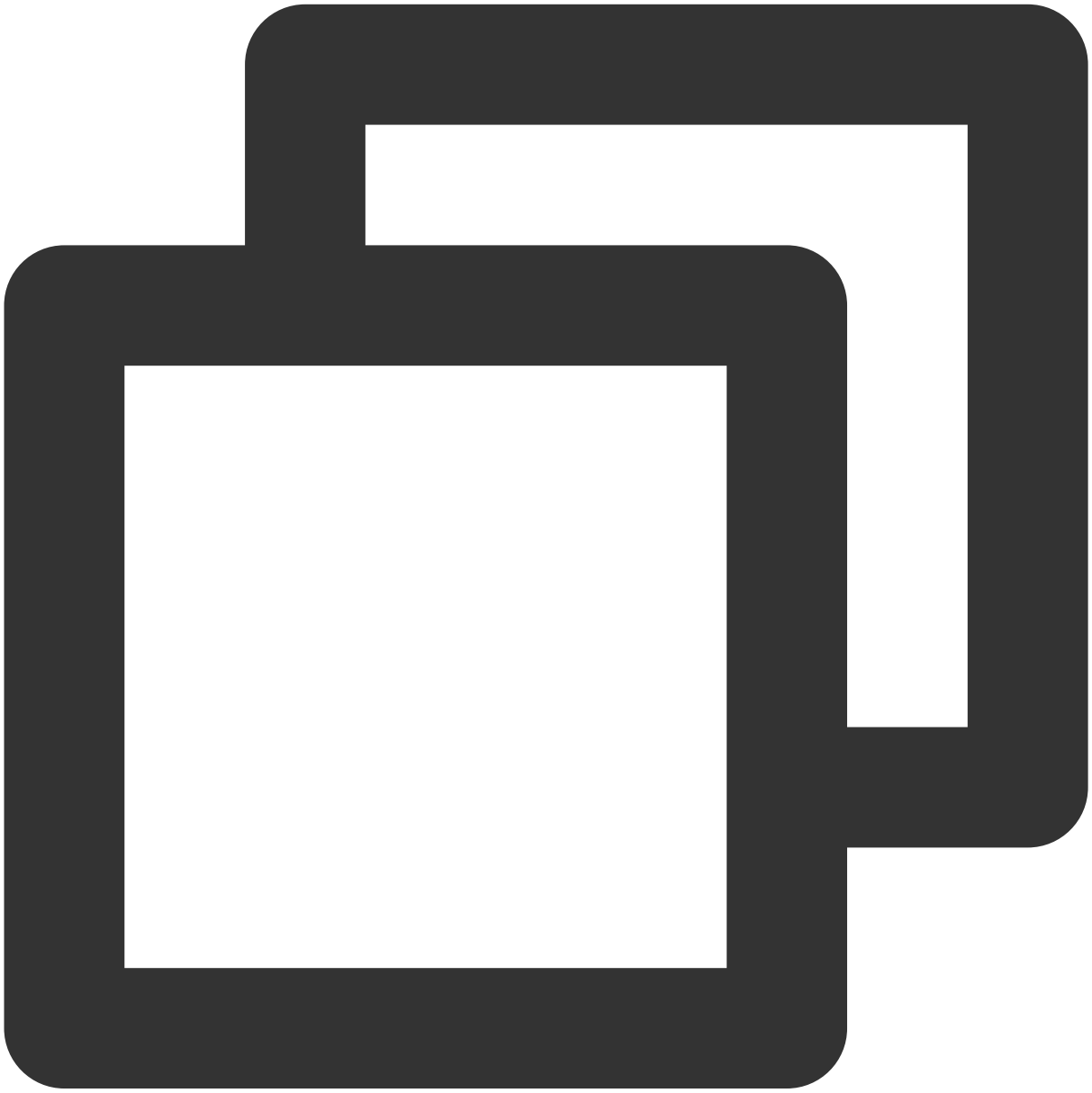


```
TimPushPlugin.getInstance().registerPush({
      onNotificationClicked: (extString) => {},
})
```

Note:

If your application requires the use of **push plugin for business message notifications** , and it does not immediately start and log in to the IM module after launching, or if you need to handle business navigation by obtaining message click callbacks before logging in to the IM module, it is recommended that you call

the `TimPushPlugin.getInstance().registerOnNotificationClickedEvent` method as soon as possible to manually mount the message click callback, so that you can promptly obtain the message parameters. In this scenario, you can execute this function before calling `TimPushPlugin.getInstance().registerPush` and place it as early as possible in the code.



```
TimPushPlugin.getInstance().registerOnNotificationClickedEvent({onNotificationClick
```

Step 7: Message Push Delivery Statistics

If you need to collect data on delivery, please complete the setup as follows:

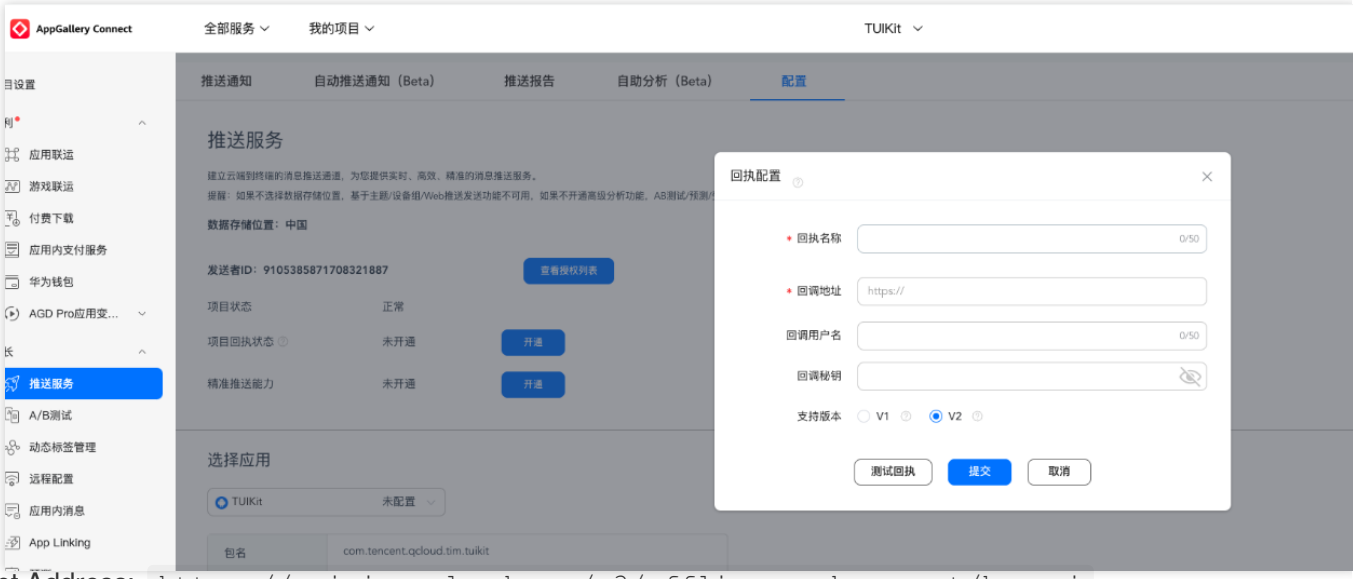
Huawei

HONOR

vivo

Meizu

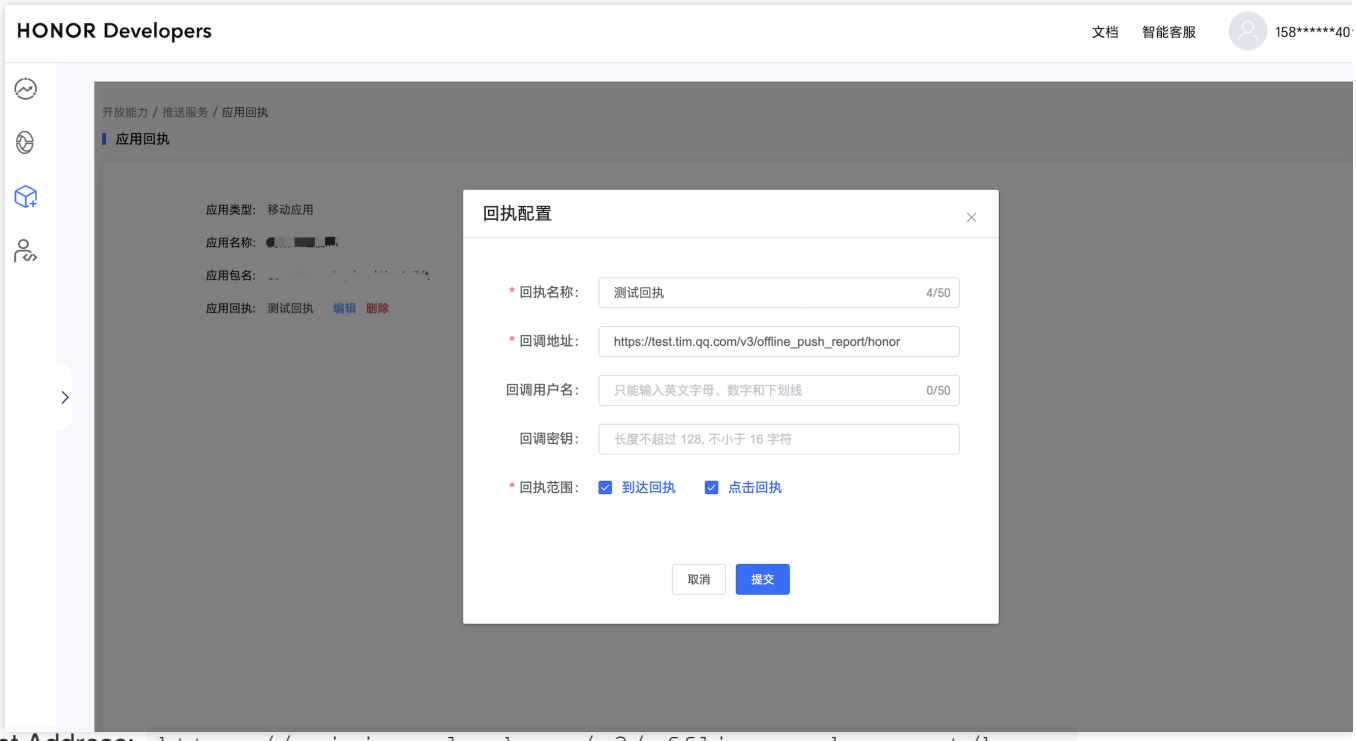
iOS



Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/huawei`

Note:

Huawei Push Certificate ID <= 11344, using Huawei Push v2 version interface does not support reach and click receipt, please regenerate and update the certificate ID.



Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/honor`

Callback Address Configuration	Receipt ID Configuration i
 Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/vivo`	

Enable Receipt Switch	Configure Receipt Addres

Receipt Address: `https://api.im.qcloud.com/v3/offline_push_report/meizu`

Note:

After enabling the Receipt Switch, please make sure the Receipt Address is configured correctly. Failing to configure or configuring the wrong address will affect the push feature.

For iOS Push Notification Reach Statistics Configuration, please refer to **Statistics Push Arrival Rate**.

No configuration is needed for other supported manufacturers; FCM does not support the push notification statistics feature.

Congratulations! You have completed the integration of the push plugin. Please be reminded: After the trial period or subscription expires, the push service (including regular message offline push, all-staff/Tag push, etc.) will automatically cease. To avoid affecting the normal use of your services, please make sure to purchase/renew in advance.

Statistics

Last updated：2024-06-13 10:21:45

This article aims to introduce the various statistics pages for push statistics data, enabling users to quickly query statistical data.

Note:

The Statistics feature will only record the push data details of the last device logged in to, and does not support scenarios with multiple devices logged in to.

Regular Push

Push Records

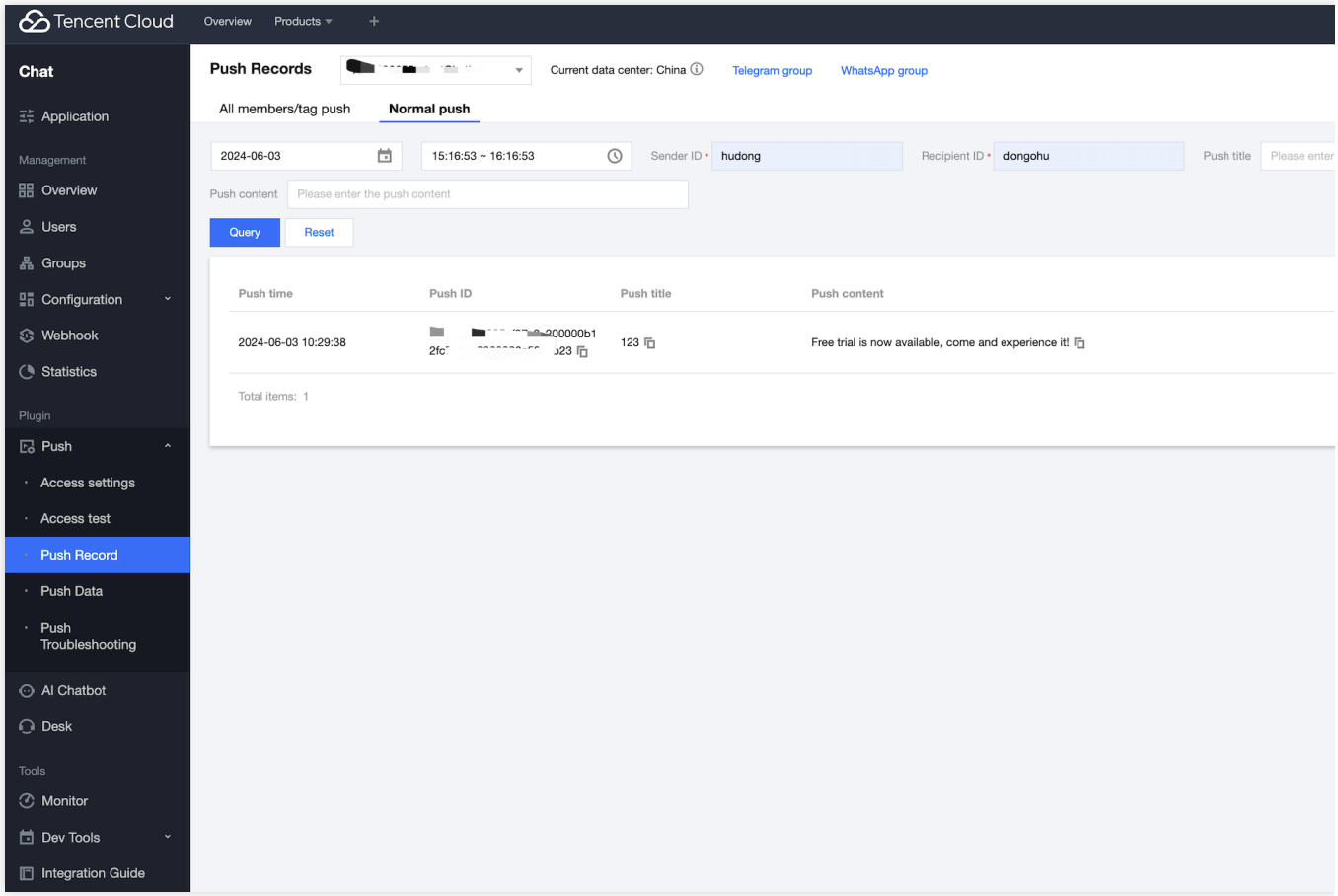
You can query all push record data for users under this application. It is necessary to specify the time window, Sender ID, and Recipient ID. Querying user push data includes push time, Push ID, push title, and push content; it also supports locating specific push records by searching for push titles or push content.

Query conditions

Time window: A specific time period within a specified date, maximum one hour, precise to minutes and seconds, required.
Sender ID: Required.
Recipient ID: Required.
Push Title: Optional.
Push Content: Optional.

Query results (Recorded in the last 7 days)

Push Time: The exact time of push reach.
Push ID: The unique ID of the push message, which can be used to locate the full push link situation of this push in the troubleshooting tool.
Push Title: Push display title.
Push Content: Push display content.



Push Data

The statistics display various push metrics data of the application in recent days, mainly including an overview of yesterday's push, push data conversion funnel for multi-type time intervals, supports viewing classified by vendor dimensions, and details of daily push metrics data.

Yesterday's Push Overview Metrics Data

Includes the number that can be sent, the number sent, the number reached, the number of clicks, the actual delivery rate, the reach rate, the click-through rate.

Time Range

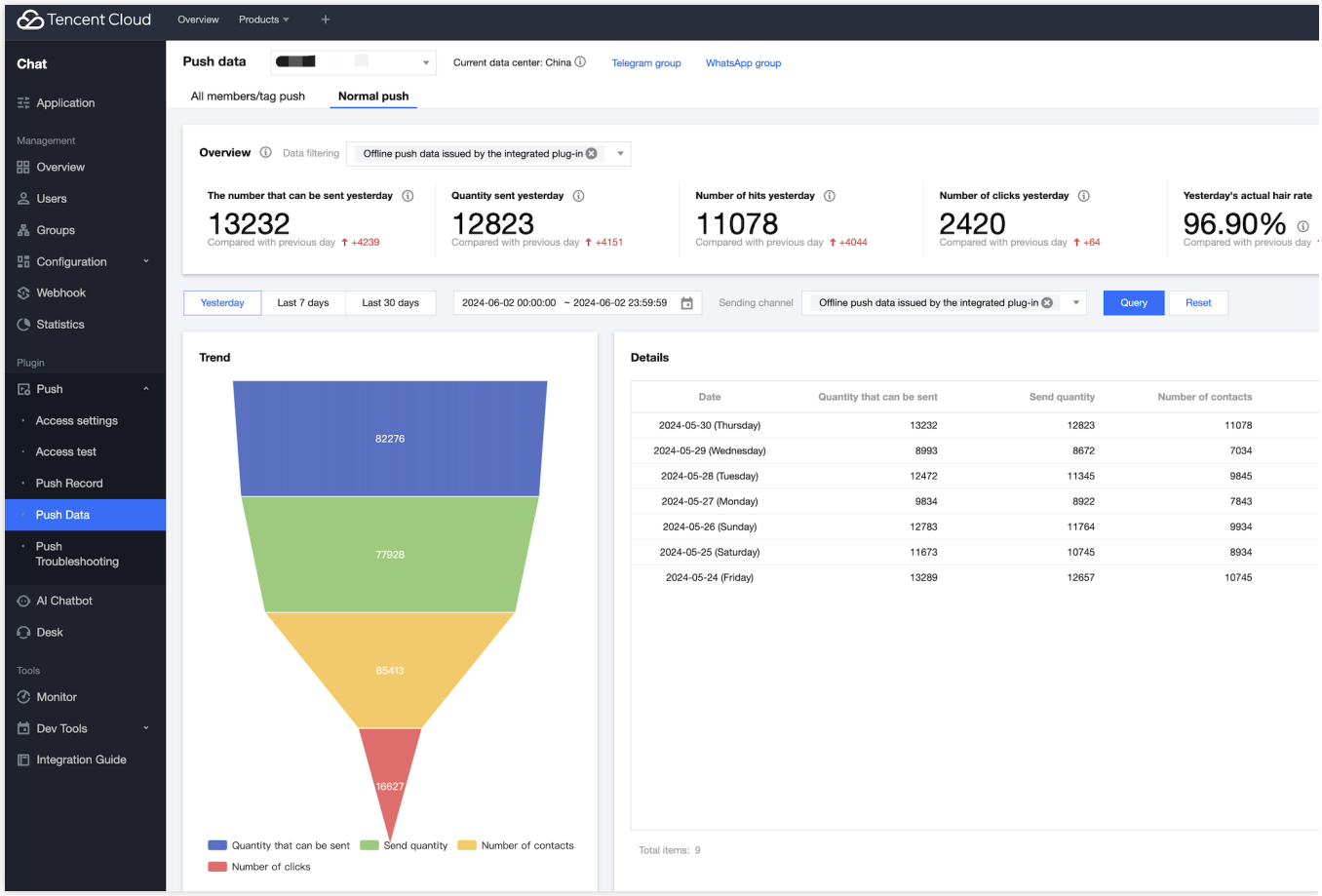
Supports yesterday, the last 7 days, the last 30 days, and specified time window types.

Push Data Conversion Funnel

Includes the dimensions of sendable quantity, sent quantity, reached quantity, and clicked quantity.

Support viewing by vendor dimension

Support viewing by all vendor categories.



All-staff/Tag Push

Push Records

You can query all the full/Tag push record data sent to users under this application, a specific time window must be specified, and the push data queried includes push time, task ID, and push request content; it also supports searching and locating specific push records based on push content.

Query conditions

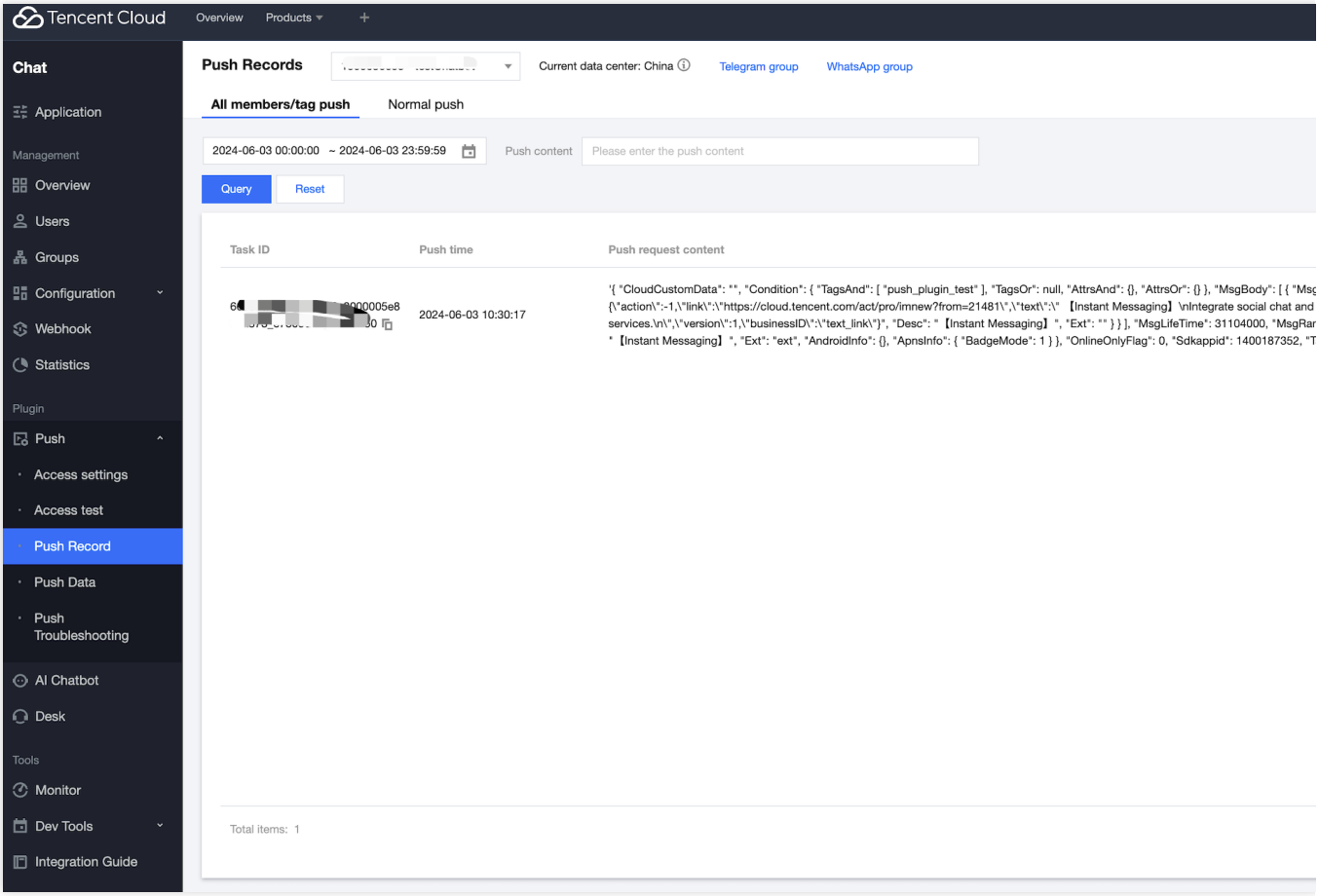
Time window: A specific time period within a specified date, up to 7 days, precise to minutes and seconds, required.
Push Content: Optional.

Query results (Recorded in the last 7 days)

Push Time: The exact time of push reach.

Task ID: The unique ID of the push message, which can be used to locate the full push notification link status in the troubleshooting tool.

Push Content: The detailed data in JSON format of the push.



Push Data

The statistics display various push metrics data of the application in recent days, including the features page and the meaning of the metric data, same as standard push.

Yesterday's Push Overview Metrics Data

Includes the number that can be sent, the number sent, the number reached, the number of clicks, the actual delivery rate, the reach rate, the click-through rate.

Time Range

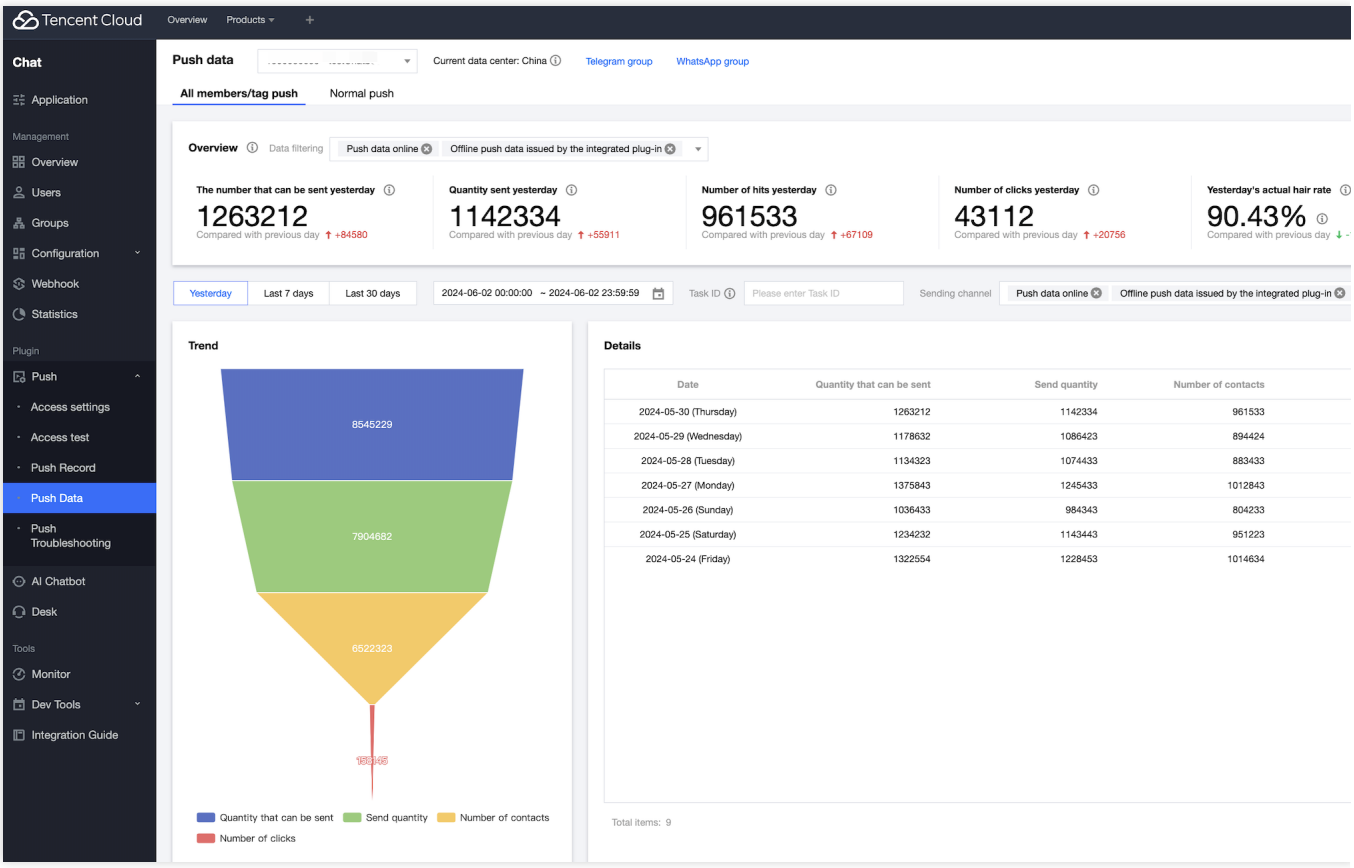
Supports yesterday, the last 7 days, the last 30 days, and specified time window types.

Push Data Conversion Funnel

Includes the dimensions of sendable quantity, sent quantity, reached quantity, and clicked quantity.

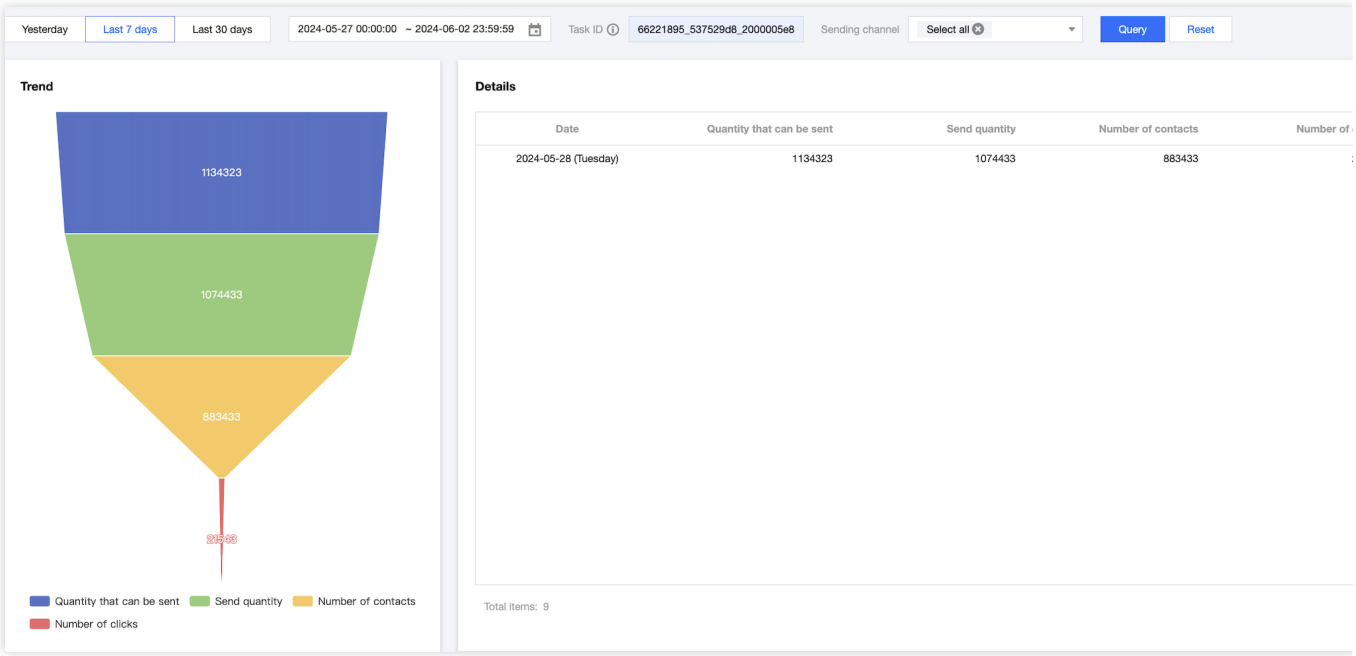
Support viewing by vendor dimension

Support viewing by all vendor categories.



Task ID

All-staff/Tag Push supports viewing the detailed metrics data and details of a single push by Task ID.



Metrics Data Calculation Method

Sendable Quantity: The total number of valid devices that can be reached by filtering the target population selected for the push task (when a device is online and switches to the background, both online and offline pushes will be sent simultaneously, and it will be counted as two devices, not deduplicated).

Sent Quantity: The total number of valid devices that have been successfully distributed through the Instant Messaging (IM) Channel and the Manufacturer Channel among the valid devices that can be sent (when a device is online and switches to the background, both online and offline pushes will be sent simultaneously, and it will be counted as two devices, not deduplicated).

Reached Quantity: The total number of receipts successfully received by the device terminal through the Instant Messaging (IM) Channel and the Manufacturer Channel (when a device is online and switches to the background, both online and offline pushes will be sent simultaneously, and it will be counted as two devices, not deduplicated).

Clicked Quantity: The total number of receipts for clicks after the push is successfully displayed.

Actual Delivery Rate: Sent Quantity / Sendable Quantity * 100%.

Reach Rate: Reached Quantity / Sent Quantity * 100%.

Click-through Rate: Clicked Quantity / Reached Quantity * 100%.

Push Statistics Support Status by Manufacturer

Manufacturer	Reach	Click
Huawei	Supported (Requires configuration of Acknowledgement)	Supported (Requires integration of TIMPush)
HONOR	Supported (Requires configuration of Acknowledgement)	Supported (Requires integration of TIMPush)
vivo	Supported (Requires configuration of Acknowledgement)	Supported (Requires integration of TIMPush)
OPPO	Supported	Supported (Requires integration of TIMPush)
Mi	Supported	Supported (Requires integration of TIMPush)
Meizu	Supported (Requires configuration of Acknowledgement)	Supported (Requires integration of TIMPush)
FCM	Not supported	Not supported
Apns	Supported (Requires integration of TIMPush)	Supported (Requires integration of TIMPush)

Troubleshooting Tool

Last updated：2024-06-13 10:21:45

This article aims to introduce the various pages and feature usage of the Push Notification Troubleshooting Tool, guiding users to troubleshoot the entire push notification link details when an offline push message is not received.

Regular Push

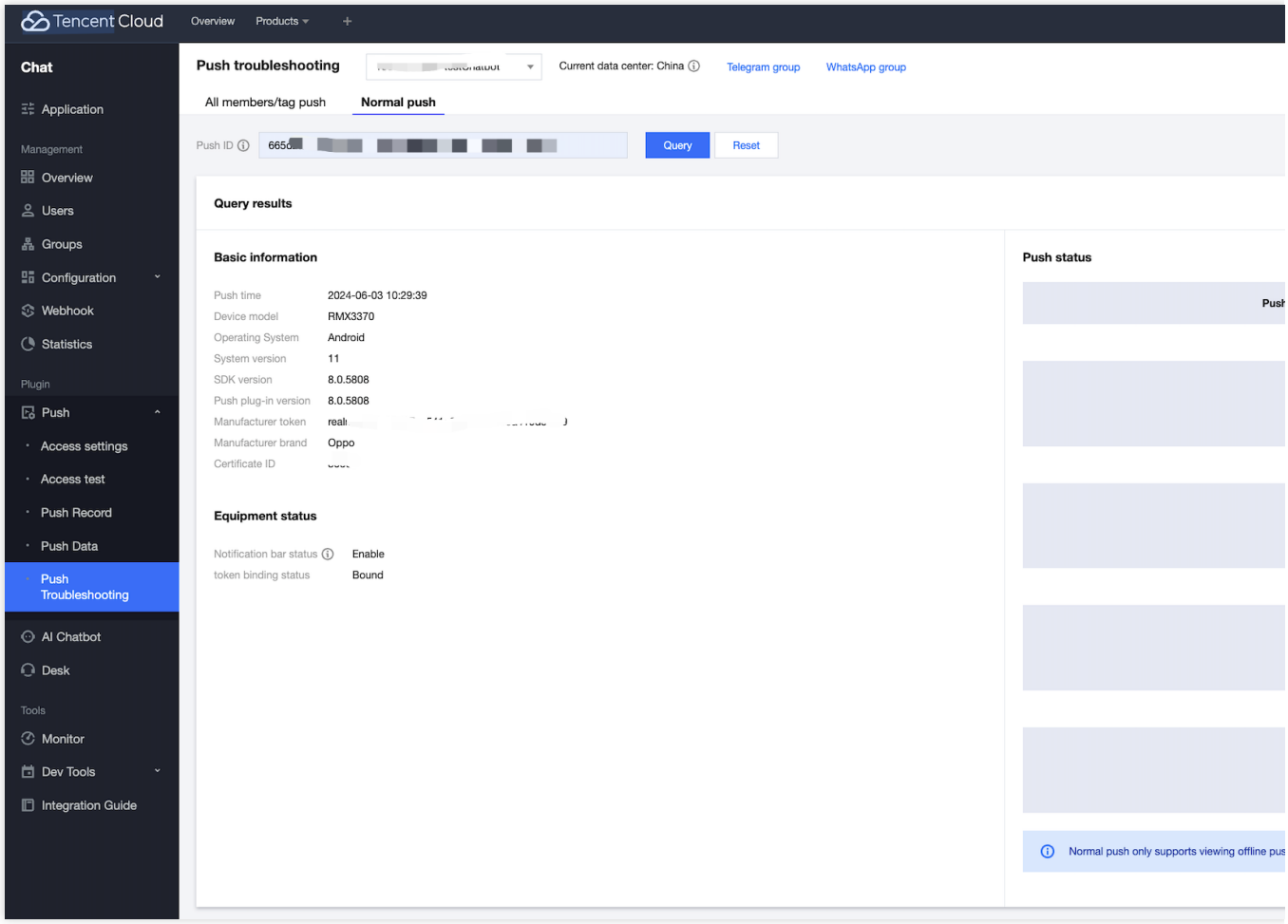
The Regular Push Troubleshooting Tool is primarily for developer users to troubleshoot specific push notification reception issues. By finding the push record and copying the unique Push ID of that push, one can query the detailed push notification link details of that push.

Query Fields

Push ID: The unique ID used to identify a regular push, which can be found in the push records. It is required.

Viewing the query result

Basic Information: The basic information of the current device at the time of push delivery, including model, operating system, SDK, and Plugin Version Number, etc.
Device Status: The push's notification bar switch status and the device's token binding status at the time of dispatch.
Push Status: The end-to-end information for the push delivery, including the entire link from IM server > Manufacturer Server > Terminal Device > User Click.



All-staff/Tag Push

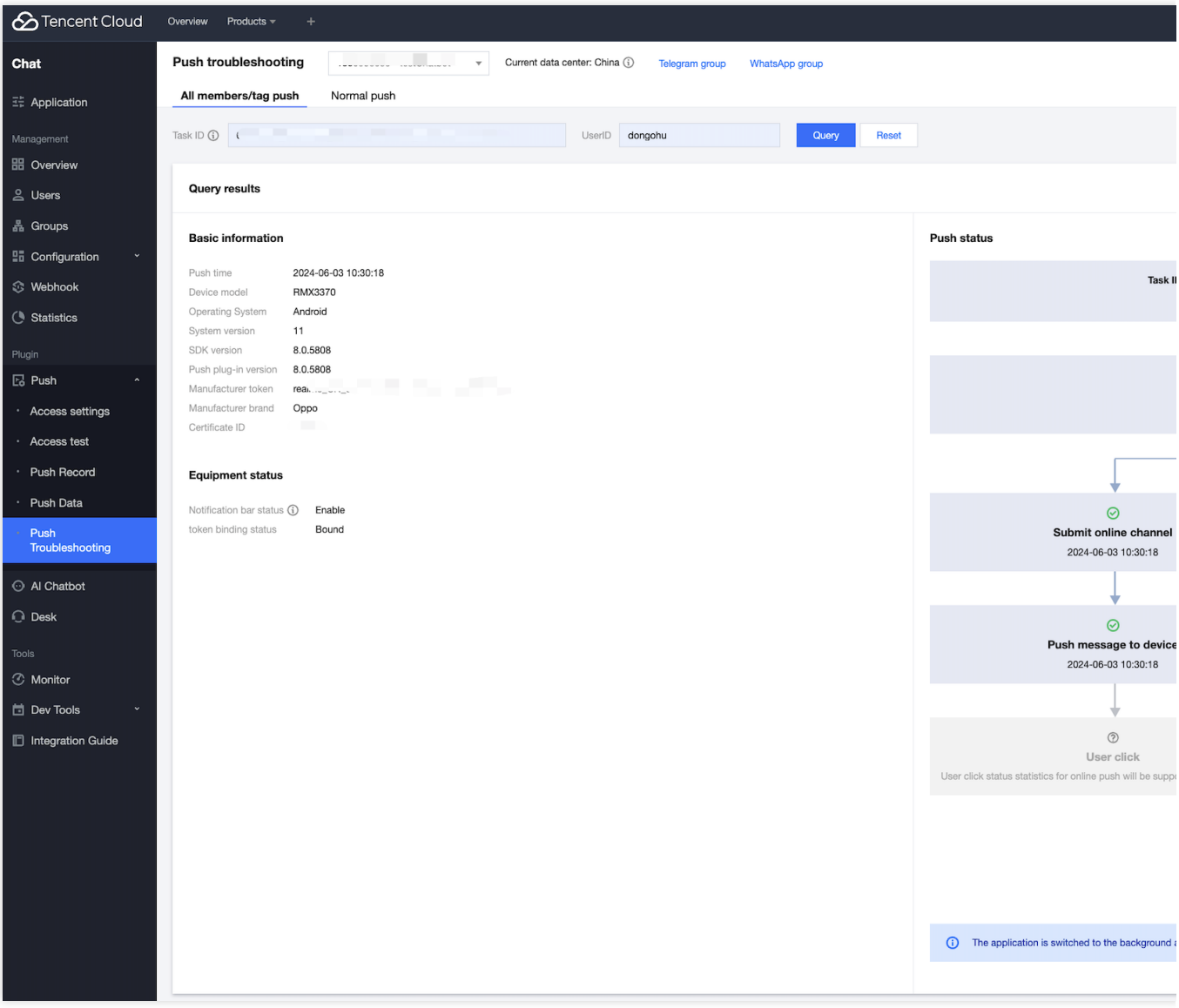
The All-staff/Tag Push Troubleshooting Tool is primarily for administrator users to investigate the reception status of a specific user in an All-staff/Tag push. By finding the push record and copying the unique Task ID of that push, and entering the recipient's UserID, one can query the detailed push notification link details for that UserID.

Query Fields

Task ID: The unique ID used to identify an All-staff/Tag push, which can be found in the push records. It is required.
UserID: A user ID for whom the All-staff/Tag push is intended, also the recipient of the push. It is required.

Viewing the query result

Basic Information: At the time of push delivery for the specified UserID, the basic information of the current device includes model, operating system, SDK, and plugin version number, etc.
Device Status: At the time of push delivery for the specified UserID, the device's notification bar switch status, as well as the device's token binding status.
Push Status: The end-to-end information of the push delivered to the specified UserID, including the entire link situation from IM server > Manufacturer Server > Terminal Device > User Click.



Client APIs
Android

Last updated：2024-08-01 11:28:30

TIMPushManager

public abstract class TIMPushManager: Push Plugin Interface Class.

API overview

Register/Unregister Push Service Interface

After initializing and successfully logging in to IM, you can register for push services.

API	Description
registerPush	To register for push services, read the configuration file timpush-configs.json from the project.
unRegisterPush	Unregister to close offline push services, call before logging out of the IM account.
disableAutoRegisterPush	To disable the plugin from automatically registering for push services after logging in, call before registering for push services.

FCM Custom Ringtone Configuration Interface

Once configured, the custom ringtone is effective. The sender must include the channelId in the offline message sent.

API	Description
setCustomFCMRing	To configure FCM's custom ringtone, it needs to be called before registering for push services.
setCustomConfigFile	The custom replacement plugin by default reads the push service registration configuration file timpush-configs.json, which needs to be called before registering for push services

Interface Details

Static Public Member Functions

static TIMPushManager getInstance(): Retrieves the TIMPushManager manager instance.

Member Function Description

abstract void registerPush(Context context, TIMPushCallback callback)

Register offline push service, call upon successful login to the IM account. (To facilitate the easiest possible integration into the push service, the plugin will automatically read the configuration file timpush-configs.json in the project to obtain the information needed to register for the push service)

Note:

You need to use the login API provided by TUILogin of the TUICore component to log in; the plugin will automatically detect this and register the push service.

If you do not wish to use the API provided by TUILogin, you need to manually call this interface to register the service after completing the login operation.

abstract void unRegisterPush(TIMPushCallback callback)

Unregister to close offline push services, call before logging out of the IM account.

Note:

If you do not wish to use the push service, you can manually call this interface to unregister the service.

If you log out using the logout API provided by TUILogin of the TUICore component, the plugin will automatically detect this and unregister the push service.

abstract void disableAutoRegisterPush()

To disable the plugin from automatically registering for the push service, it's necessary to call this before logging in.

Note:

If you log in using the login API provided by TUILogin in the TUICore component, the plugin will by default automatically register for the push service. Calling this interface can disable the automatic registration.

abstract void setCustomFCMRing(String channelId, String ringName, boolean enable)

To configure FCM's custom ringtone, it needs to be called before registering for push services.

Note:

Once configured, the custom ringtone is effective. The sender must include the channelId in the offline message sent.

Parameter description:

API	Description
channelId	FCM channel uniquely defines the channel ID of the notification bar within the app.
ringName	FCM channel defines the name of the push ringtone for the notification bar, located in the raw directory and does not require a file suffix.

enable	Set whether the offline push prompt ringtone uses a custom ringtone.

abstract void setCustomConfigFile(String configs)

The custom replacement plugin by default reads the push service registration configuration file timpush-configs.json, which needs to be called before registering for push services.

Note:

Mainly used for dynamically switching different configuration files for push registration in multiple environments, for example: push feature integration and testing under different configuration files in production and test environments;

For methods to switch during the static compilation period, please refer to: buildConfigField("String", "custom_timpush_configs", "\\"Custom Definition File Name\\"")

Parameter description:

Parameter	Description
configs	The name of the custom Definition configuration file should remain unchanged: "Engineering Root Directory/app/src/assets/"

iOS

Last updated：2024-06-13 10:21:45

API Overview

Register/Unregister Push Service Interface

After initialization and successful log in to IM, you can register for the push service.

API	Description
registerPush	Register the push service after log in to is complete
unRegisterPush	Unregister the offline push service when logging out of the IM account.
disableAutoRegisterPush	Disable the plugin after logging in to automatically register the push service; it needs to be called before registering the push service

Statistics on the push arrival rate of TIMPush

If you need to track the arrival and click data of push notifications, you need to proactively call this function in the Notification Service Extension.

API	Description
onReceiveNotificationRequest:inAppGroupID:callback:	It is only supported to be called in the Notification Service Extension's '-didReceiveNotificationRequest:withContentHandler:' method; appGroup indicates the App Group shared between the main App and the Extension. It needs to be configured in the Capability of the main App to enable App Groups.

Interface details

Function Description

+ (void)registerPush

Register the offline push service when the IM account logs in successfully. (The information needed to register the push service comes from the offlinePushCertificateID of the TIMPushDelegate protocol implemented in your AppDelegate)

Note:

You need to log in using the login interface provided by TUILogin in the TUICore component, and the plugin will automatically perceive and register the push service.

If you don't want to use the interface provided by TUILogin, after completing the login operation, you need to manually call this interface to register the service.

Usage: `[TIMPush registerPush];`

+ (void)unRegisterPush

Unregister the offline push service when logging out of the IM account.

Note:

You need to log in using the login interface provided by TUILogin in the TUICore component, and the plugin will automatically perceive and register the push service.

If you don't want to use the interface provided by TUILogin, after completing the login operation, you need to manually call this interface to register the service.

Usage: `[TIMPush unRegisterPush];`

+ (void)disableAutoRegisterPush

To turn off the plugin's automatic registration of the push service, you need to call this before logging in.

Note:

If you log in using the login interface provided by TUILogin in the TUICore component, the plugin automatically registers the push service by default. Calling this interface will turn off automatic registration.

Usage: `[TIMPush disableAutoRegisterPush];`

+ (void)onReceiveNotificationRequest:(UNNotificationRequest *)request inAppGroupID:(NSString *)appGroupID callback:(TIMPushNotificationExtensionCallback)callback

Statistics on the push arrival rate of TIMPush

You need to implement the `- applicationGroupID` method in the AppDelegate.m file, returning the App Group ID. And call this function in the Notification Service Extension's '- didReceiveNotificationRequest:withContentHandler:' method.

Note:

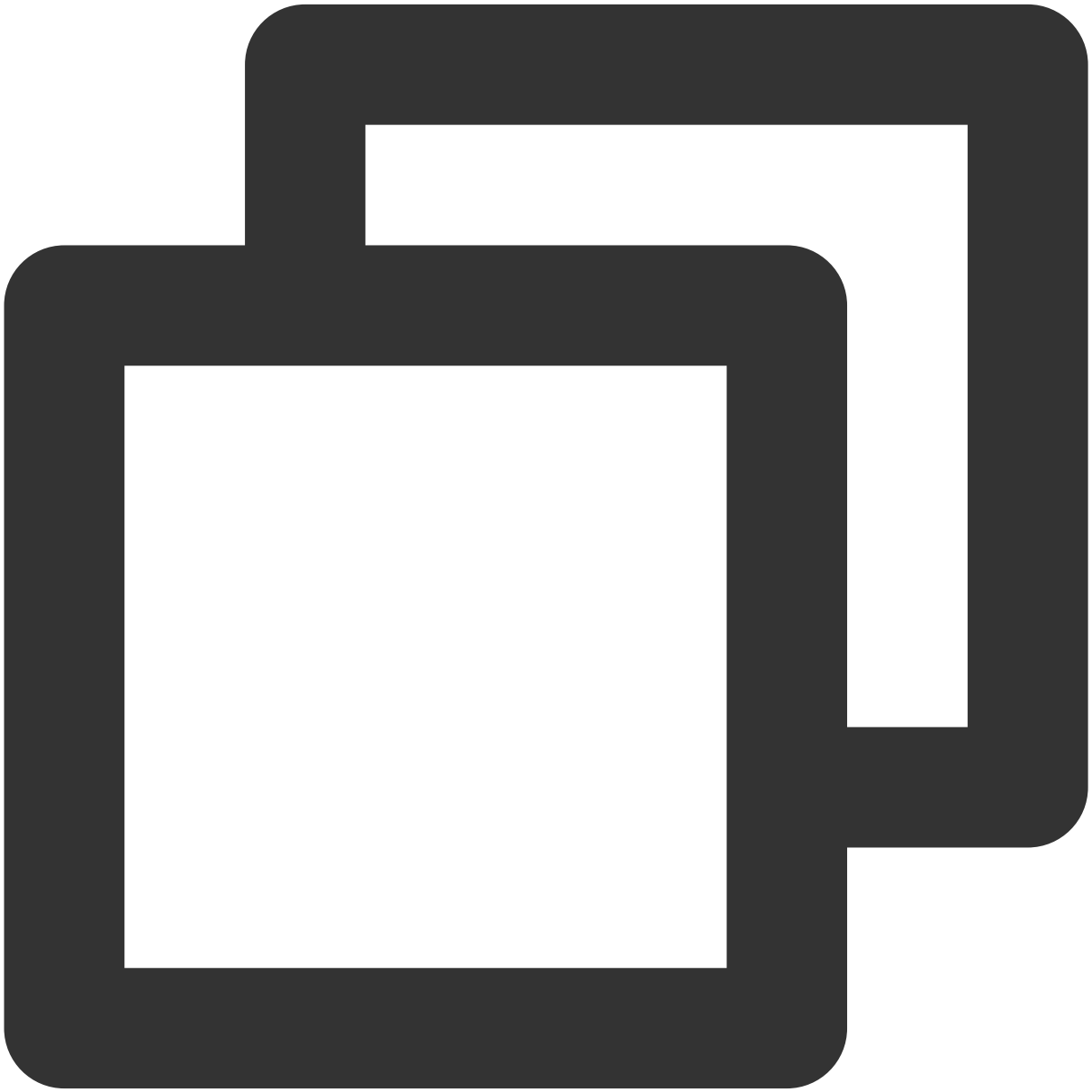
The appGroup identifier represents the App Group shared between the main App and its Extension. It needs to be configured in the main App's Capability as App Groups.

Parameter description:

request	Parameters carried by the UNNotificationServiceExtension callback

appGroupID	The appGroup identifier represents the App Group shared between the main App and its Extension. It needs to be configured in the main App's Capability as App Groups.
callback	typedef void(^TIMPushNotificationExtensionCallback)(UNNotificationContent *content) Statistical function Callback, carrying content information

Usage:



```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentH
```

```
    self.contentHandler = contentHandler;
    NSString * appGroupID = kTIMPushAppGorupKey;
    __weak typeof(self) weakSelf = self;
    [TIMPush onReceiveNotificationRequest:request inAppGroupID:appGroupID callback:
        weakSelf.bestAttemptContent = [content mutableCopy];
        // Modify the notification content here...
        // self.bestAttemptContent.title = [NSString stringWithFormat:@"%@ [modifie
        weakSelf.contentHandler(weakSelf.bestAttemptContent);
    }];
}
```

Flutter

Last updated：2024-06-13 10:21:45

TencentCloudChatPush

class TencentCloudChatPush: Push Plugin Interface Class.

API Overview

Register/Unregister Push Service Interface

After initialization and successful log in to IM, you can register for the push service.

API	Description
registerOnNotificationClickedEvent	Register message click callback in advance.
registerPush	Register Push Service, optionally override the push message from interface parameter json.
unRegisterPush	Unregister the offline push service when logging out of the IM account.

FCM Custom Ringtone Configuration Interface

Once configured, the custom ringtone is effective. The sender must include the channelId in the offline message.

API	Description
configFCMPrivateRing	To configure the custom ringtone for FCM, it needs to be called before registering for the push service.

Special Configuration Interface for Push Channel

API	Description
setPushBrandId	To specify the vendor channel type for device offline push, it needs to be called before registering for the push service.
getPushBrandId	Obtain the vendor channel type currently in use for device offline push.
checkPushStatus	After completing the access configuration for each vendor, you can use this interface to test the push status on the corresponding vendor

	devices.
setApnsCertificateID	Configure the push certificate ID for APNs separately.
setApplicationGroupID	Configure the Application Group ID for the iOS project.
getAndroidPushToken	Retrieve Android device manufacturer Token.
setAndroidPushToken	Manually specify the Android device manufacturer Token.
setAndroidCustomTIMPushConfigs	Manually replace the default push configuration file timpush-configs.json read by the plugin with a self-defined one. This needs to be invoked before registering the push service.

Interface details

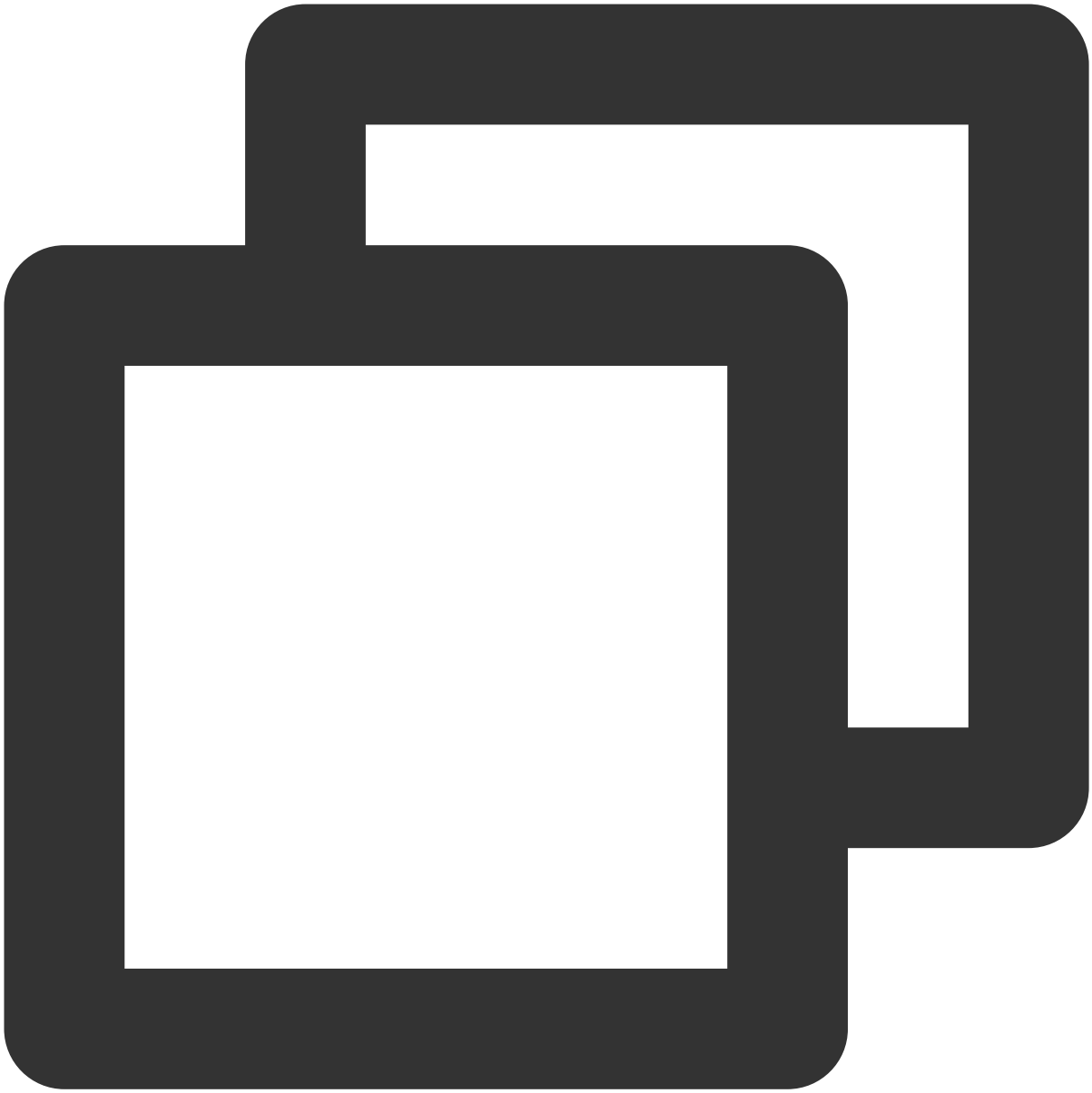
Push Plugin Class

TencentCloudChatPush(): Retrieves the TencentCloudChatPush push plugin instance, which is a Static Singleton. All subsequent steps will invoke methods through this singleton instance.

Member Function Description

registerOnNotificationClickedEvent

Configure Message Click Callback Function.
You can call or adjust the callback function dynamically as needed, or directly pass it during `registerPush` .
Sample Code:



```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
  /// Custom processing
}
TencentCloudChatPush().registerOnNotificationClickedEvent(onNotificationClicked: _o
```

Parameter Description:

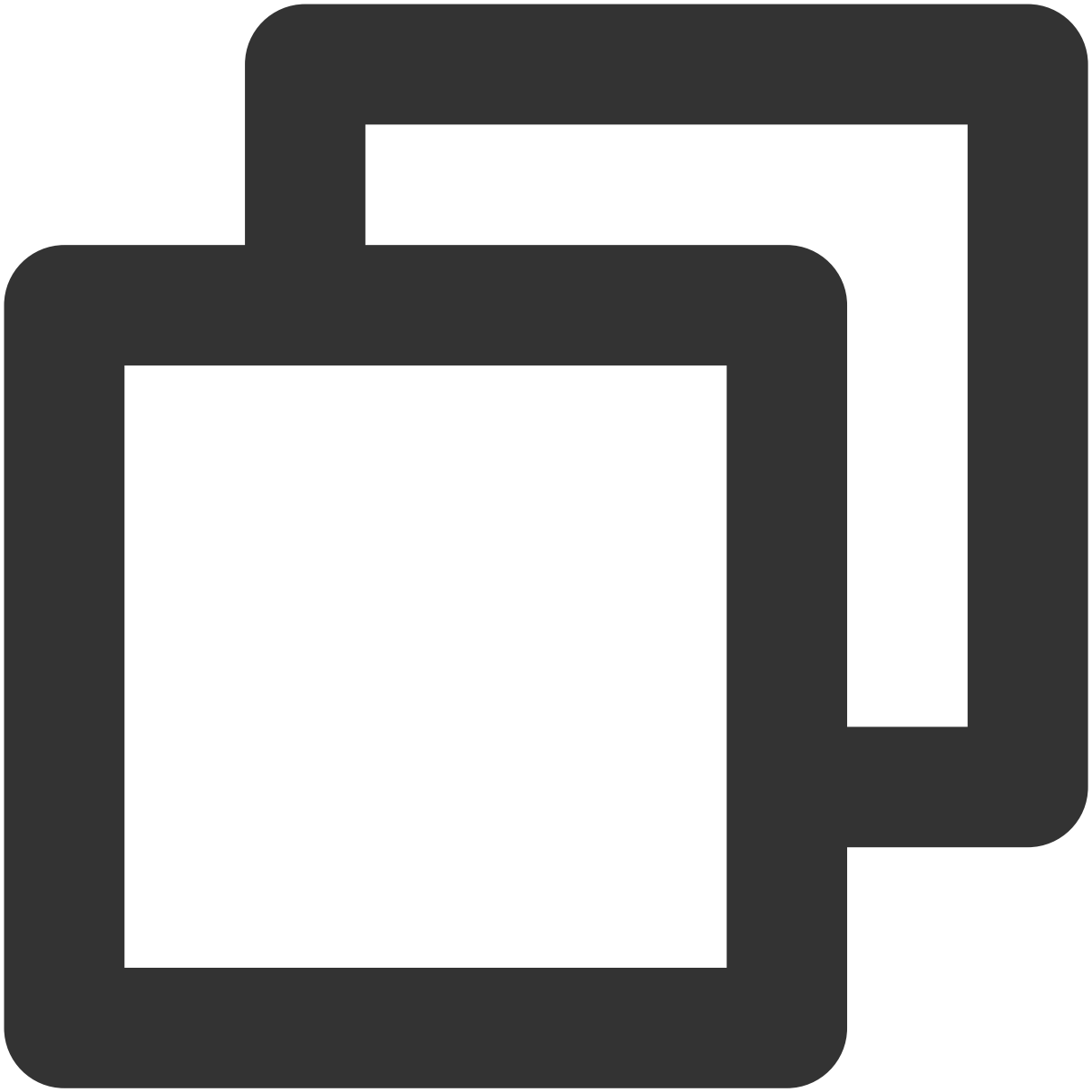
Parameter		Type	Description
onNotificationClicked	ext	String	This is the complete ext information carried by the message,

			specified by the sender. If not specified, a default value is used. You can navigate to the corresponding page by resolving this field.
	userID	String?	This parameter corresponds to userID, automatically attempts to parse the ext Json String, to retrieve the One-on-one Chat counterpart's userID within. **Note:** If you have not defined the ext field yourself, and it is set by default by the SDK or UIKit, you can use the default parsing here. If the attempt to parse fails, it will be null empty.
	groupID	String?	This parameter corresponds to groupID, automatically attempts to parse the ext Json String, to retrieve the Group Chat groupID information within. **Note:** If you have not defined the ext field yourself, and it is set by default by the SDK or UIKit, you can use the default parsing here. If the attempt to parse fails, it will be null empty.

registerPush

Register the offline push service, call after the IM account logs in successfully.

Sample Code:

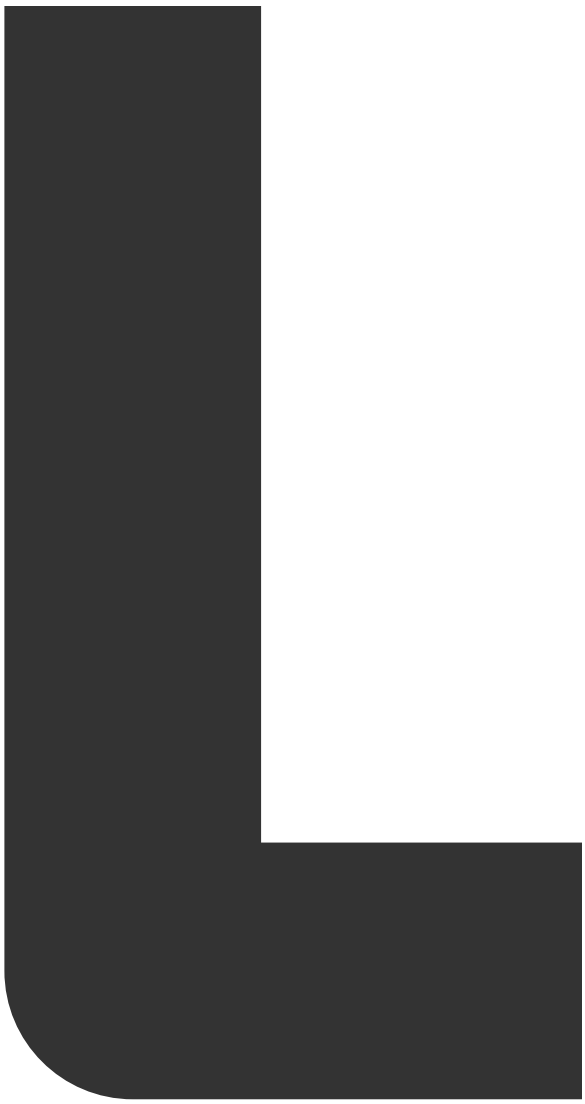


```
TencentCloudChatPush().registerPush(
  onNotificationClicked: _onNotificationClicked,
  androidPushOEMConfig: "can be left null",
  apnsCertificateID: 0,
);
```

Parameter Description:

Parameter		Type	Description
onNotificationClicked	ext	String	This is the complete ext information carried by the message, spec

	userID	String?	This parameter corresponds to userID, automatically attempts to p **Note:** If you have not defined the ext field yourself, and it is set by defaul
	groupID	String?	This parameter corresponds to groupID, automatically attempts to **Note:** If you have not defined the ext field yourself, and it is set by defaul
androidPushOEMConfig		String	If the timpush-configs.json configuration file has been imported, th Optional Android-side parameters json:



```
androidPushOEMConfig: {
      // huawei
      "huaweiPushBussinessId": "",
      // Certificate ID assigned after upload
      "huaweiBadgeClassName": "", // Badge pa
      // xiaomi
      "xiaomiPushBussinessId": "",// Certific
      "xiaomiPushAppId": "",// Application API

      // meizu
      "meizuPushBussinessId": "", // Certific
      "meizuPushAppId": "",// Application APP

      // vivo
```

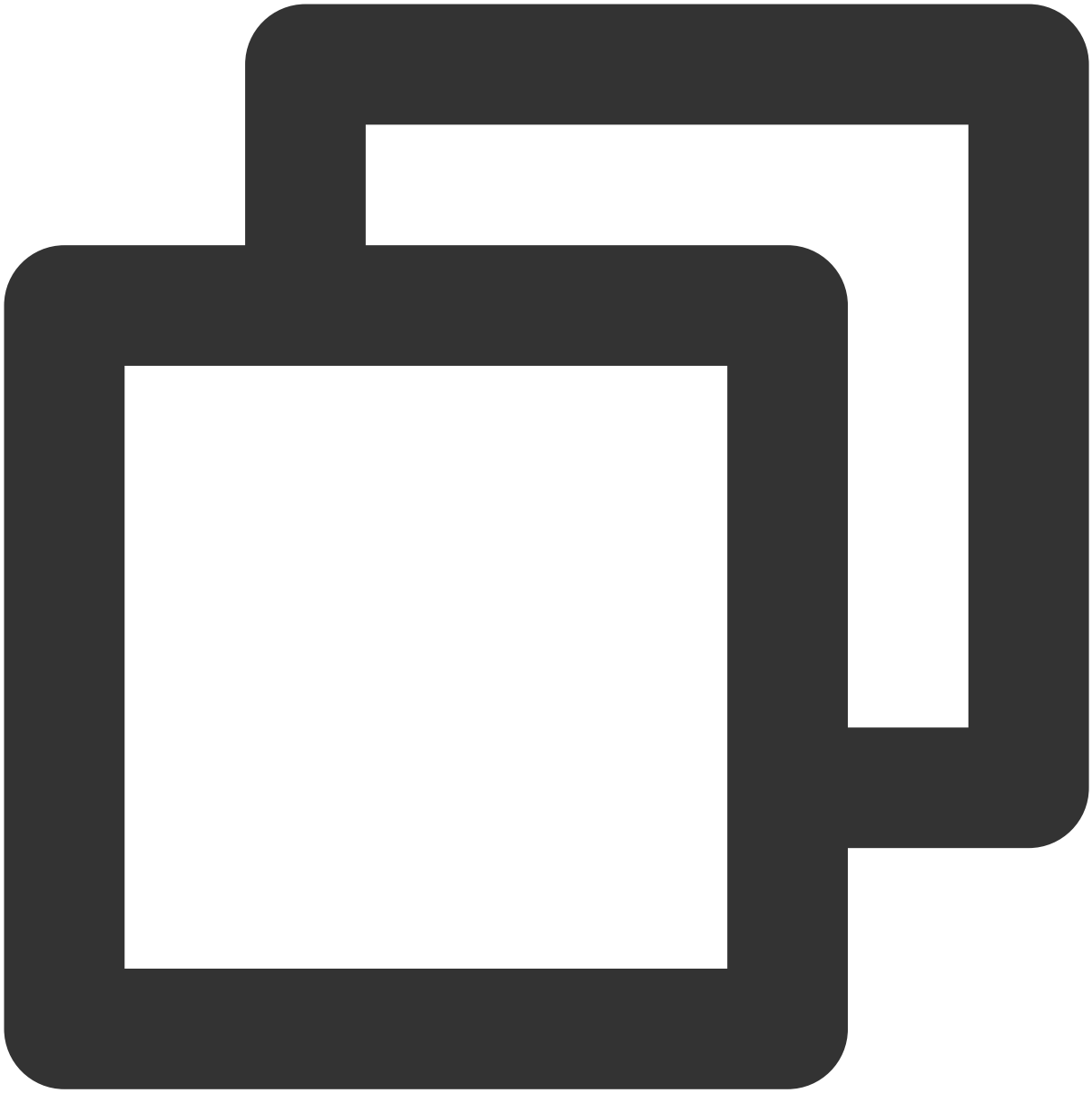
		"vivoPushBussinessId": "", // Certifica
		
		// google
		"fcmPushBussinessId": "", // Certificat
		
		// oppo
		"oppoPushBussinessId": "", // Certifica
		"oppoPushAppKey": "",// Application Appl
		"oppoPushAppSecret": "",// Application
		
		// honor
		"honorPushBussinessId": "", // Certific
		}

apnsCertificateID	int	If setApnsCertificateID method has already been called separatel

unRegisterPush

Unregister the offline push service, to be called after the IM account logs out.

Sample Code:

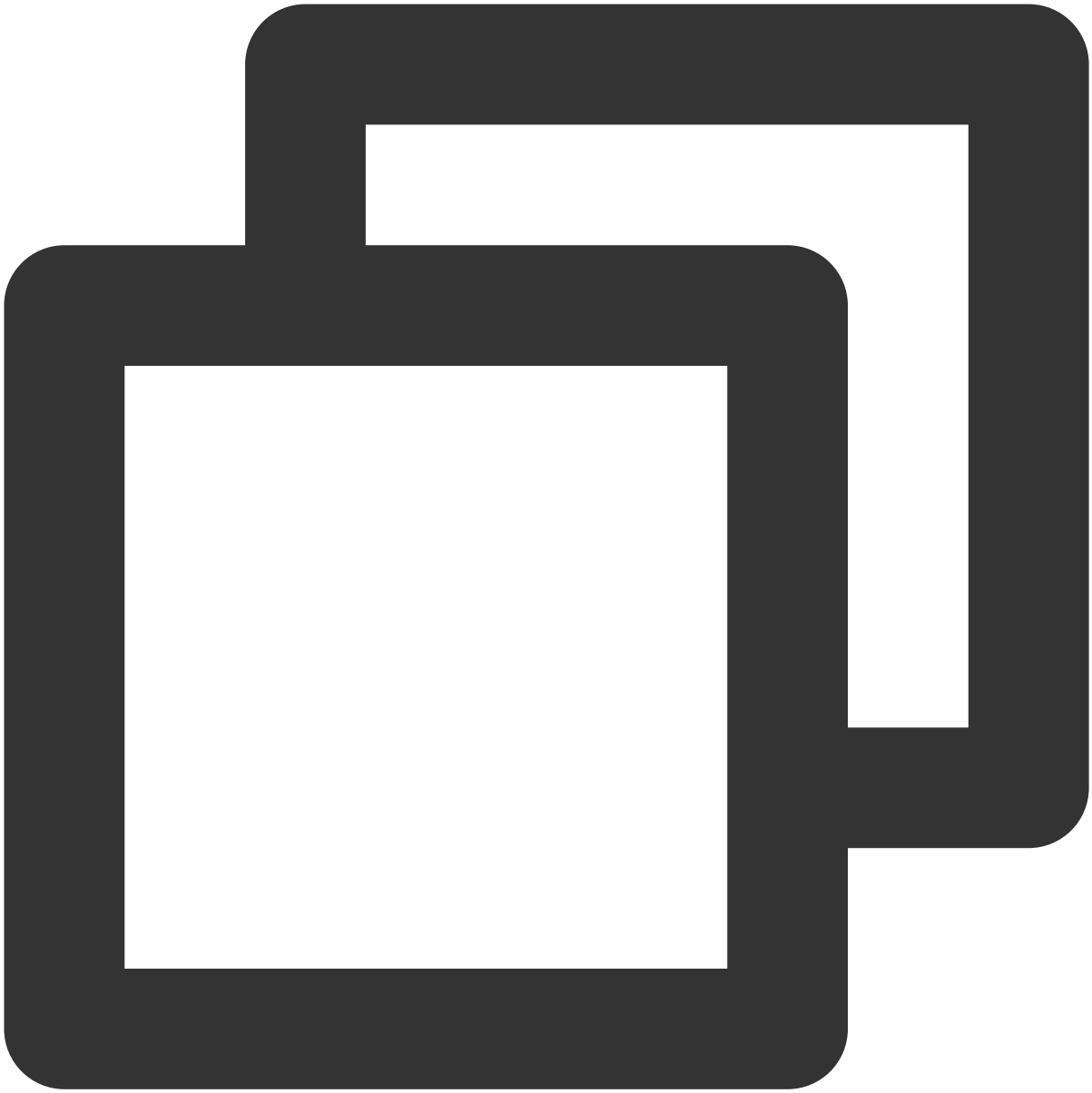


```
TencentCloudChatPush().unRegisterPush();
```

configFCMPrivateRing

To configure the custom ringtone for FCM, it needs to be called before registering for the push service.

Sample Code:



```
TencentCloudChatPush().configFCMPrivateRing(channelId: channelId, ringName: ringNam
```

Parameter Description:

Parameter Name	Type	Description
channelId	String	FCM channel custom notification bar's channel ID, unique within the app.
ringName	String	FCM channel's custom push ringtone name, located in the raw directory

		and does not require a file suffix.
enable	bool	Setting whether the offline push prompt ringtone uses a custom ringtone.

Note:

Once configured, the custom ringtone is effective. The sender must include the channelId in the offline message.

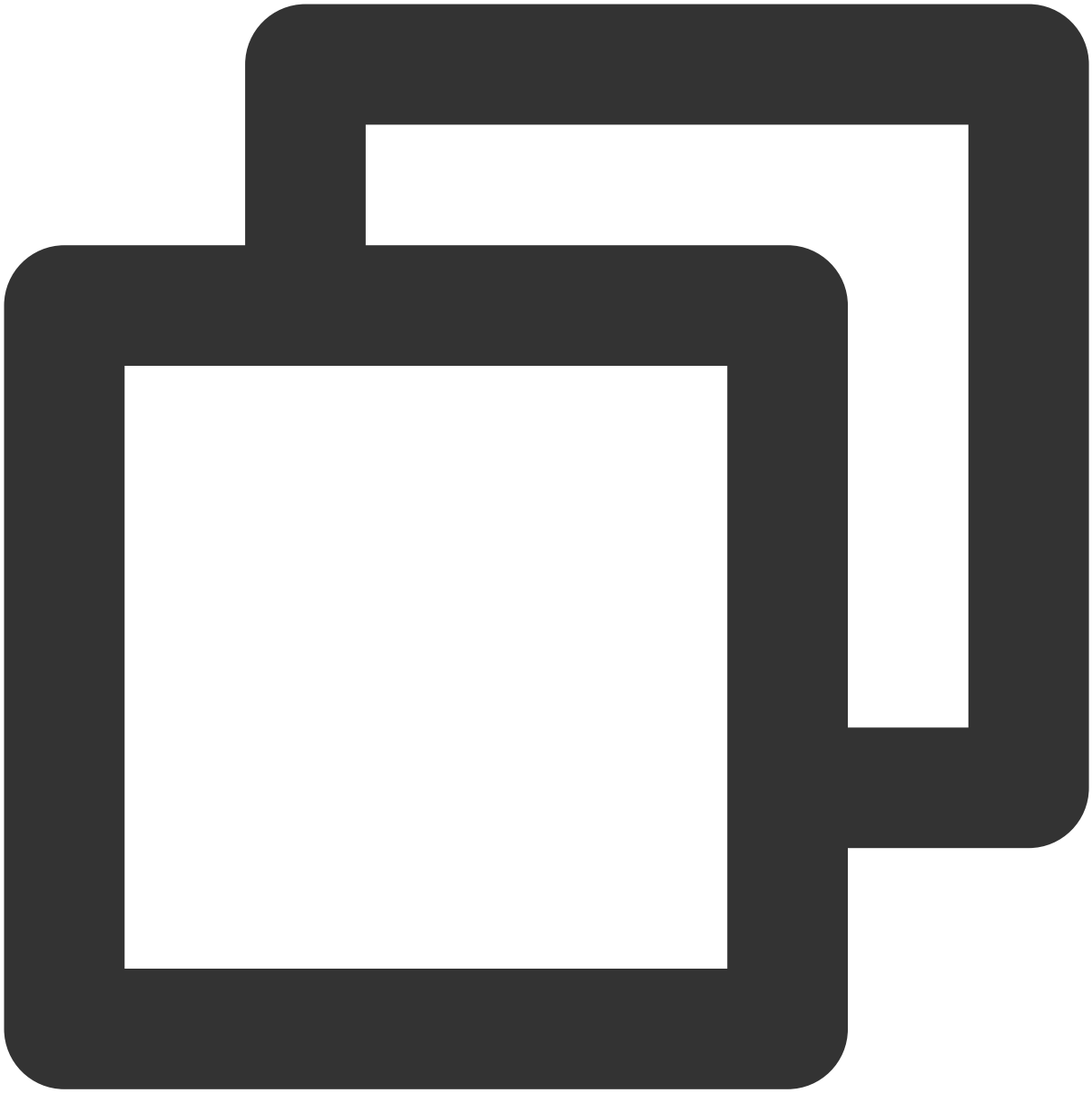
setPushBrandId

To specify the vendor channel type for device offline push, it needs to be called before registering for the push service.

Note:

This interface allows specifying the use of a manufacturer's push channel type, for example, specifying the use of the FCM channel for push on Xiaomi devices abroad with setPushBrandId(TencentCloudChatPushBrandID.FCM). Generally, there's no need to specify the channel type, as the component will automatically identify the device manufacturer category to register and use the corresponding manufacturer channel.

Sample Code:



```
TencentCloudChatPush().setPushBrandId(brandID: brandID);
```

Parameter description

:

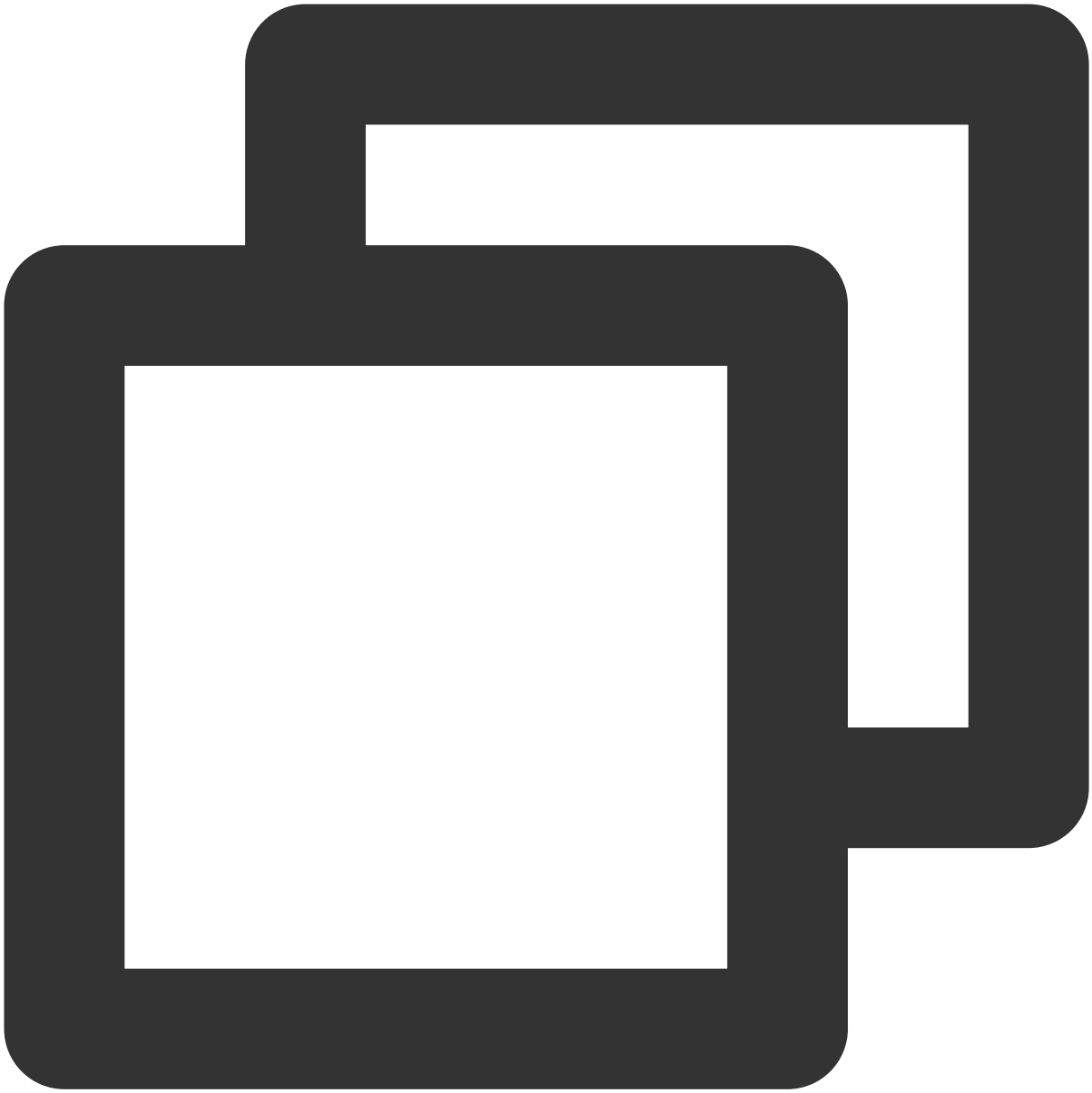
Parameter	Description	
brandID	Vendor	Device Type
	XiaoMi	TencentCloudChatPushBrandID.XiaoMi

HuaWei	TencentCloudChatPushBrandID.HuaWei
FCM	TencentCloudChatPushBrandID.FCM
Meizu	TencentCloudChatPushBrandID.Meizu
Oppo	TencentCloudChatPushBrandID.Oppo
Vivo	TencentCloudChatPushBrandID.Vivo
Honor	TencentCloudChatPushBrandID.Honor

getPushBrandId

Obtain the vendor channel type currently in use for device offline push.

Sample Code:



```
final res = await TencentCloudChatPush().getPushBrandId();
if(res.code == 0){
  final TencentCloudChatPushBrandID brandID = res.data;
}
```

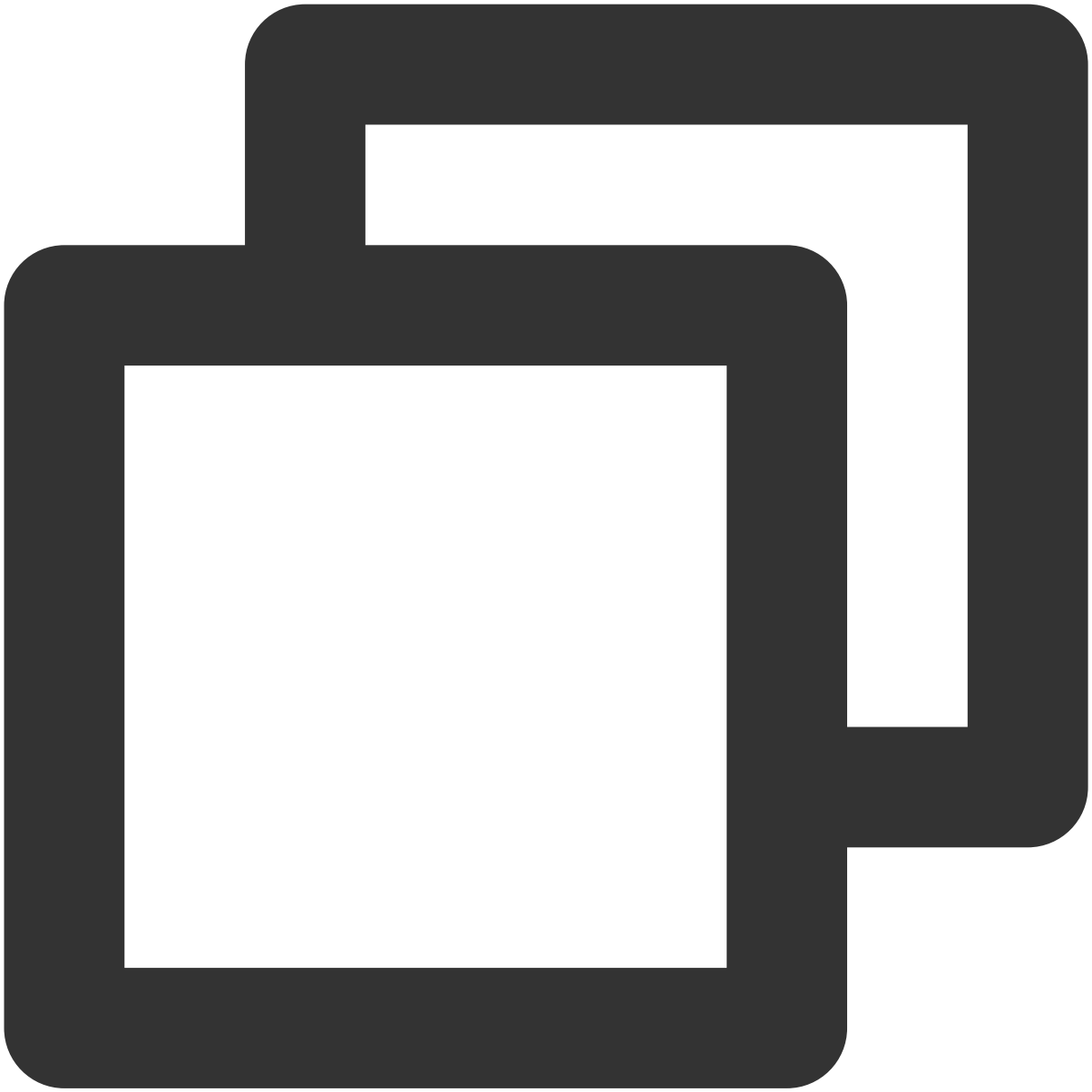
Return Type:

Future<TencentCloudChatPushResult<TencentCloudChatPushBrandID>>

checkPushStatus

After completing the access configuration for each vendor, you can use this interface to test if the corresponding vendor devices can receive pushes.

Sample Code:



```
final res = await TencentCloudChatPush().checkPushStatus(brandID: 2002);
if(res.code == 0){
  final status = res.data;
}
```

Parameter Description:

Parameter Name	Type	Description
brandID	TencentCloudChatPushBrandID	BrandID Definition as shown in the table above.

Return Type:

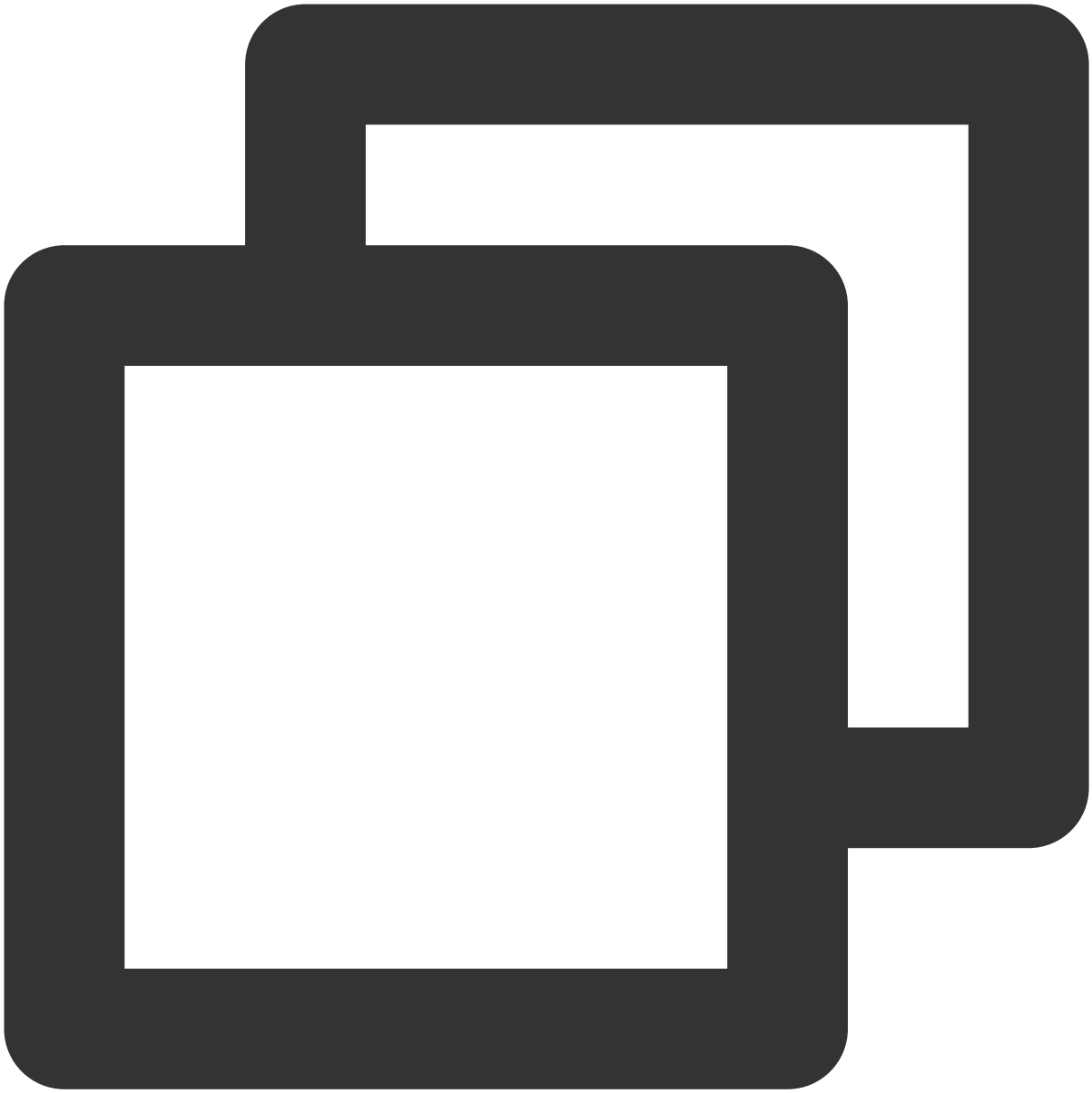
Future<TencentCloudChatPushResult<String>>

If true, push notifications can be successfully sent.

setApnsCertificateID

Separately configure the APNs Push Certificate ID. You can call or dynamically adjust the Certificate ID as needed, or directly pass it during `registerPush` .

Sample Code:



```
TencentCloudChatPush().setApnsCertificateID(apnsCertificateID: 0);
```

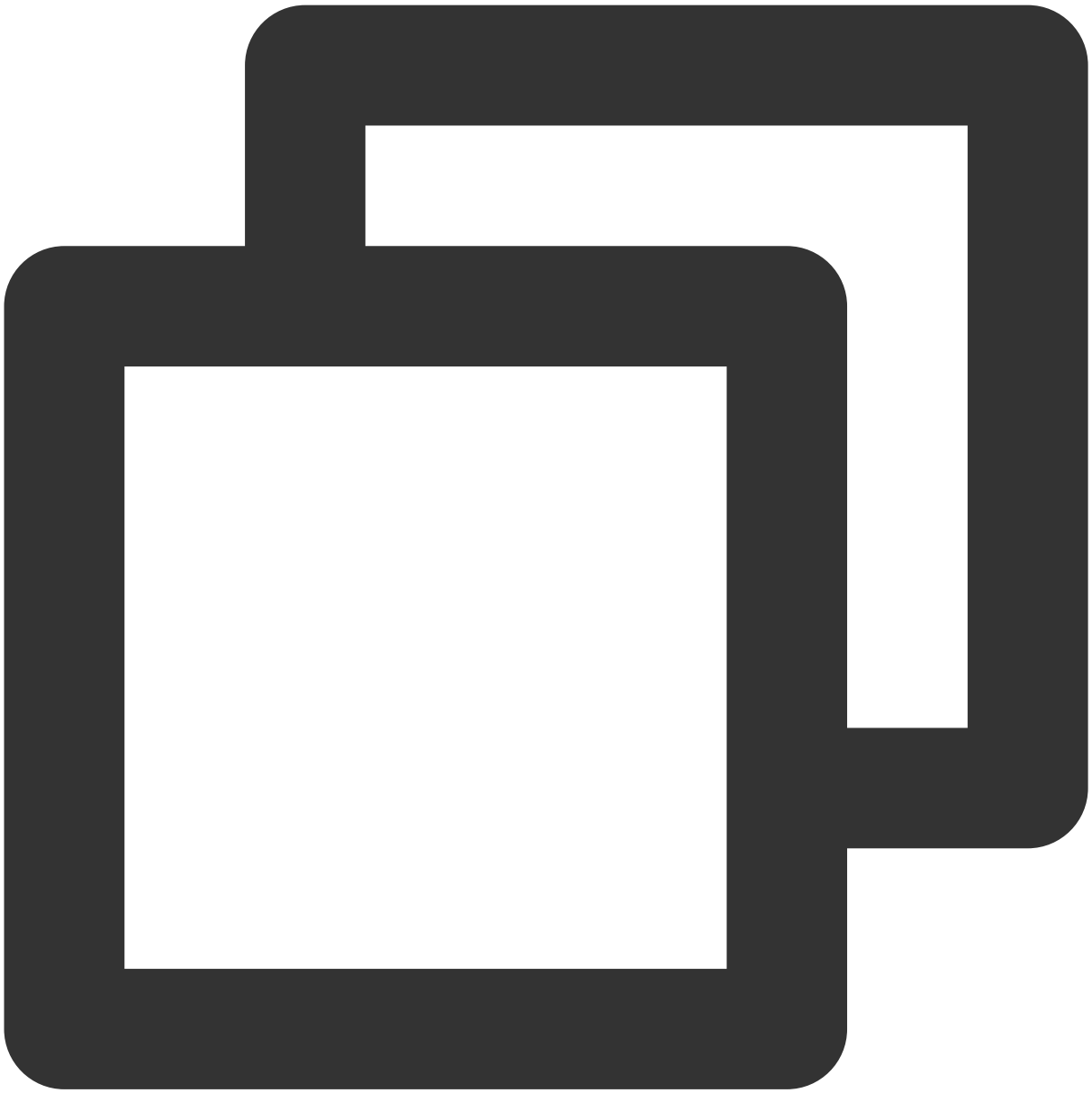
Parameter Description:

Parameter Name	Type	Description
apnsCertificateID	int	The Certificate ID assigned to the APNs certificate in the Tencent Cloud IM Console.

setApplicationGroupID

Configure the Application Group ID for the iOS project.

Sample Code:



```
TencentCloudChatPush().setApplicationGroupID(applicationGroupID: "");
```

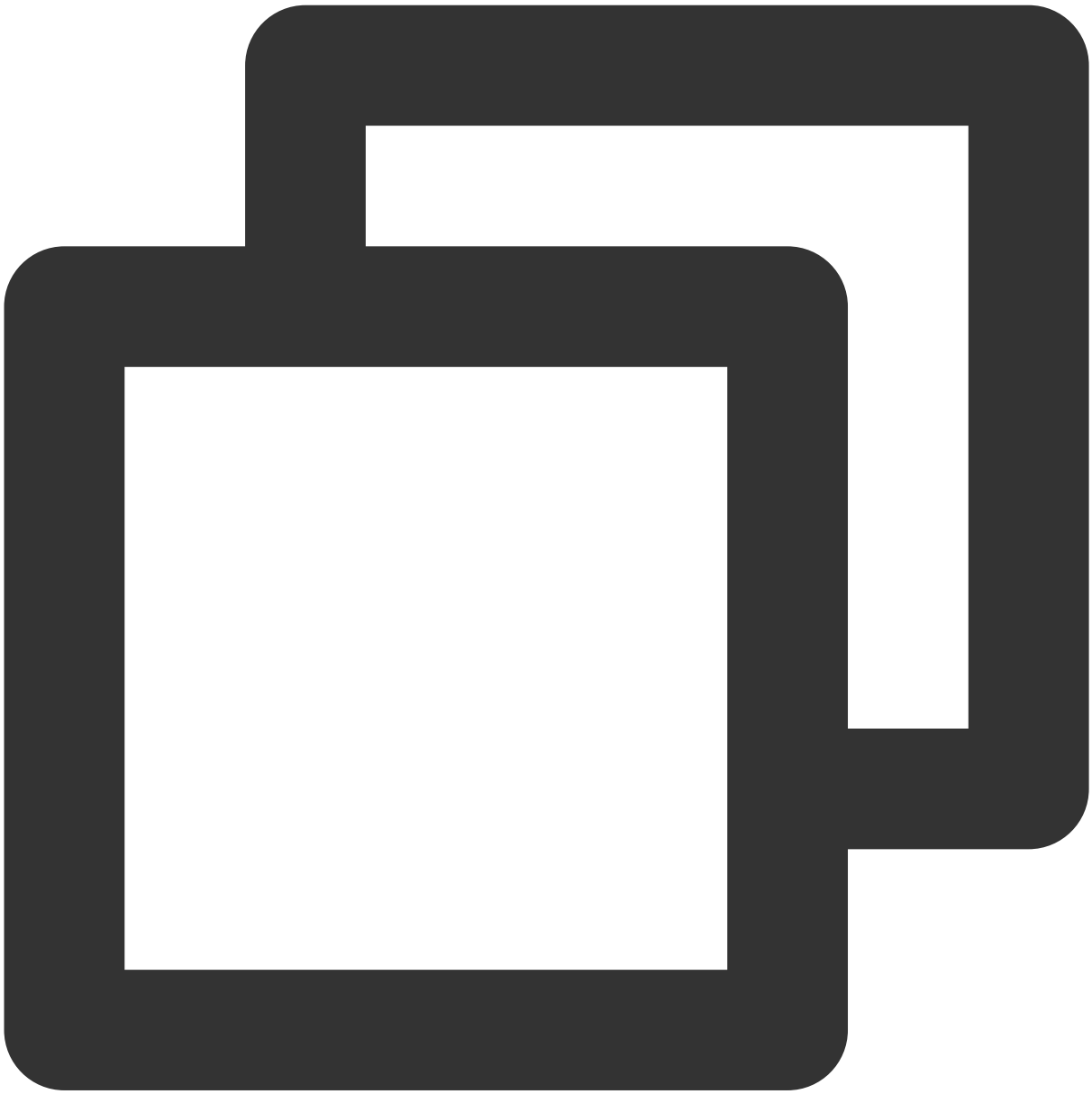
Parameter Description:

Parameter Name	Type	Description
applicationGroupID	String	The format is: group + [Main Bundle ID] + key.

getAndroidPushToken

Retrieve Android device manufacturer Token.

Sample Code:



```
TencentCloudChatPush().getAndroidPushToken();
```

Return Type:

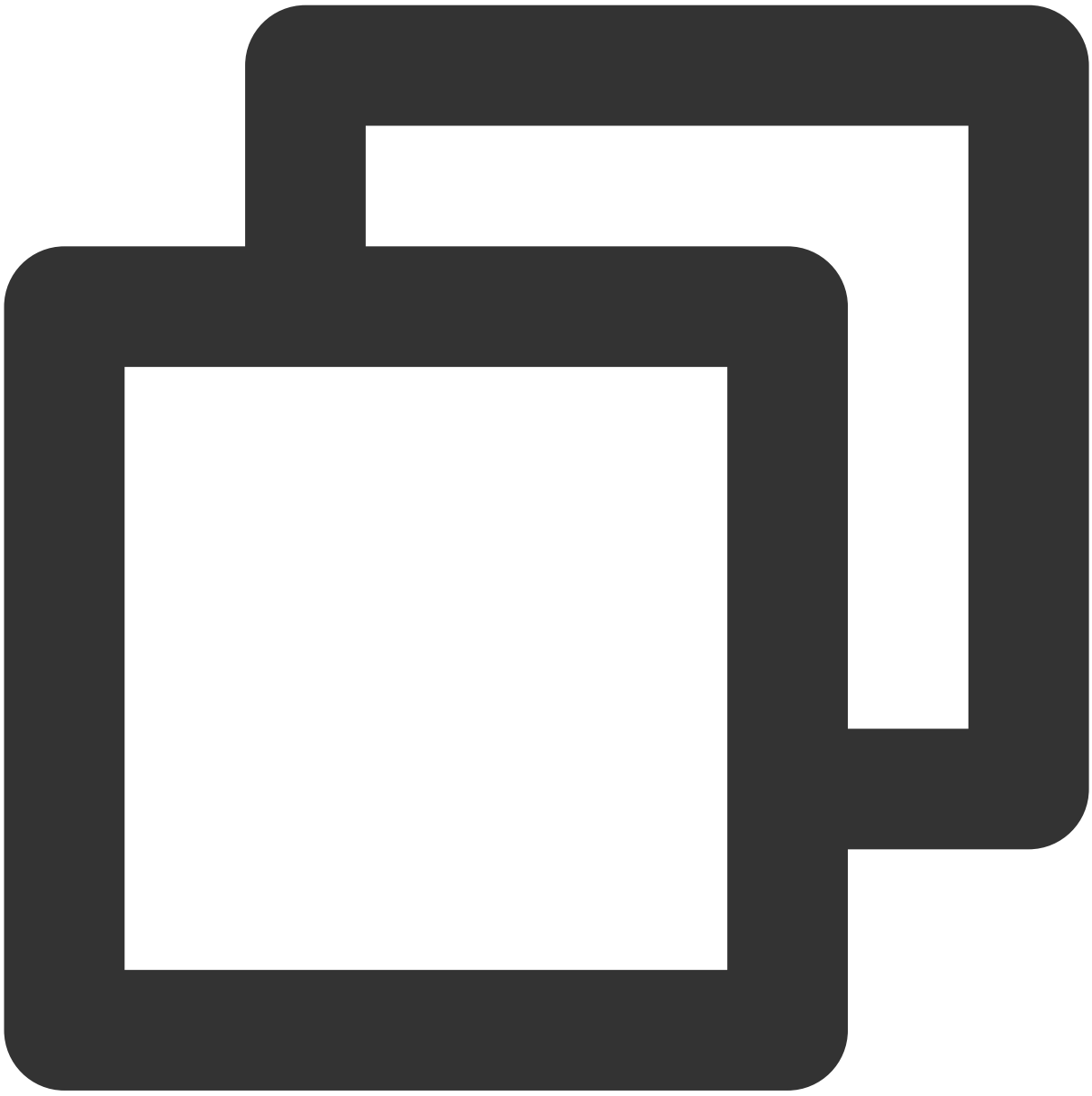
Future<TencentCloudChatPushResult<String>>

The returned String type data is the vendor's push Token..

setAndroidPushToken

Manually specify the Android device manufacturer Token.

Sample Code:



```
TencentCloudChatPush().setAndroidPushToken(businessID: 10000, pushToken: "pushToken
```

Parameter Description:

Parameter Name	Type	Description

businessID	String	Push certificate ID, available from the Tencent Cloud IM console, under the push certificate card.
pushToken	String	Obtain the vendor's push Token in your own way.

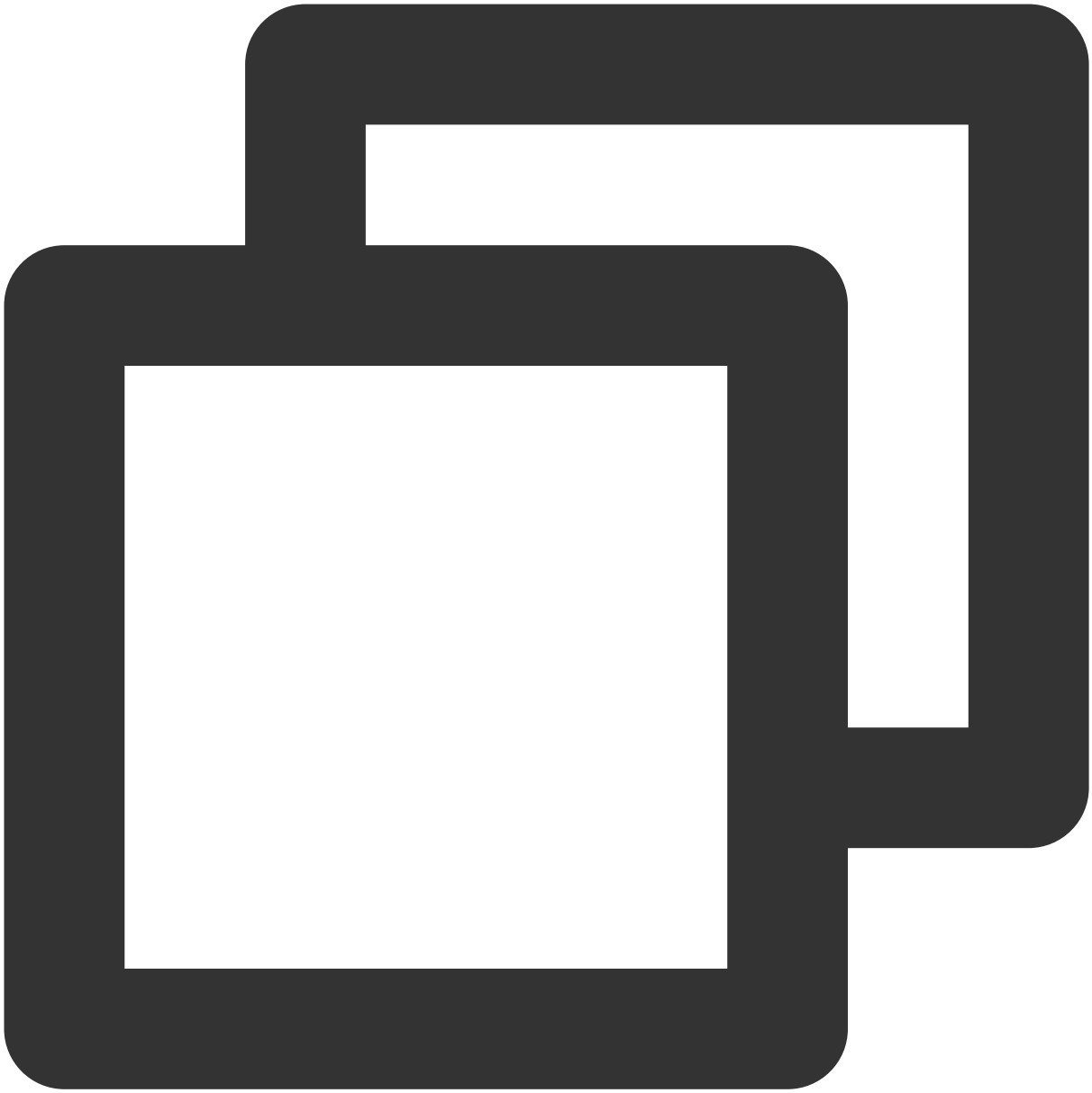
setAndroidCustomTIMPushConfigs

Manually replace the default push configuration file timpush-configs.json read by the plugin with a custom one. This needs to be invoked before registering the push service (registerPush).

Note:

Mainly used for dynamically switching different configuration files for push registration in various environments, for example: integrating and testing push features under different configuration files in production and testing environments.

Sample Code:



```
TencentCloudChatPush().setAndroidCustomTIMPushConfigs(configs: "");
```

Parameter Description:

Parameter Name	Type	Description
configs	String	The name of the custom configuration file, the path must remain unchanged: "Engineering Root Directory/android/app/src/assets/".

All Staff/Tag Push
Pushing to All/Tagged Users

Last updated：2024-07-23 17:57:52

Pushing to all/tagged users not only supports sending specific content to all users, but also can send personalized content to specific user groups based on tags and attributes, such as member events and regional notifications. This helps effectively attract, convert, and activate new users. It also supports pushing delivery reports and self-service troubleshooting tools. Refer to Feature Overview for the effect.

Note:

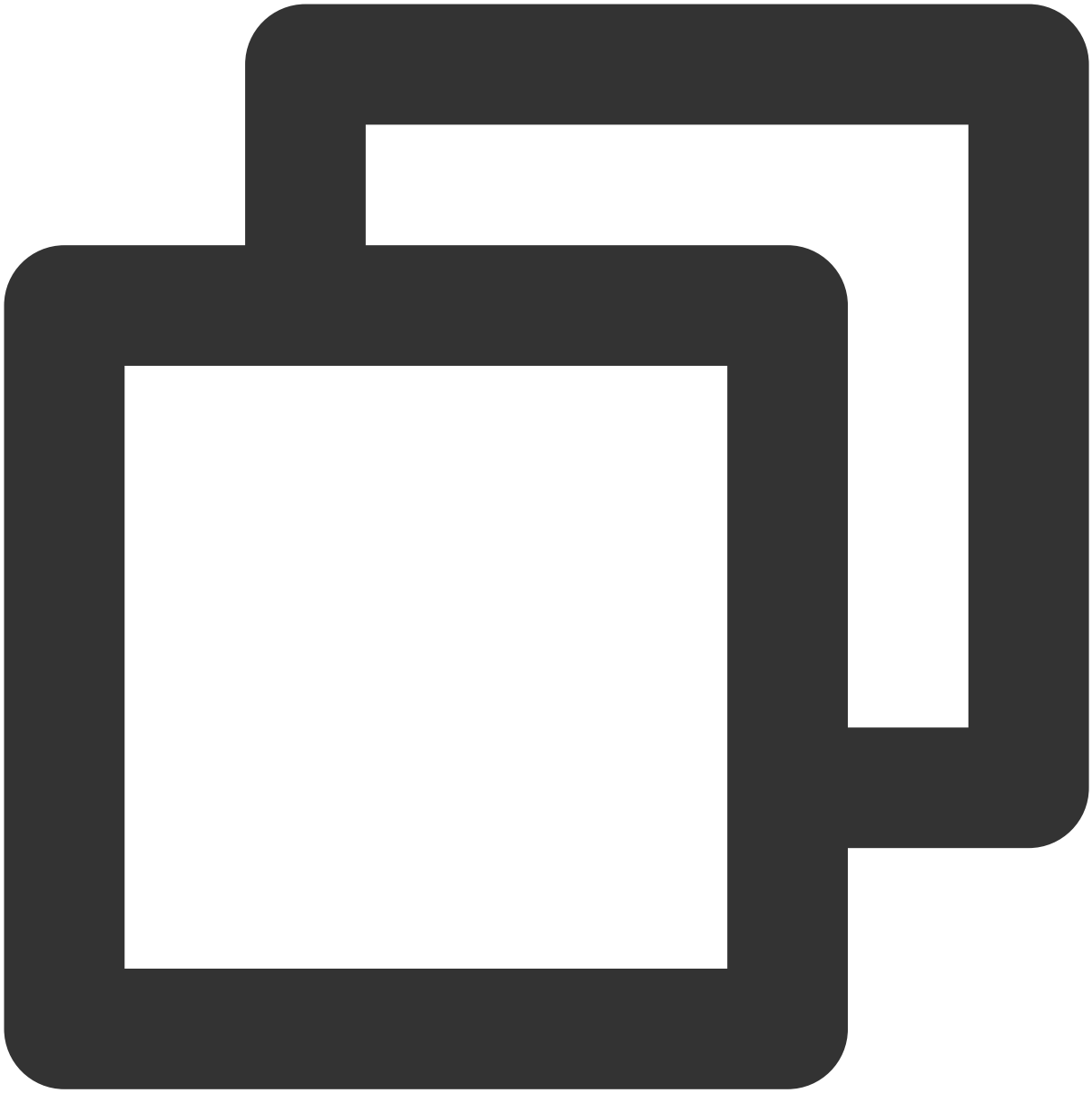
The account must have been logged in or manually imported before it can receive push messages.

Feature Overview

Supports sending push messages to all users.
Supports sending push messages based on user attributes.
Supports sending push messages based on user tags.
The administrator pushes a message, and the recipient sees that the sender of the message is the administrator.
The administrator specifies an account to push a message to other accounts, and the recipient sees that the sender is not the administrator, but the account specified by the administrator.
Supports roaming, with the same storage duration as regular messages.
Due to the large number of accounts that need to be delivered for pushing to all/tagged users, it takes a certain amount of time (depending on the total number of accounts) to complete the delivery of all accounts.
By setting OnlineOnlyFlag to 1, push messages can only be sent without saving sessions, roaming, and unread messages.

API Call Description

Sample Request URL



```
https://xxxxxx/v4/timpush/push?usersig=xxx&identifier=admin&sdkappid=88888888&rando
```

Request Parameters

Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/push	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Call Frequency

This API includes pushing to all users and pushing to users by attribute or tag. By default, it can be called 100 times per day. The interval between two pushes must be greater than 1 second.

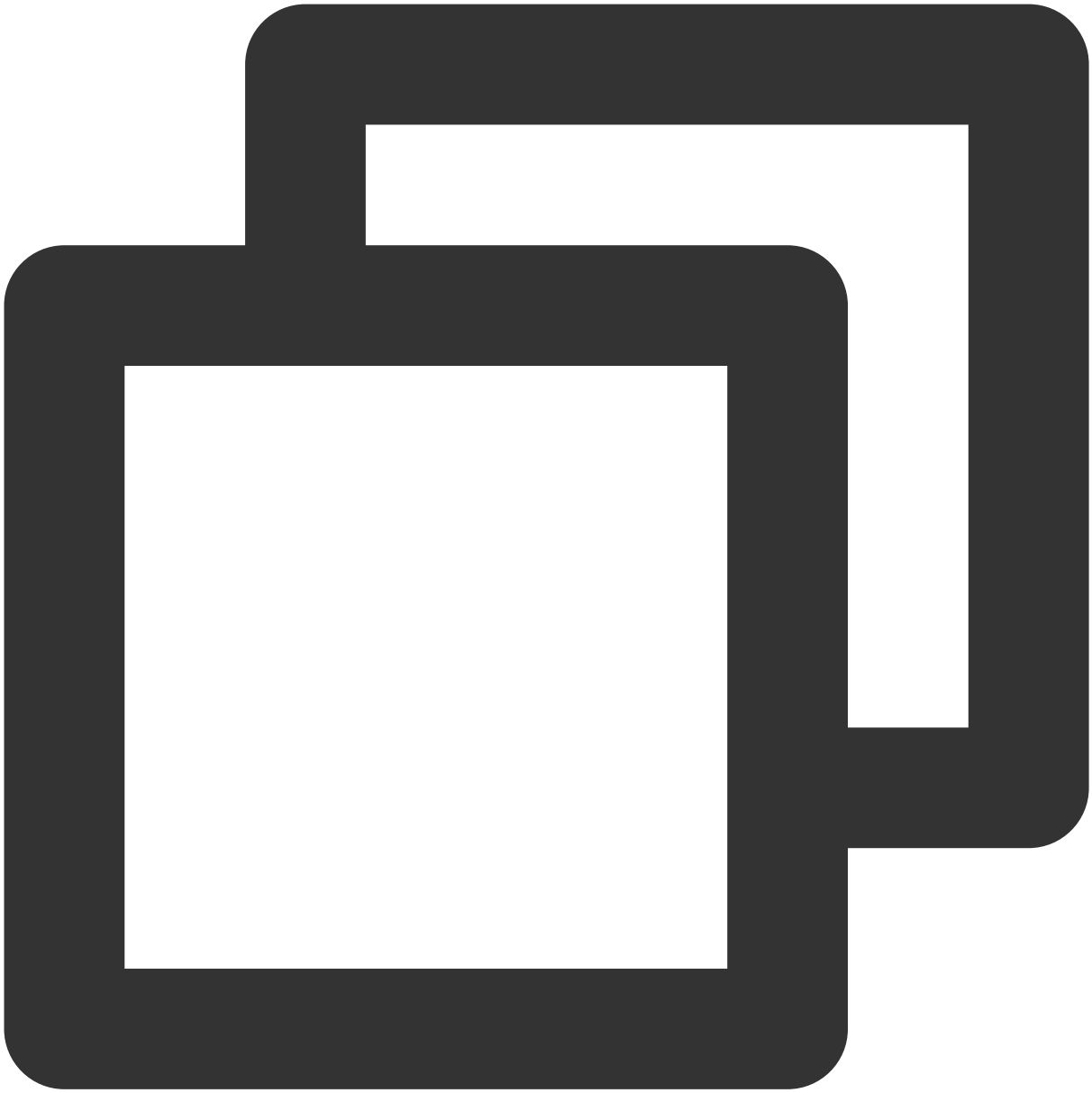
Note:

By default, the API can be called 100 times per day for free. For each additional call per day, the cost will increase by 10 USD per day. If you need to adjust the frequency of API calls, go to the Chat console to modify it.

Sample Request Packets

Pushing Messages to All Users

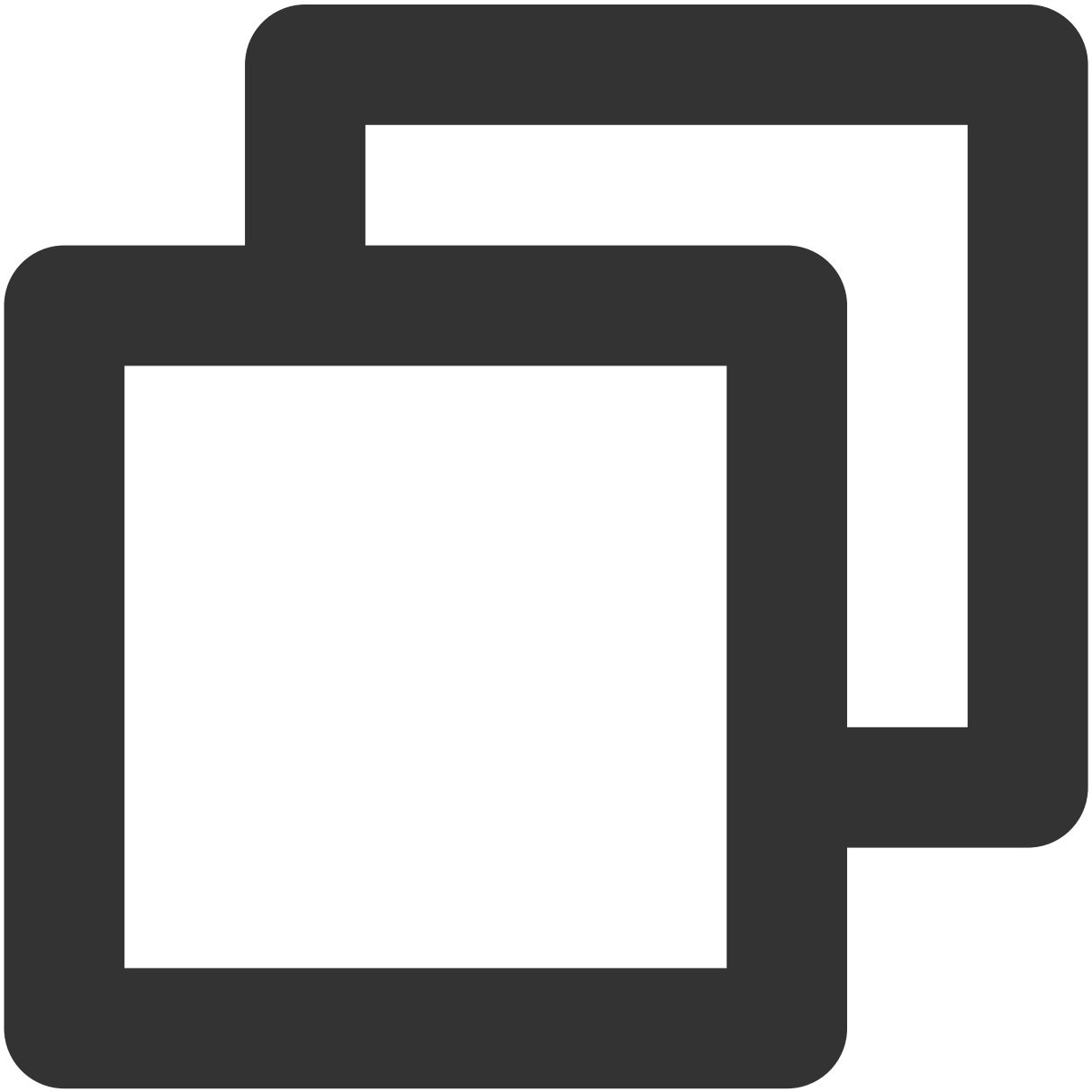
The administrator pushes messages to all users:



```
{
  "From_Account": "admin",
  "MsgRandom": 3674128,
  "OnlineOnlyFlag": 0, // 0: Saves roaming and unread messages. In this case, the p
                       // 1: Does not save roaming or unread messages. In this case
  "MsgBody": [
      {
          "MsgType": "TIMTextElem",
          "MsgContent": {
              "Text": "hi, beauty"
          }
```

```
        }
    ],
    "OfflinePushInfo": {
        "PushFlag": 0,  // 0: Enables offline push; 1: Disables offline push.
        "Title": "offline push title",
        "Desc": "offline push content"
    }
}
```

The administrator specifies an account for pushing messages to all users. In the example, the sender account is `xiaoming`:

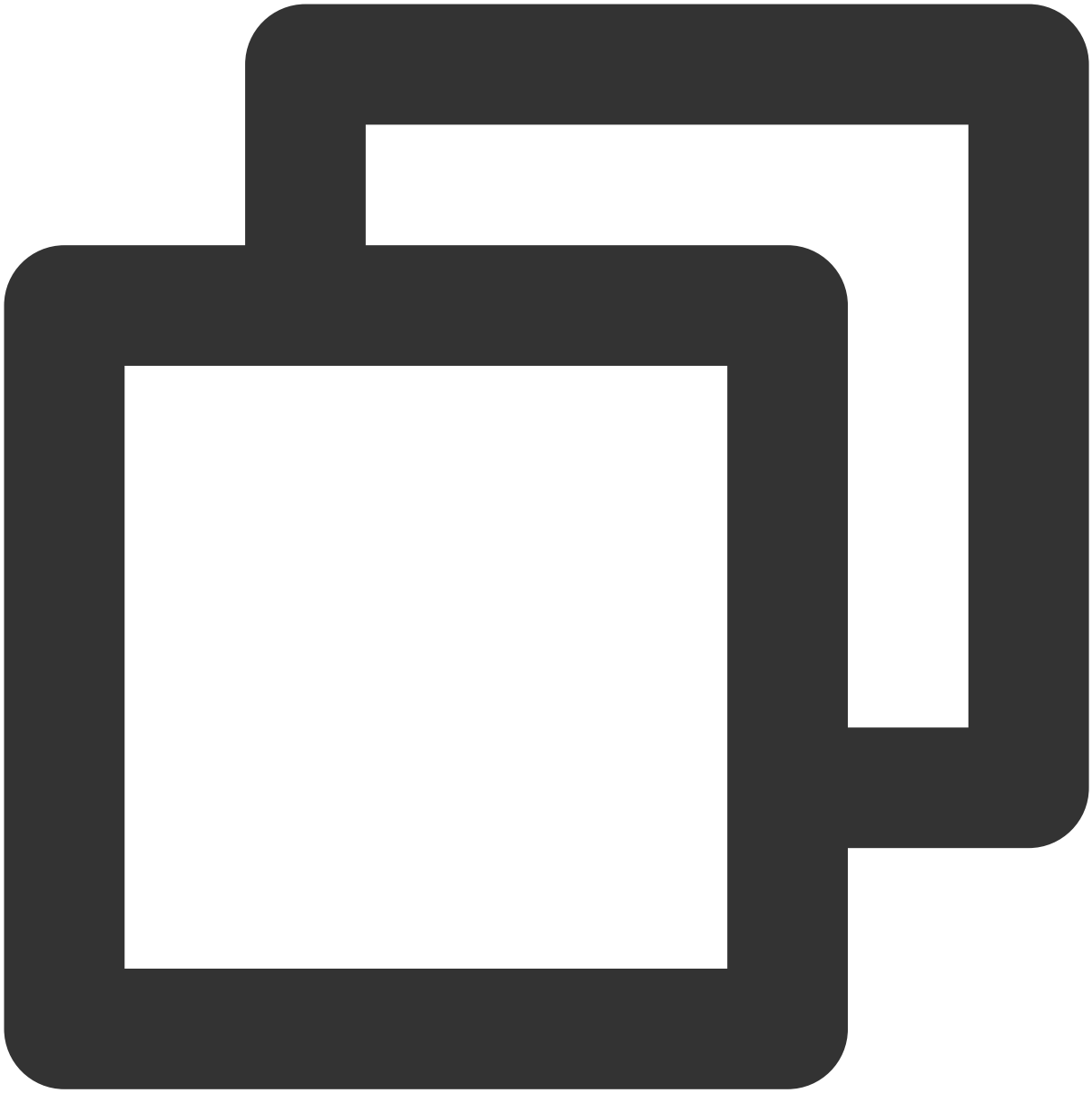


```
{
  "From_Account": "xiaoming",
  "MsgRandom": 3674128,
  "OnlineOnlyFlag": 0, // 0: Saves roaming and unread messages. In this case, the p
                       // 1: Does not save roaming or unread messages. In this case
  "MsgBody": [
      {
          "MsgType": "TIMTextElem",
          "MsgContent": {
              "Text": "hi, beauty"
          }
      }
  ],
  "OfflinePushInfo": {
      "PushFlag": 0,  // 0: Enables offline push; 1: Disables offline push.
      "Title": "offline push title",
      "Desc": "offline push content"
  }
}
```

Note:

From_Account is the message sender account and can be specified as any existing account. If no sender is specified or the specified sender does not exist, the API caller account will be taken by default.

The administrator pushes messages only to online users (online push):



```
{
"From_Account": "xiaoming",
"MsgRandom": 3674128,
"OnlineOnlyFlag": 1, // 0: Saves roaming and unread messages. In this case, the p
                     // 1: Does not save roaming or unread messages. In this case
"MsgBody": [
    {
        "MsgType": "TIMTextElem",
        "MsgContent": {
            "Text": "hi, beauty"
        }
```

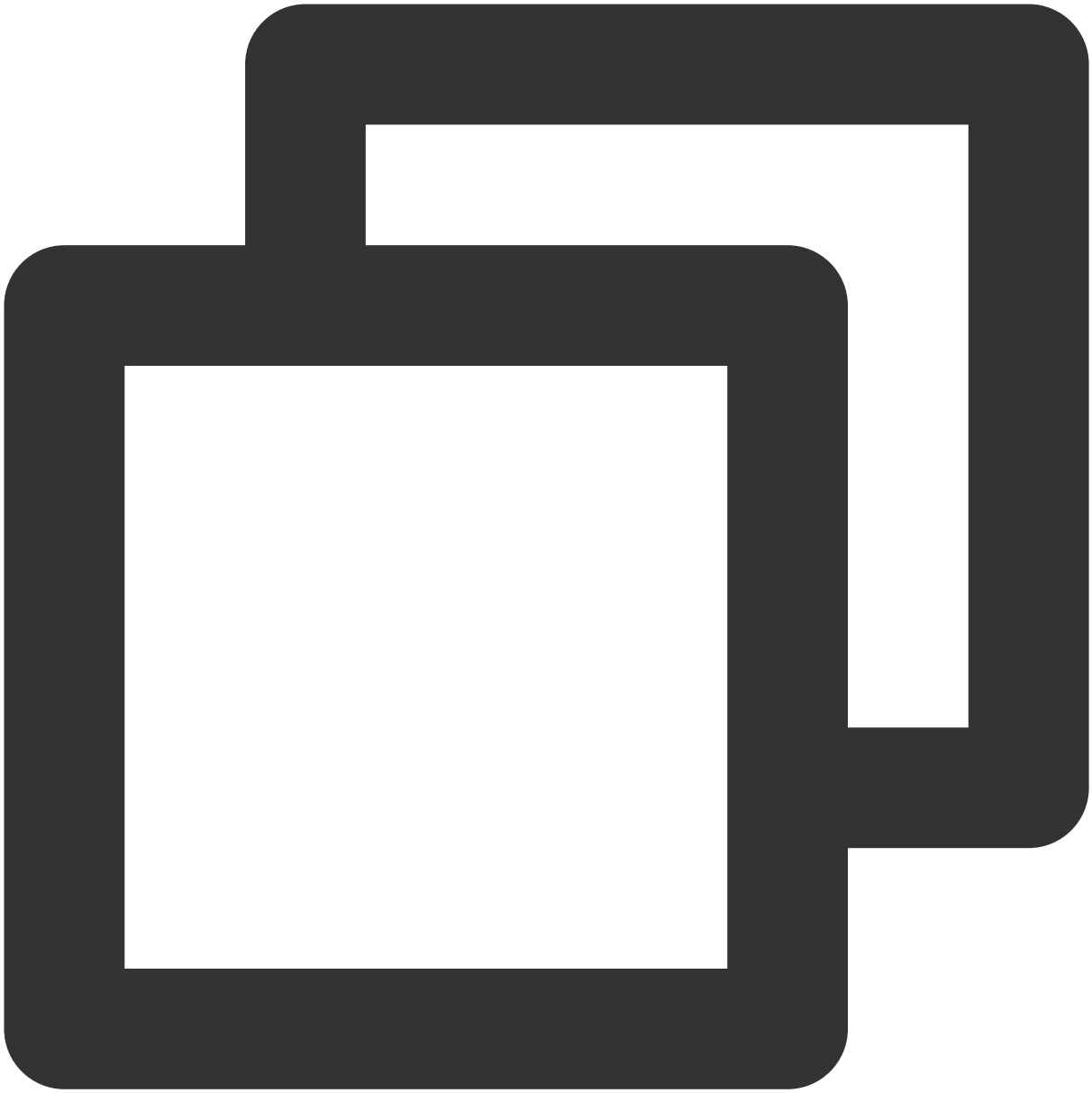
```
        }
    ],
    "OfflinePushInfo": {
        "PushFlag": 1,  // 0: Enables offline push; 1: Disables offline push.
    }
}
```

Note:

Setting OnlineOnlyFlag to 1 indicates only push (online push+offline push). Setting OfflinePushInfo.PushFlag to 1 indicates no offline push. Therefore, the above example indicates that messages are pushed only to online users.

Pushing Messages by User Tag

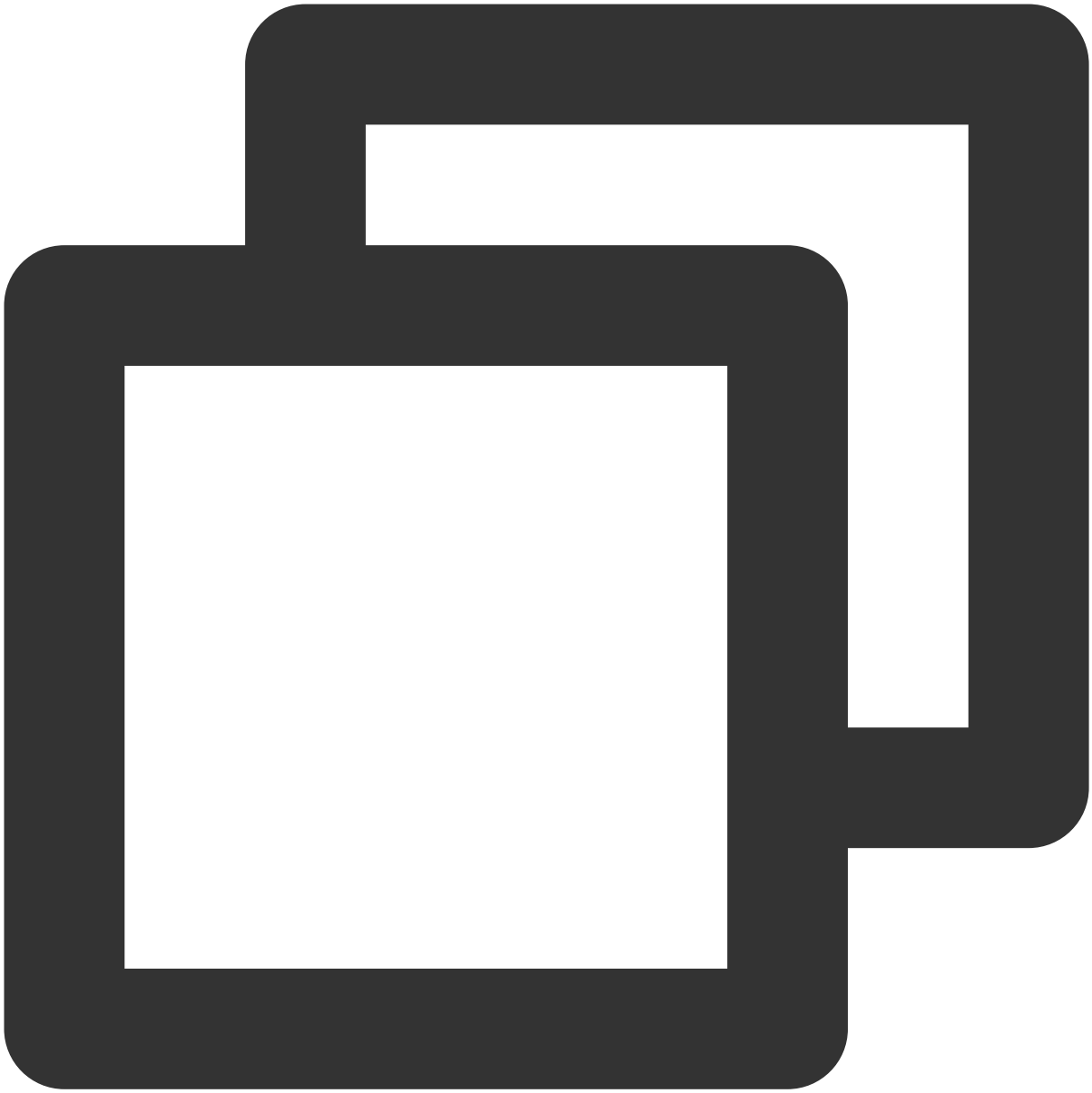
The administrator pushes messages to users tagged with "Stock A" and "Stock B":



```
{
  "From_Account": "admin",
  "MsgRandom": 124032,
  "OnlineOnlyFlag": 0, // 0: Saves roaming and unread messages. In this case, the p
                       // 1: Does not save roaming or unread messages. In this case
  "Condition": {
      "TagsAnd": ["Stock A","Stock B"]
  },
  "MsgBody": [
      {
          "MsgType": "TIMTextElem",
```

```
            "MsgContent": {
                "Text": "hi, beauty"
            }
        }
    ],
    "OfflinePushInfo": {
        "PushFlag": 0,  // 0: Enables offline push; 1: Disables offline push.
        "Title": "offline push title",
        "Desc": "offline push content"
    }
}
```

The administrator pushes messages to users tagged with "Stock A" or "Stock B":

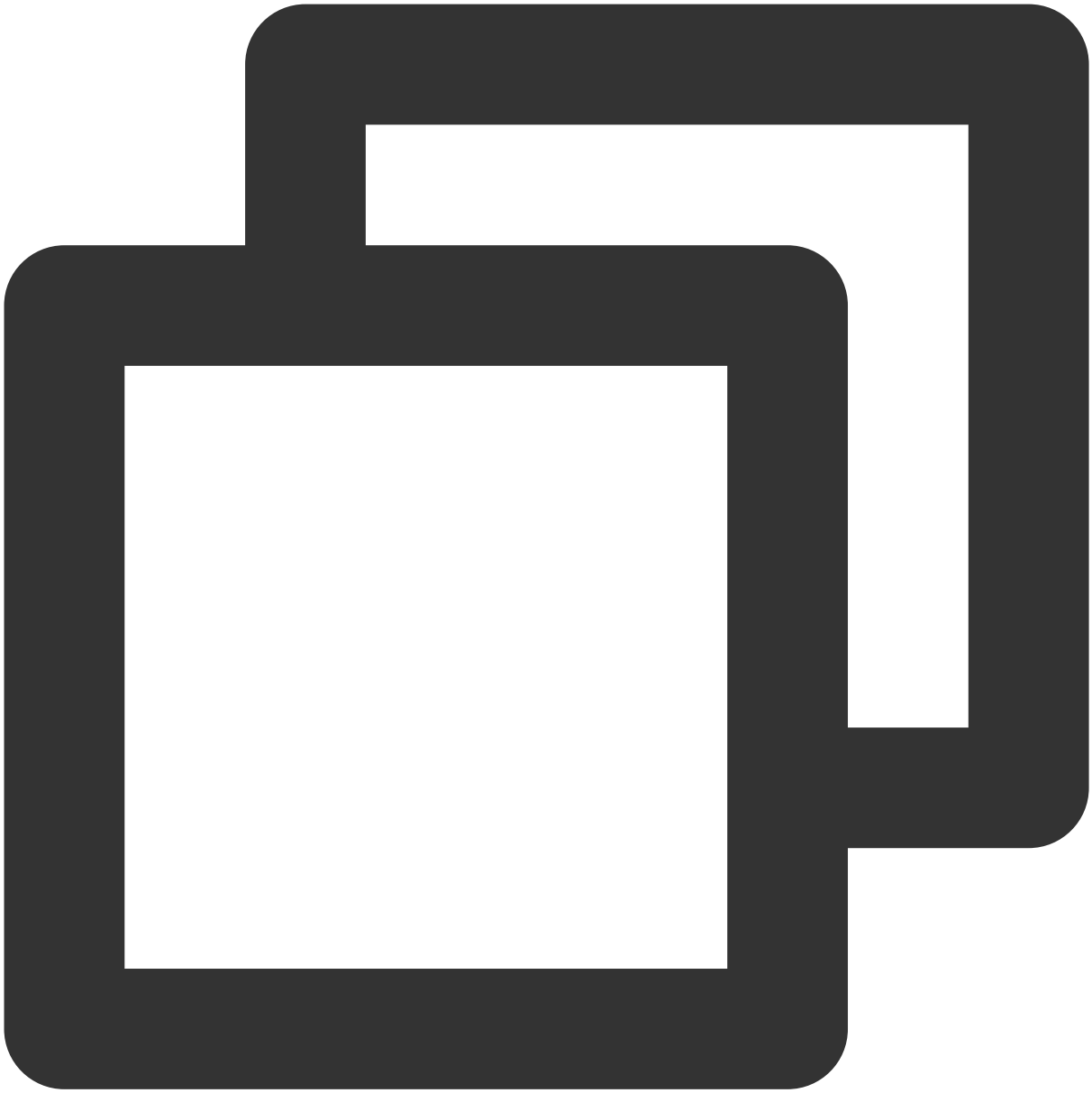


```
{
  "From_Account": "admin",
  "MsgRandom": 124032,
  "OnlineOnlyFlag": 0, // 0: Saves roaming and unread messages. In this case, the p
                       // 1: Does not save roaming or unread messages. In this case
  "Condition": {
      "TagsOr": ["Stock A","Stock B"]
  },
  "MsgBody": [
      {
          "MsgType": "TIMTextElem",
```

```
            "MsgContent": {
                "Text": "hi, beauty"
            }
        }
    ],
    "OfflinePushInfo": {
        "PushFlag": 0,  // 0: Enables offline push; 1: Disables offline push.
        "Title": "offline push title",
        "Desc": "offline push content"
    }
}
```

Pushing Messages by User Attribute

The administrator pushes messages to Platinum Premier membership in Shenzhen:



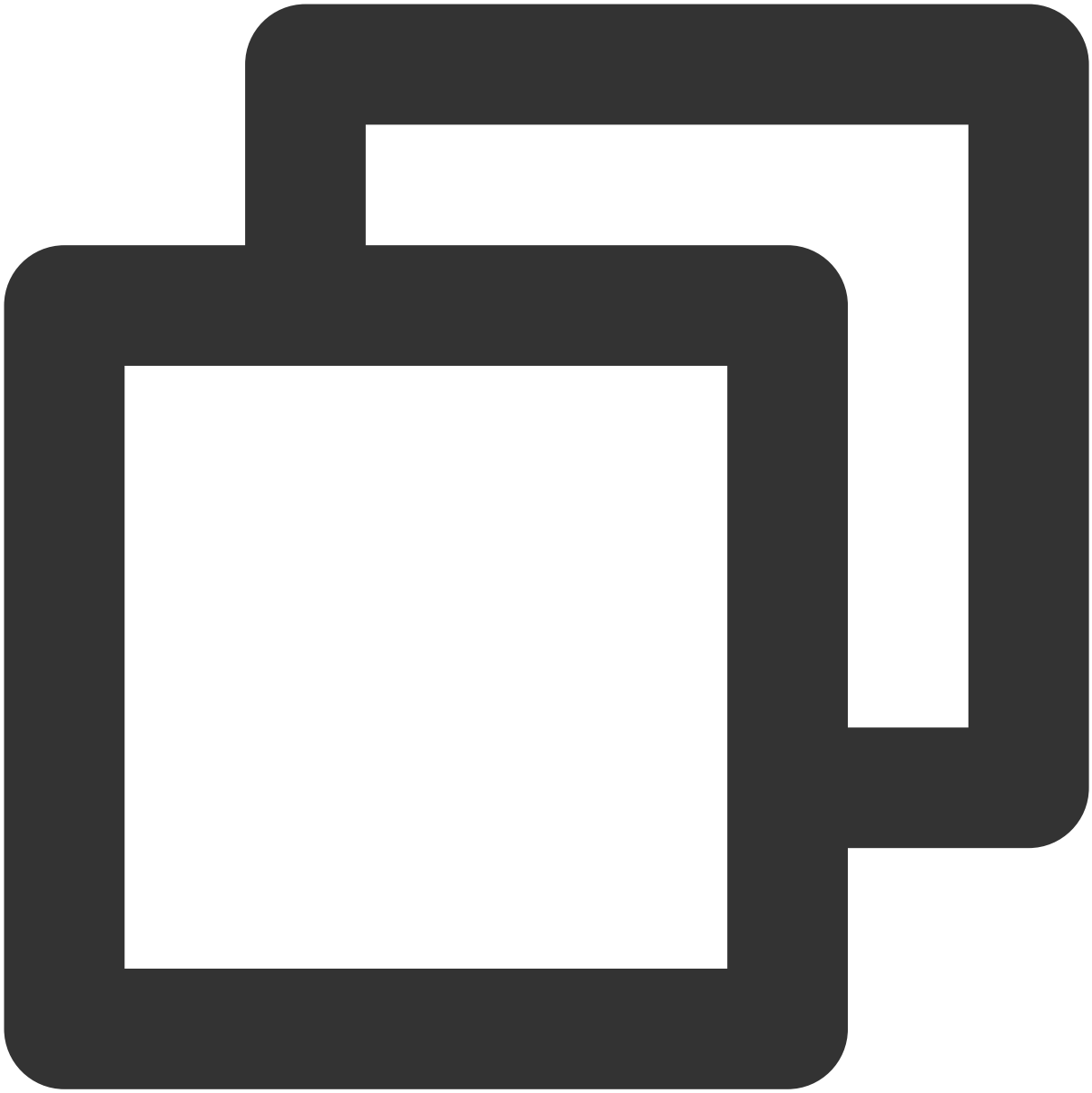
```
{
  "From_Account": "admin",
  "MsgRandom": 389475,
  "OnlineOnlyFlag": 0, // 0: Saves roaming and unread messages. In this case, the p
                       // 1: Does not save roaming or unread messages. In this case
  "Condition": {
      "AttrsAnd": {
          "level": "Platinum Premier Membership",
          "city": "Shenzhen"
      }
  },
```

```
    "MsgBody": [
        {
            "MsgType": "TIMTextElem",
            "MsgContent": {
                "Text": "hi, beauty"
            }
        }
    ],
    "OfflinePushInfo": {
        "PushFlag": 0,  // 0: Enables offline push; 1: Disables offline push.
        "Title": "offline push title",
        "Desc": "offline push content"
    }
}
```

Note:

From_Account is the message sender account and can be specified as any existing account. If no sender is specified or the specified sender does not exist, the API caller account will be taken by default.

The administrator pushes messages to Platinum Premier users in Shenzhen:



```
{
  "From_Account": "admin",
  "MsgRandom": 389475,
  "OnlineOnlyFlag": 0, // 0: Saves roaming and unread messages. In this case, the p
                       // 1: Does not save roaming or unread messages. In this case
  "Condition": {
      "AttrsAnd": {
          "level": "Platinum Premier Users",
          "city": "Shenzhen"
      }
  },
```

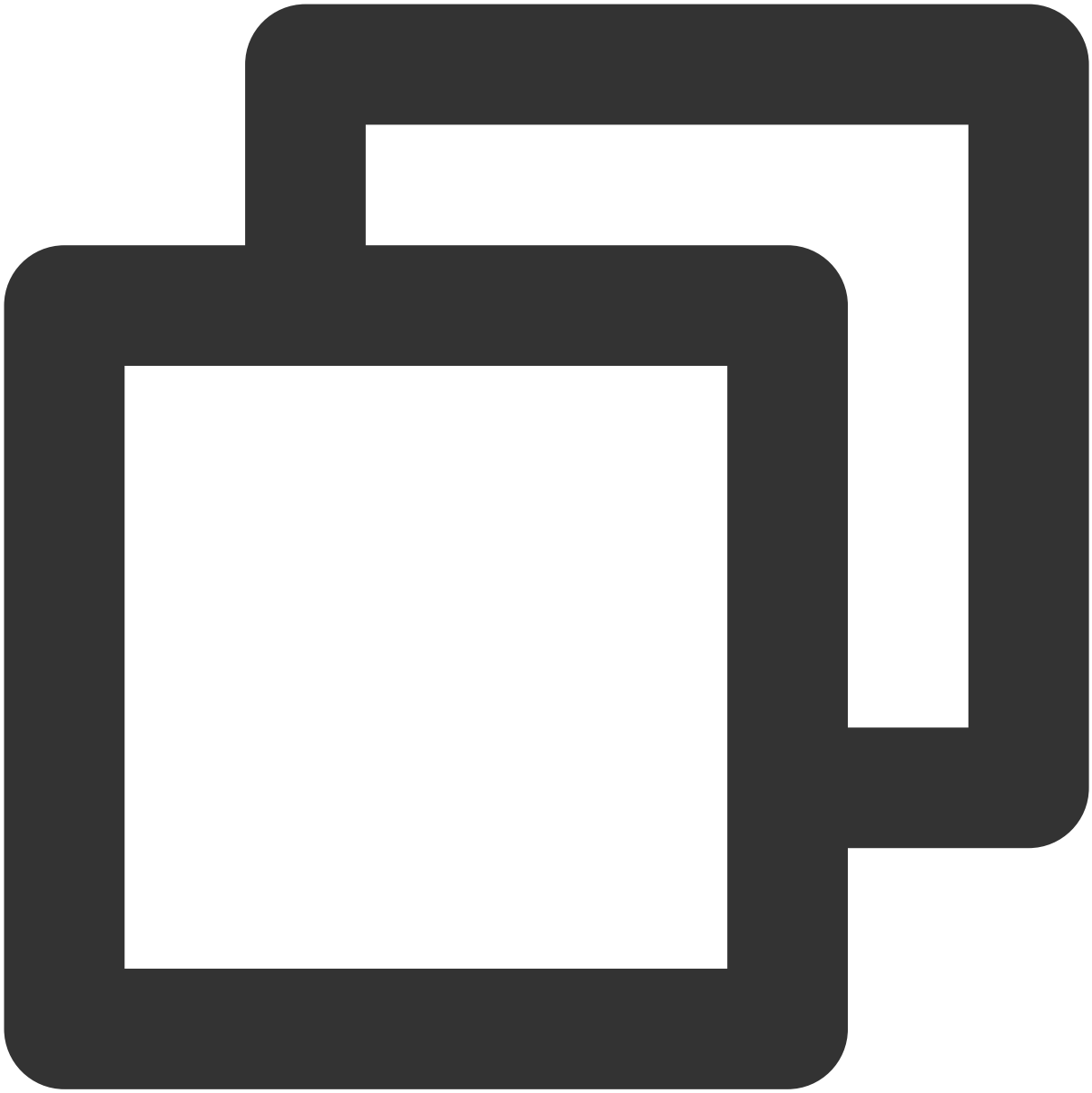
```
    "MsgBody": [
        {
            "MsgType": "TIMTextElem",
            "MsgContent": {
                "Text": "hi, beauty"
            }
        }
    ],
    "OfflinePushInfo": {
        "PushFlag": 0,  // 0: Enables offline push; 1: Disables offline push.
        "Title": "offline push title",
        "Desc": "offline push content"
    }
}
```

Request Fields

Field	Type	Required	Description
From_Account	String	No	Account of the message sender, which can be specified as any existing account. **Note:** If no sender is specified or the specified sender does not exist, the API caller account will be taken by default.
MsgRandom	Integer	Yes	A random number (32-bit unsigned integer) of the message. It is used by the backend for message deduplication within the same second. Make sure a random number is entered.
OnlineOnlyFlag	Integer	No	The default value is 0, which means to save roaming and unread messages. In this case, the push concurrency limit is 200 users/second. 1 means not to save historical messages and not to count unread messages. In this case, there is no push concurrency limit.
Condition	Object	No	Valid values: The or condition of attributes: AttrsOr The and condition of attributes: AttrsAnd The or condition of tags: TagsOr The and condition of tags: TagsAnd **Note:** AttrsOr, AttrsAnd, TagsOr, and TagsAnd cannot coexist. If no condition is specified, messages are pushed to all users.
TagsOr	Array	No	The union of tag conditions. A tag is a string of up to 50 bytes. **Note:** Attributes and tags cannot be used as push conditions at the same time. The TagsOr condition can contain up to 10 tags.

TagsAnd	Array	No	The intersection of tag conditions. A tag is a string of up to 50 bytes. **Note:** Attributes and tags cannot be used as push conditions at the same time. The TagsAnd condition can contain up to 10 tags.
AttrsOr	Object	No	The union of attribute conditions.
AttrsAnd	Object	No	The intersection of attribute conditions.
MsgBody	Array	Yes	Message body. For more information on the message format, see Message Formats. Note that `MsgBody` is an array that can contain multiple message elements.
MsgType	String	Yes	TIM message object type. Valid values: `TIMTextElem` (text message) `TIMLocationElem` (location message) `TIMFaceElem` (emoji message) `TIMCustomElem` (custom message) `TIMSoundElem` (voice message) `TIMImageElem` (image message) `TIMFileElem` (file message) `TIMVideoFileElem` (video message)
MsgContent	Object	Yes	Different message object types (`MsgType`) have different formats (`MsgContent`). For more information, see the **TIMMsgElement** section in Message Formats.
OfflinePushInfo	Object	No	The information to be pushed offline. For more information, see Message Formats.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0,
    "TaskId": "53743040_144115212910570789_4155518400_15723514"
}
```

Response Fields

Field	Type	Description

ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.
TaskId	String	Push task ID.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90002	The `MsgBody` field in the JSON request packets does not meet message format requirements or it is not an array. For more information, see the **TIMMsgElement** section in Message Formats.
90005	The `MsgRandom` field is missing in the JSON request packets or it is not an integer.
90007	The `MsgBody` field in the JSON request packets is not an array. Change it to an array.
90009	The request requires app admin permissions.
90010	The JSON request packets do not meet message format requirements. For more information, see the **TIMMsgElement** section in Message Formats.
90020	The tag length exceeds the limit (the maximum length allowed is 50 bytes).
90022	`TagsOr` and `TagsAnd` in the push conditions contain repeated tags.
90024	Pushes are too frequent. The interval between two pushes must be greater than 1 second.
90026	Incorrect offline message storage period. The value cannot exceed 7 days.
90032	The number of tags in the push conditions exceeds 10, or the number of tags in the tag adding request exceeds 10.
90033	Invalid attribute.

90039	Message push by attribute and message push by tag are mutually exclusive.
90040	A tag in the push conditions is null.
90045	The feature of pushing to all/tagged users is not enabled.
90047	The number of pushes exceeds the daily quota (default quota: 100).
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags
Adding User Tags
Deleting User Tags
Deleting All User Tags
Recalling Push

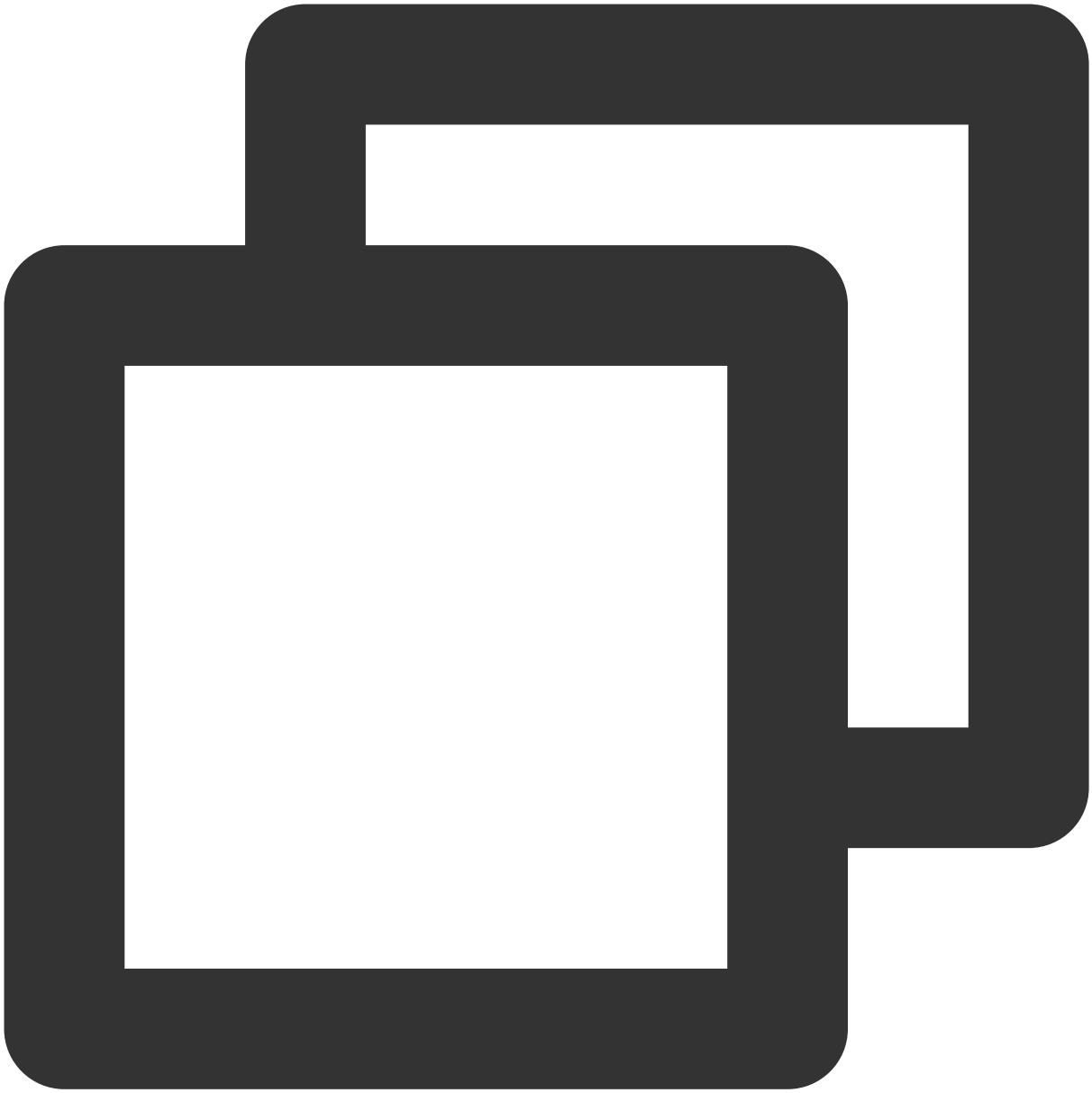
Obtaining Application Attribute Names

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to obtain application attribute names. To call this API, you need to set application attribute names first.

Sample Request URL



```
https://xxxxxx/v4/timpush/get_attr_name?usersig=xxx&identifier=admin&sdkappid=88888
```

Request Parameters

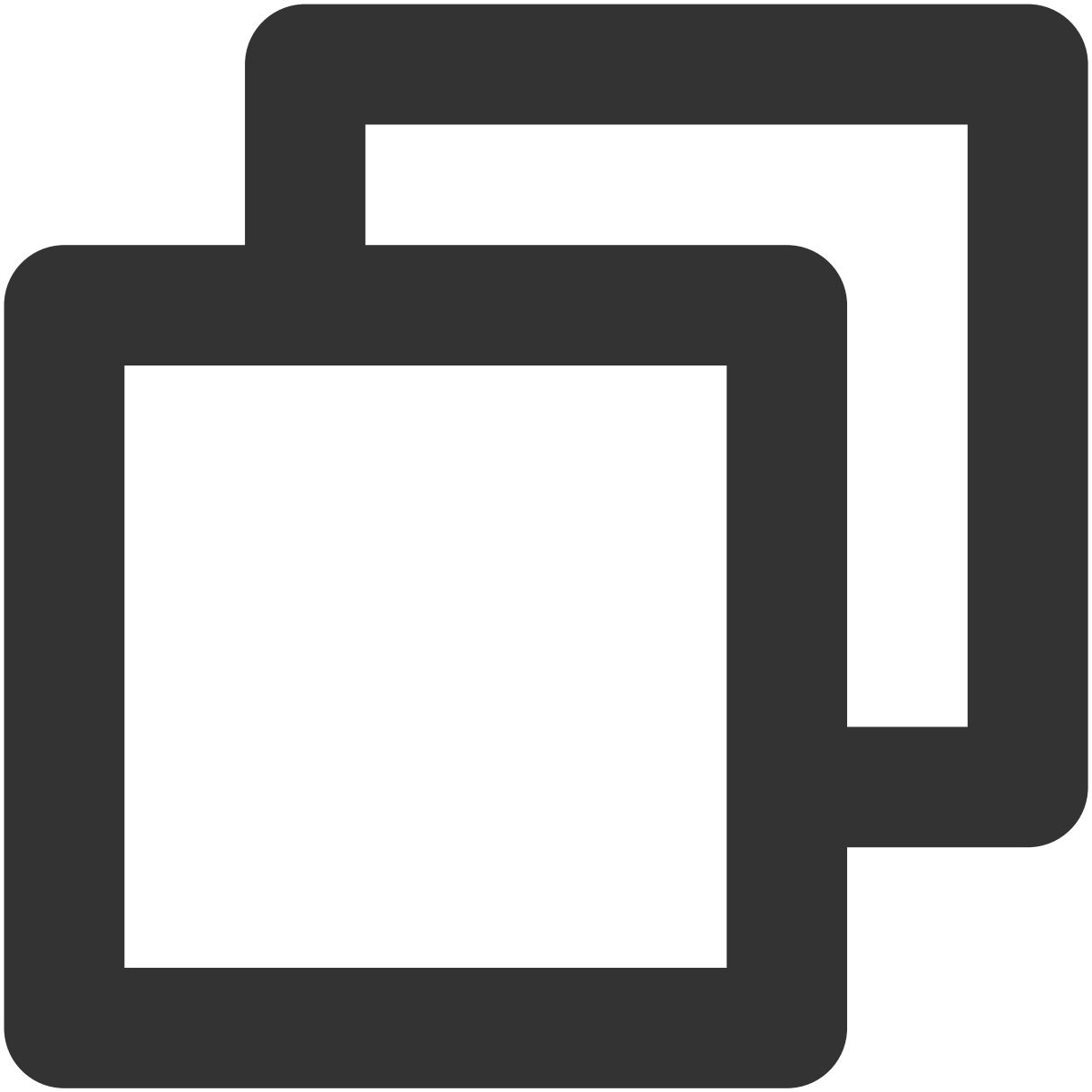
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID.

	China: `console.tim.qq.com` Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/get_attr_name	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets

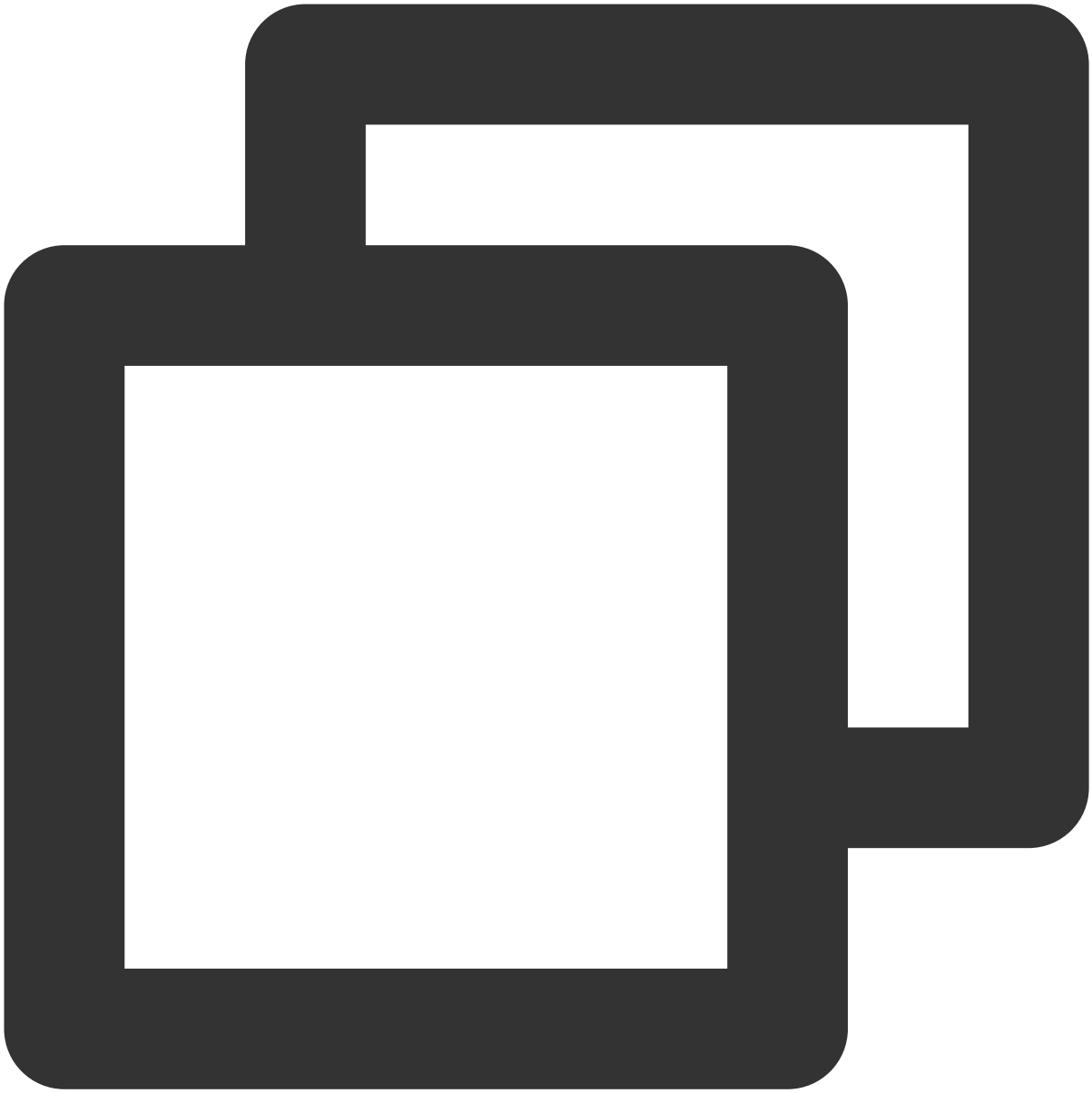


```
{}
```

Request Fields

None.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0,
    "AttrNames": {
        "0": "sex",
        "1": "city",
        "2": "level"
    }
}
```

Response Fields

Field	Type	Description
ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.
AttrNames	Object	A series of key-value pairs. Each key-value pair indicates the name of the corresponding attribute. For example, "0":"xxx" indicates that the name of attribute 0 is xxx.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users

Setting Application Attribute Names

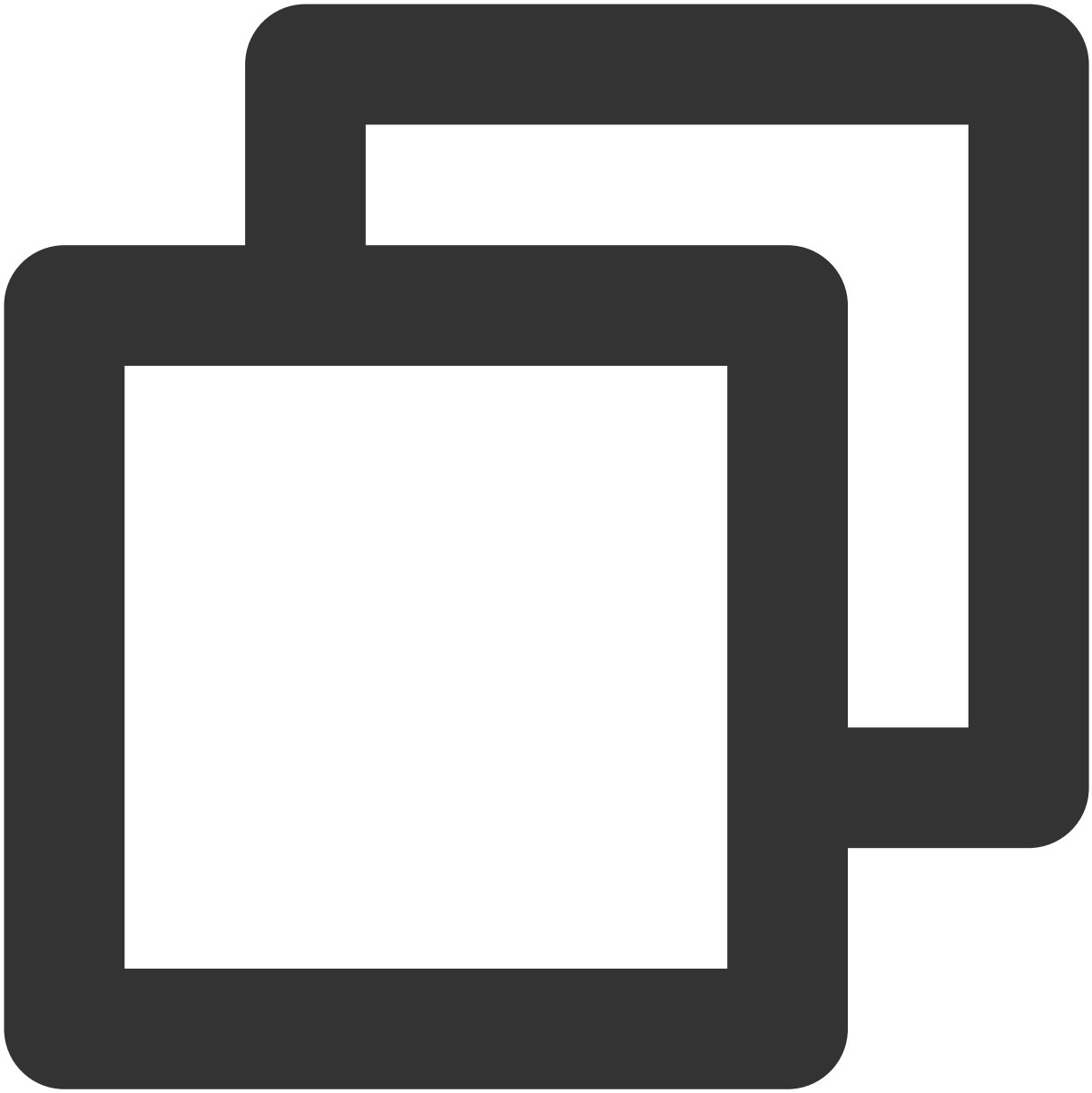
Setting Application Attribute Names

Last updated：2024-07-16 11:40:03

Feature Overview

You can set a maximum of 10 custom user attributes for each application. This API is used to set the name of each attribute. After you set attribute names, they can be used for push by user attribute.

Sample Request URL



```
https://xxxxxx/v4/timpush/set_attr_name?usersig=xxx&identifier=admin&sdkappid=88888
```

Request Parameters

Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID.

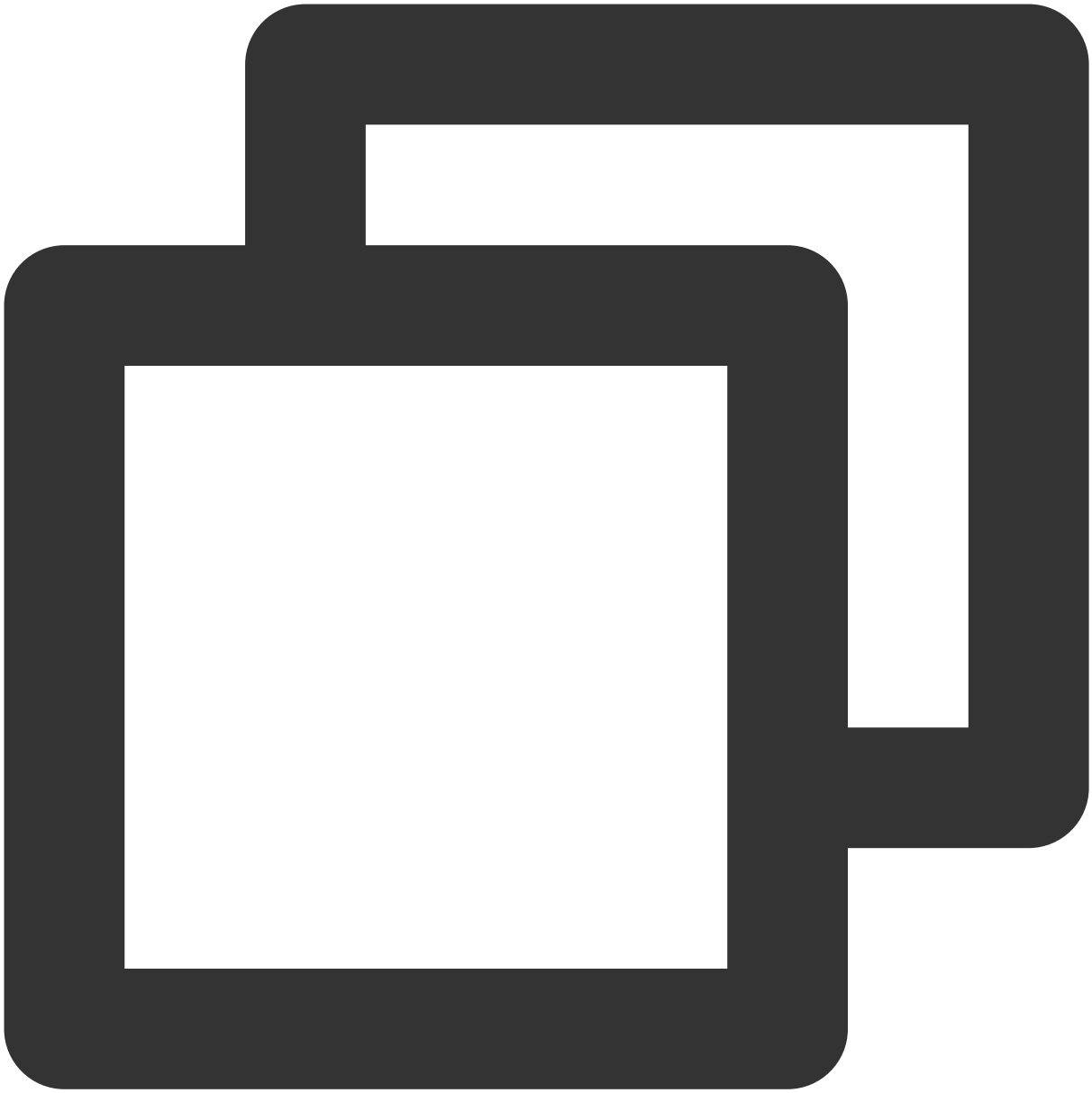
	China: `console.tim.qq.com` Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/set_attr_name	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets

Set attribute `0` of the application to `sex` , attribute `1` to `city` , and attribute `2` to `country` .

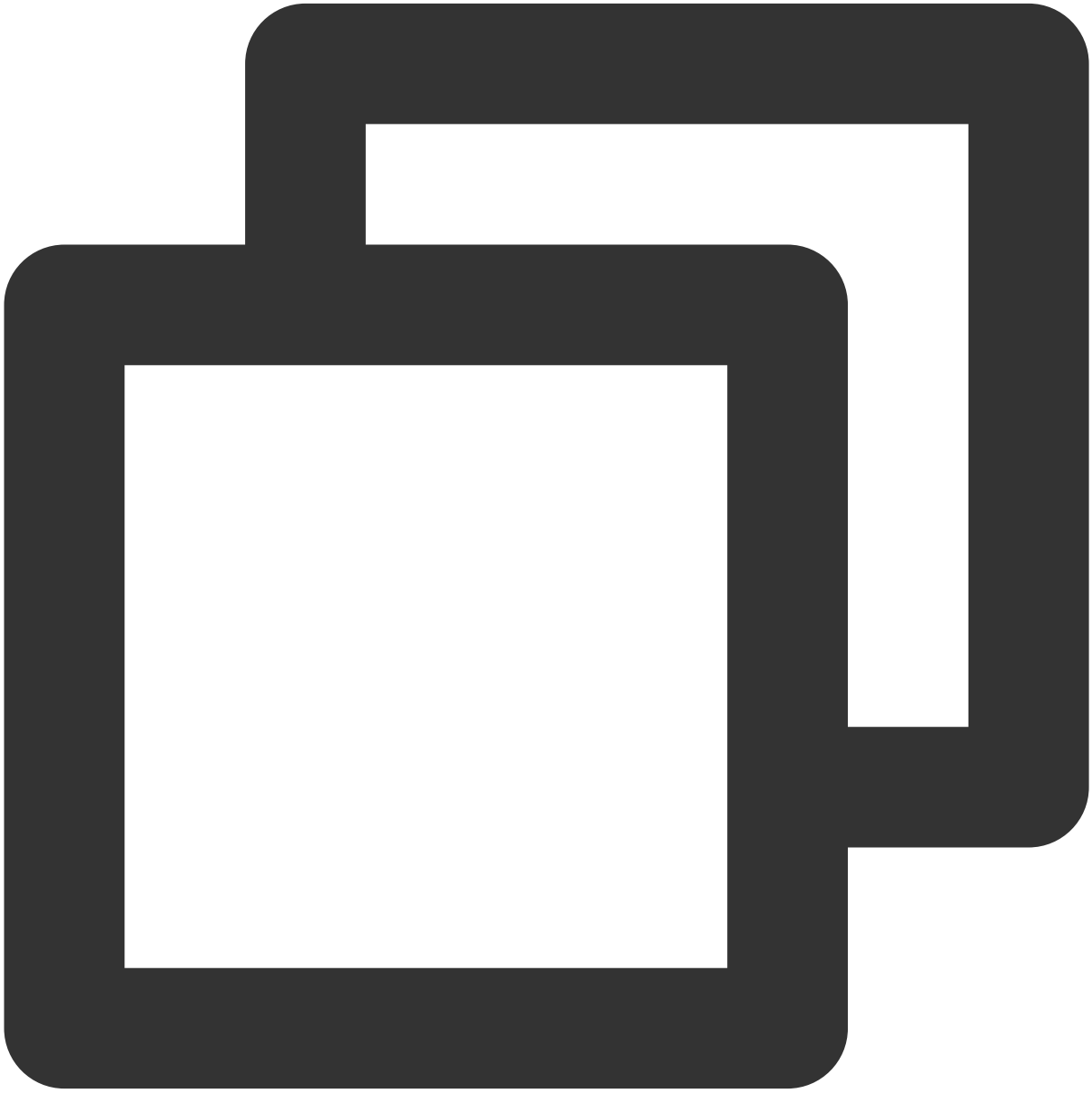


```
{
    "AttrNames": {
        "0": "sex",
        "1": "city",
        "2": "country"
    }
}
```

Request Fields

Field	Type	Required	Description
Digital key	String	Yes	Attribute number (0 to 9).
Attribute name	String	Yes	The attribute name cannot exceed the length limit of 50 bytes. An application can have a maximum of 10 push attributes (numbered from 0 to 9), and users can customize the meaning of each attribute.

Sample Response Packets



```
{
```

```
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description
ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users

Setting Application Attribute Names

Obtaining Application Attribute Names

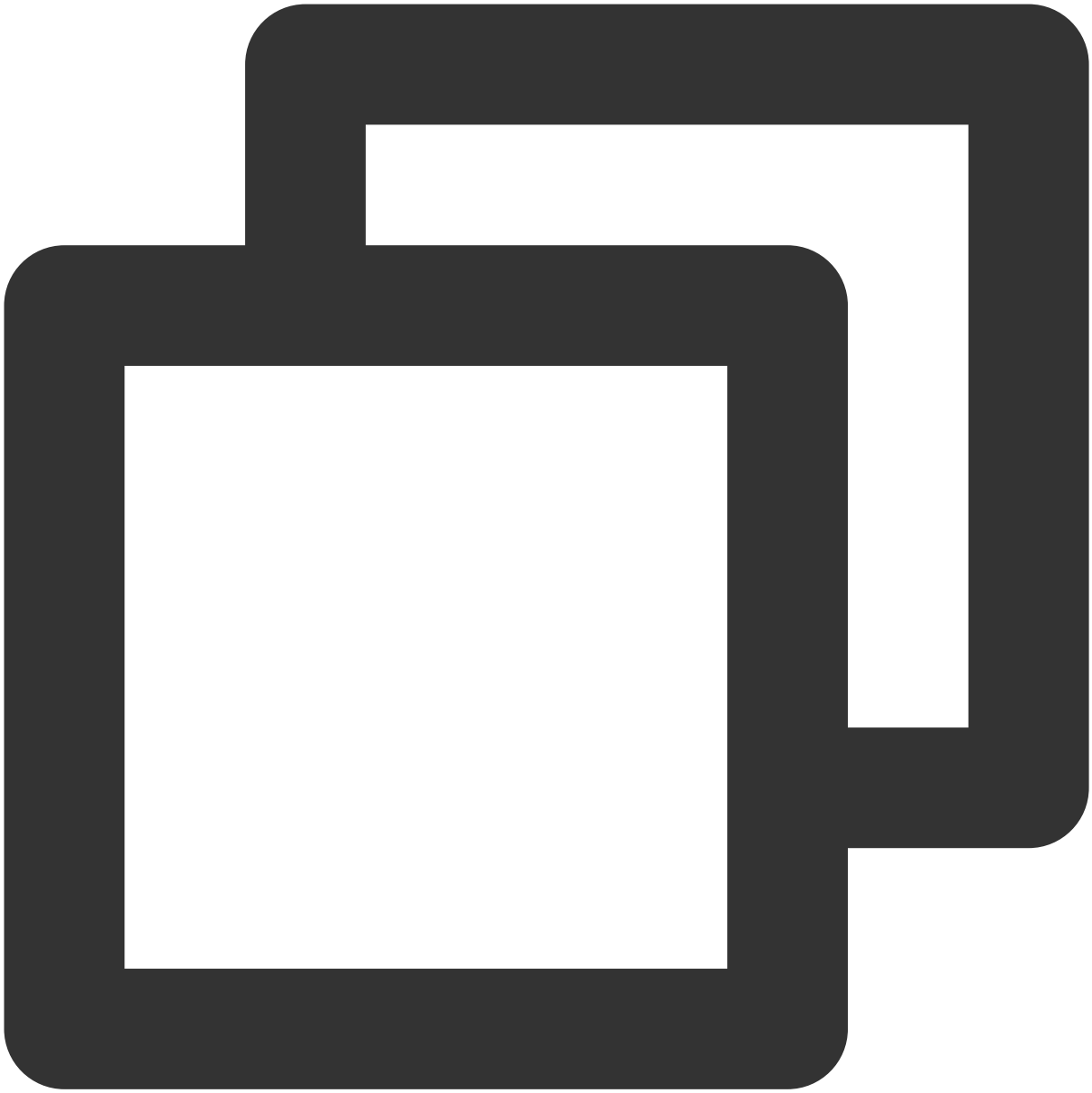
Obtaining User Attributes

Last updated：2024-07-23 18:06:24

Feature Overview

This API is used by the administrator to obtain user attributes. Up to 100 users' attributes can be obtained at a time. To call this API, you need to set application attribute names first.

Sample Request URL



```
https://xxxxxx/v4/timpush/get_attr?usersig=xxx&identifier=admin&sdkappid=88888888&r
```

Request Parameters

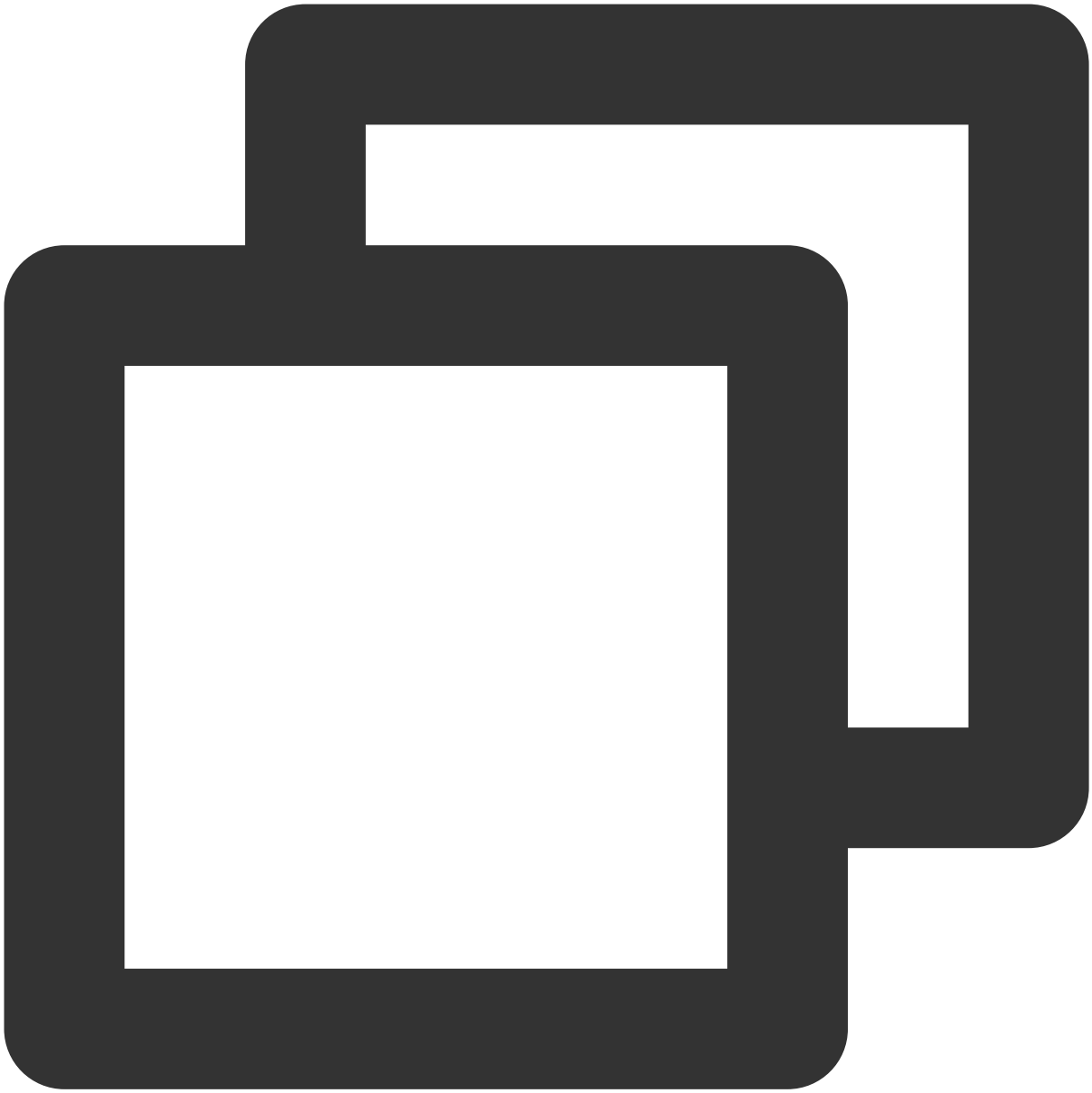
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/get_attr	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets

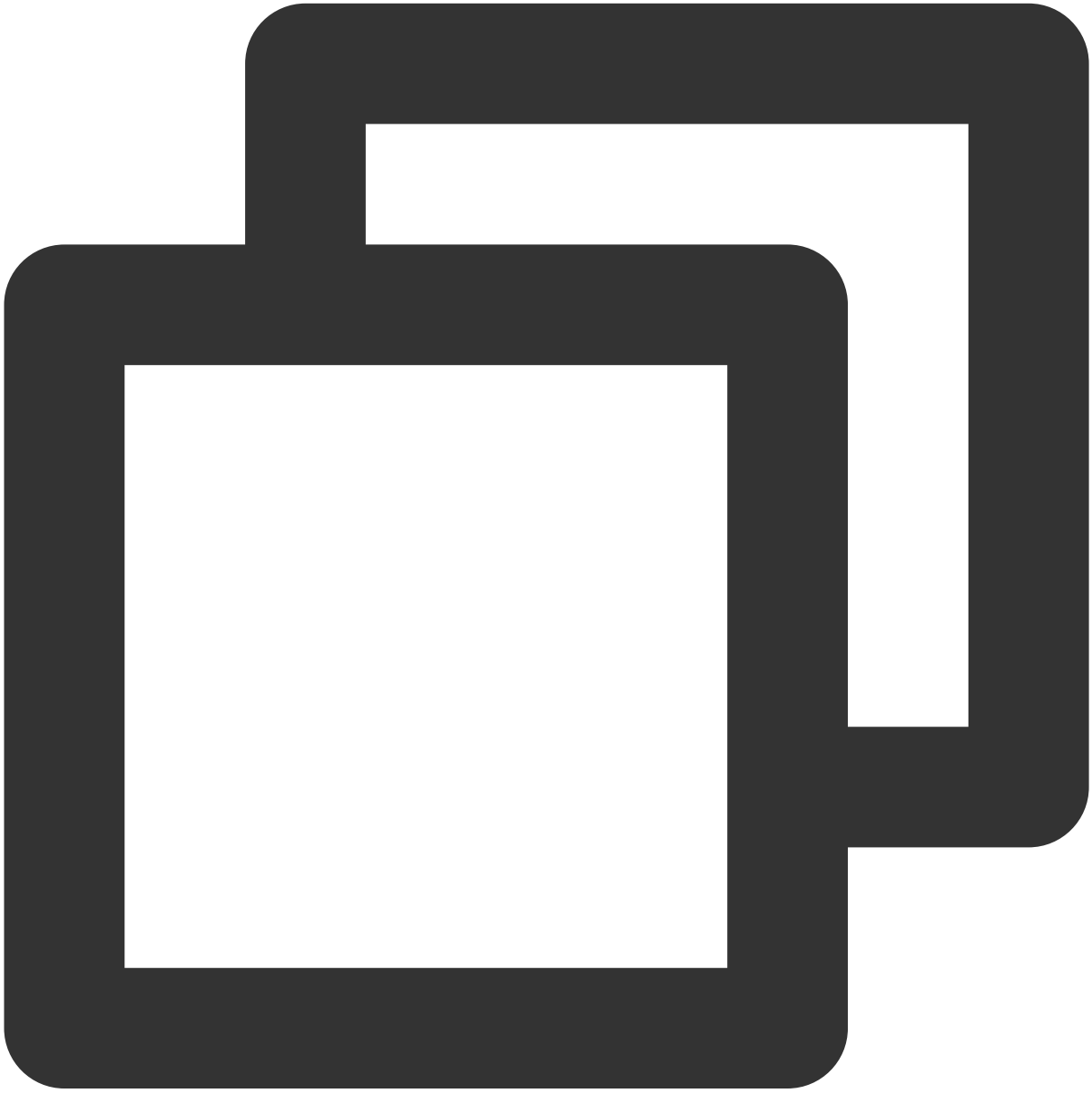


```
{
    "To_Account": [
        "Jack",
        "Daved",
        "abc"
    ]
}
```

Request Fields

Field	Type	Required	Description
To_Account	Array	Yes	List of target user accounts.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0,
```

```
    "UserAttrs": [
        {
            "To_Account": "Jack",
            "Attrs": {
                "sex" : "Female",
                "city": "New York"
            }
        },
        {
            "To_Account": "abc",
            "Attrs": {}
        },
        {
            "To_Account": "Daved",
            "Attrs": {
                "sex": "M",
                "city": "Shenzhen"
            }
        }
    ]
}
```

Response Fields

Field	Type	Description
ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.
UserAttrs	Array	List of user attributes.
To_Account	String	User account.
Attrs	Object	Attribute content.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90018	The number of requested accounts exceeds the limit.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags
Adding User Tags
Deleting User Tags
Deleting All User Tags
Recalling Push

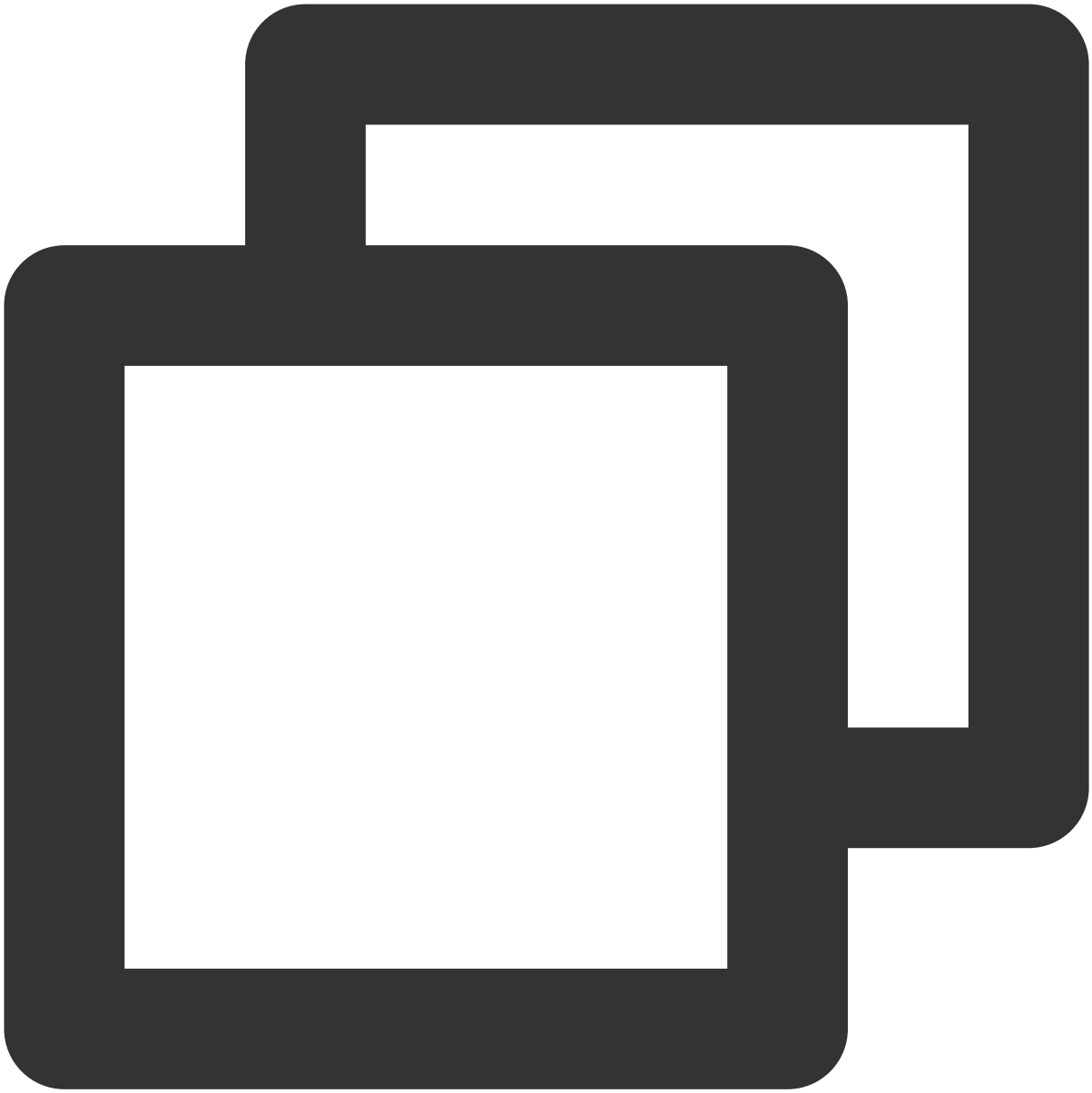
Setting User Attributes

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to set attributes for users. Up to 100 users' attributes can be set at a time. To call this API, you need to set application attribute names first.

Sample Request URL



```
https://xxxxxx/v4/timpush/set_attr?usersig=xxx&identifier=admin&sdkappid=88888888&r
```

Request Parameters

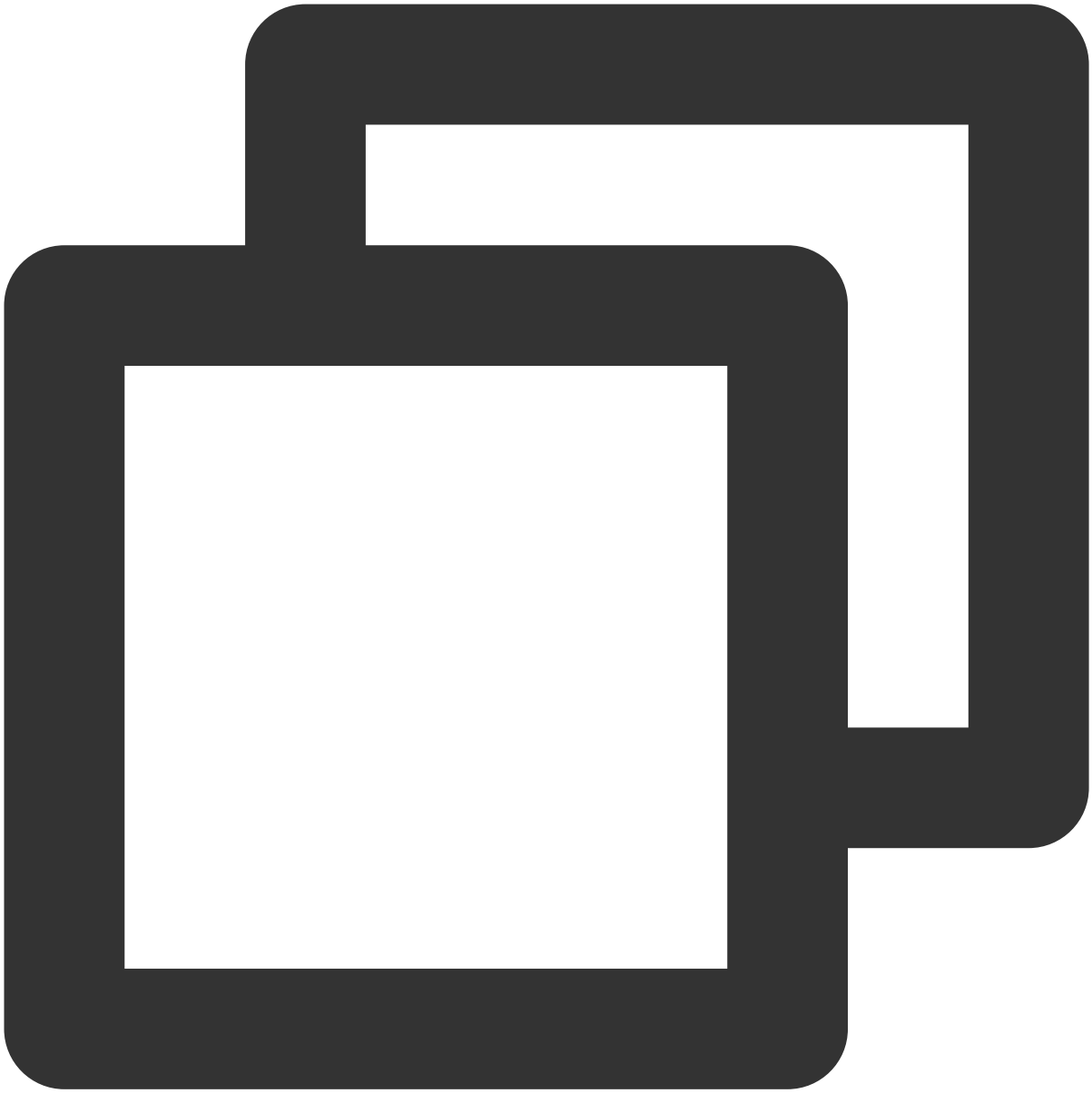
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/set_attr	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets



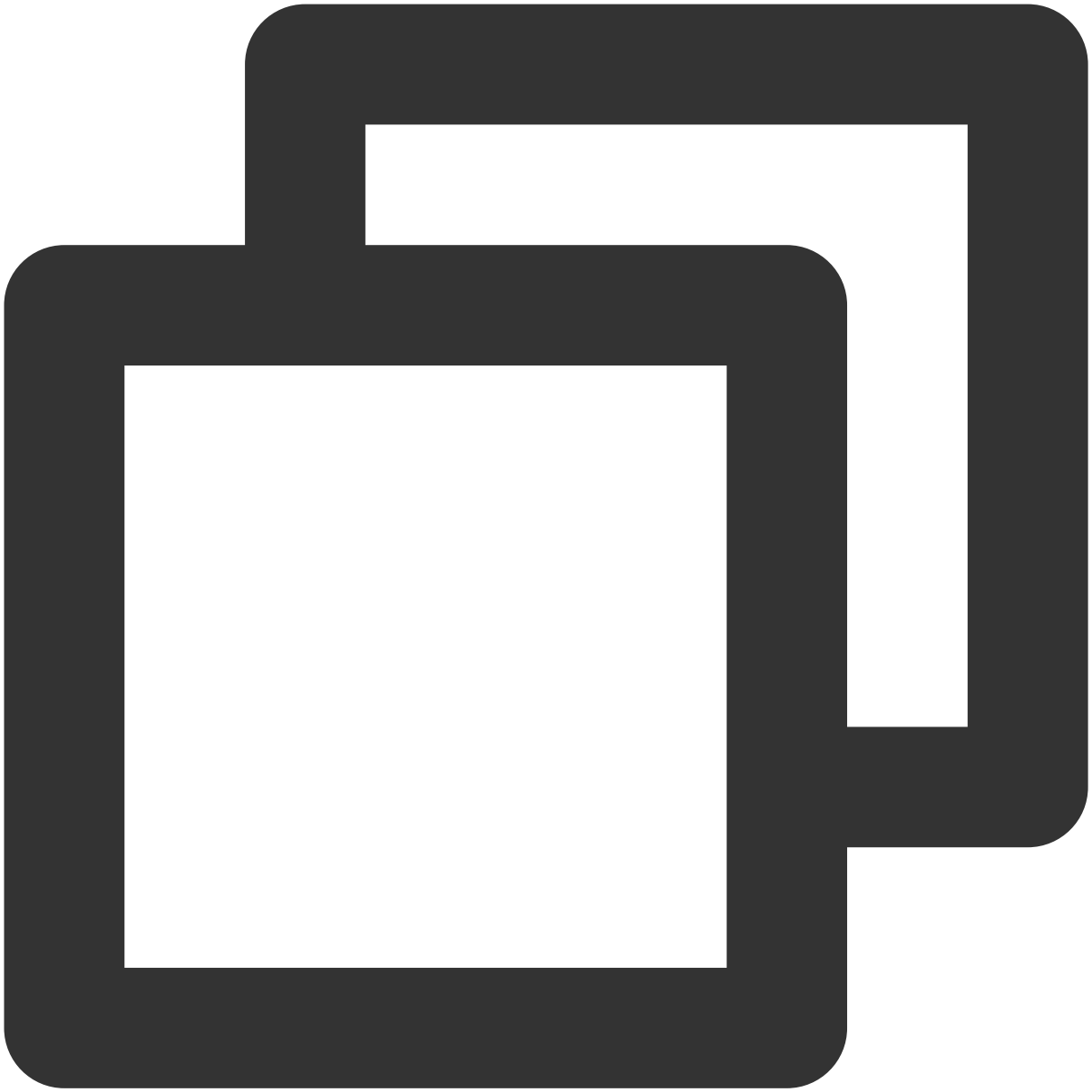
```
{
    "UserAttrs":
    [
        {
            "To_Account": "xiaojun012",
            "Attrs": {
                "sex": "attr1",
                "city": "attr2"
            }
        },
        {
```

```
            "To_Account": "xiaojun013",
            "Attrs": {
                "city": "attr3",
                "sex": "attr4"
            }
        }
    ]
}
```

Request Fields

Field	Type	Required	Description
To_Account	String	Yes	Target user account.
Attrs	Object	Yes	Attribute set. Each attribute is a key-value pair, with the key being the attribute name and the value being the user's corresponding attribute value. A user attribute value cannot exceed 50 bytes.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description

ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
90033	Invalid attribute.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags
Adding User Tags

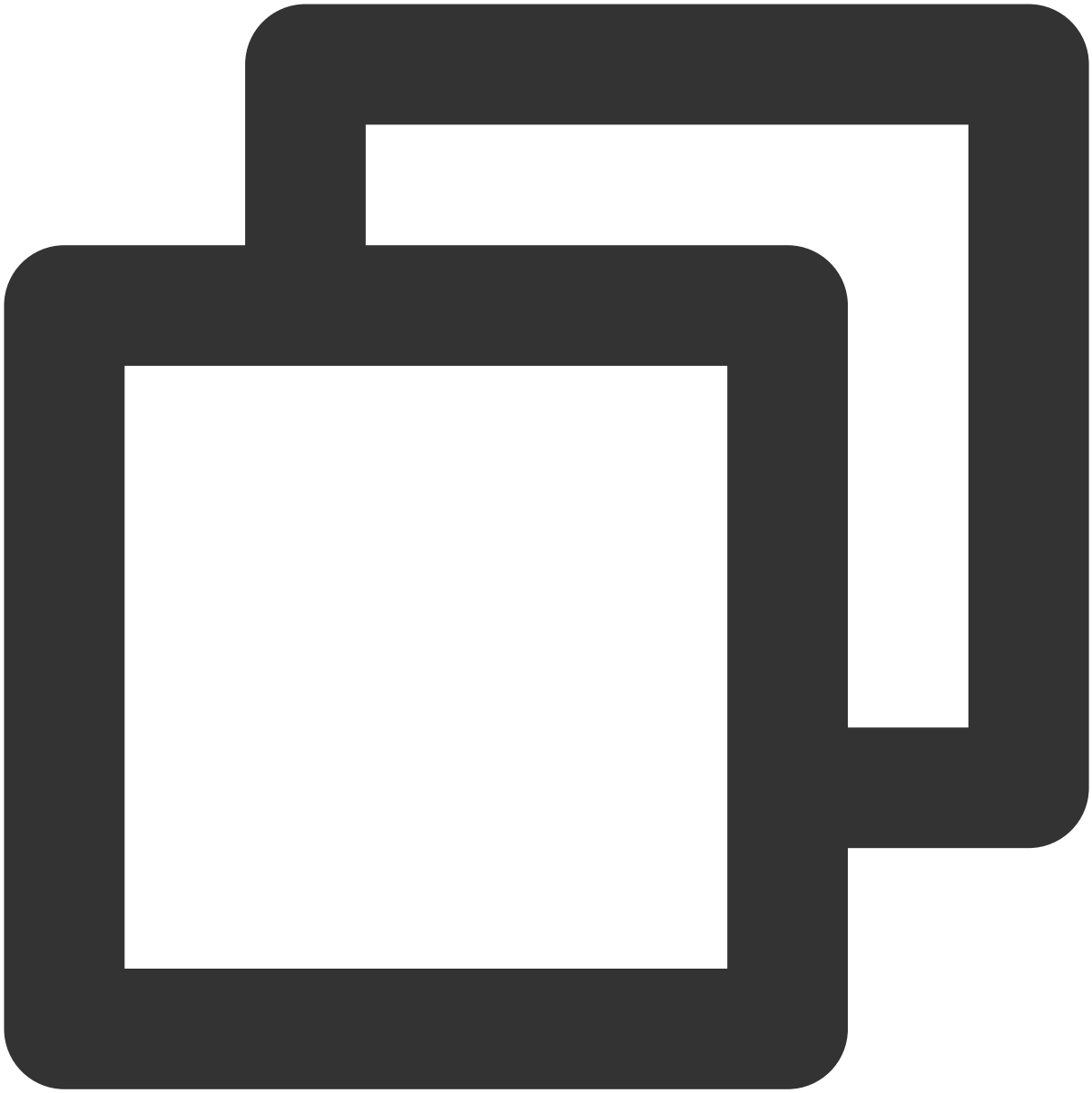
Deleting User Attributes

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to delete attributes for users. Up to 100 users' attributes can be deleted at a time. To call this API, you need to set application attribute names first.

Sample Request URL



```
https://xxxxxx/v4/timpush/remove_attr?usersig=xxx&identifier=admin&sdkappid=8888888
```

Request Parameters

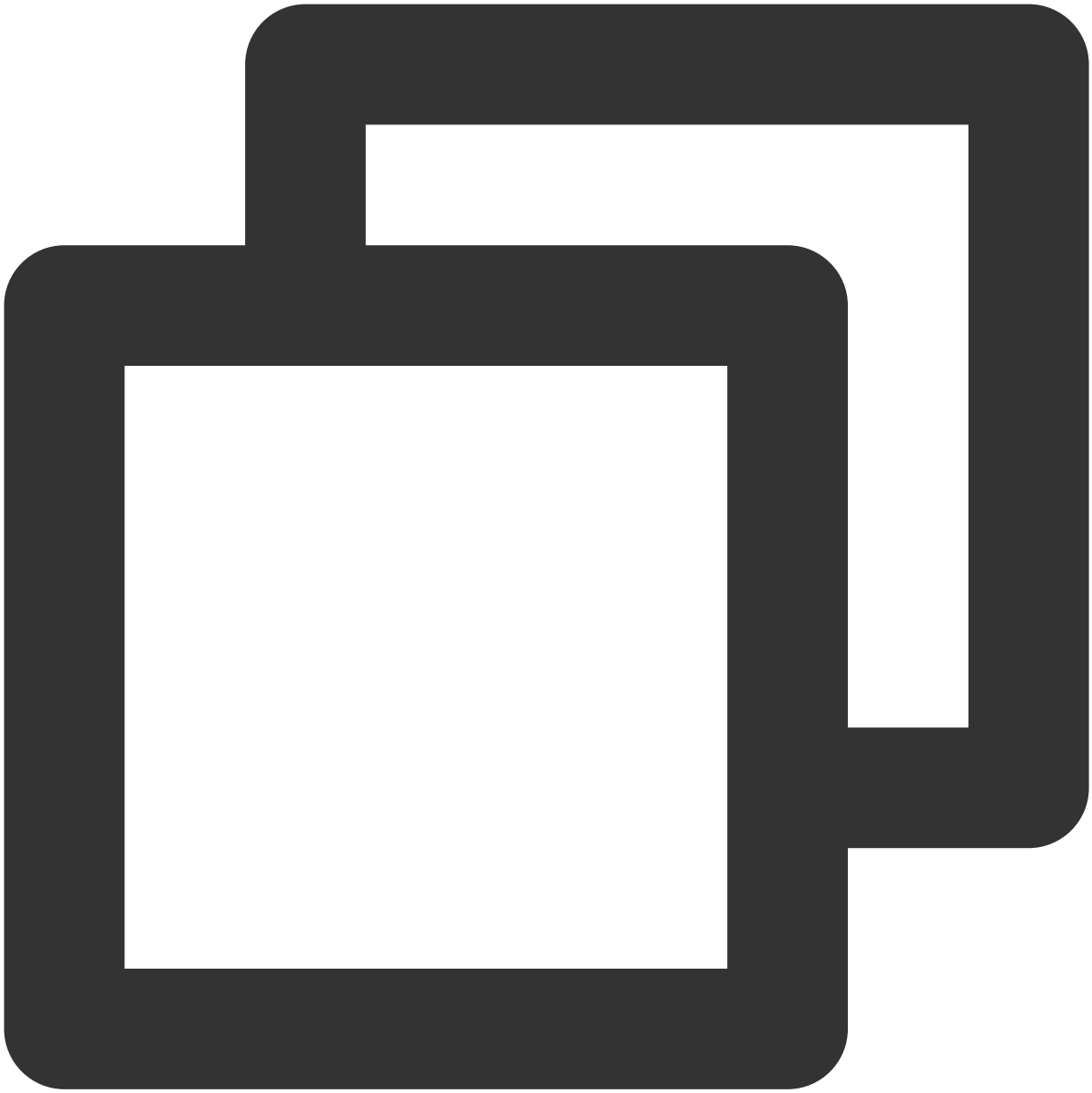
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/remove_attr	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets



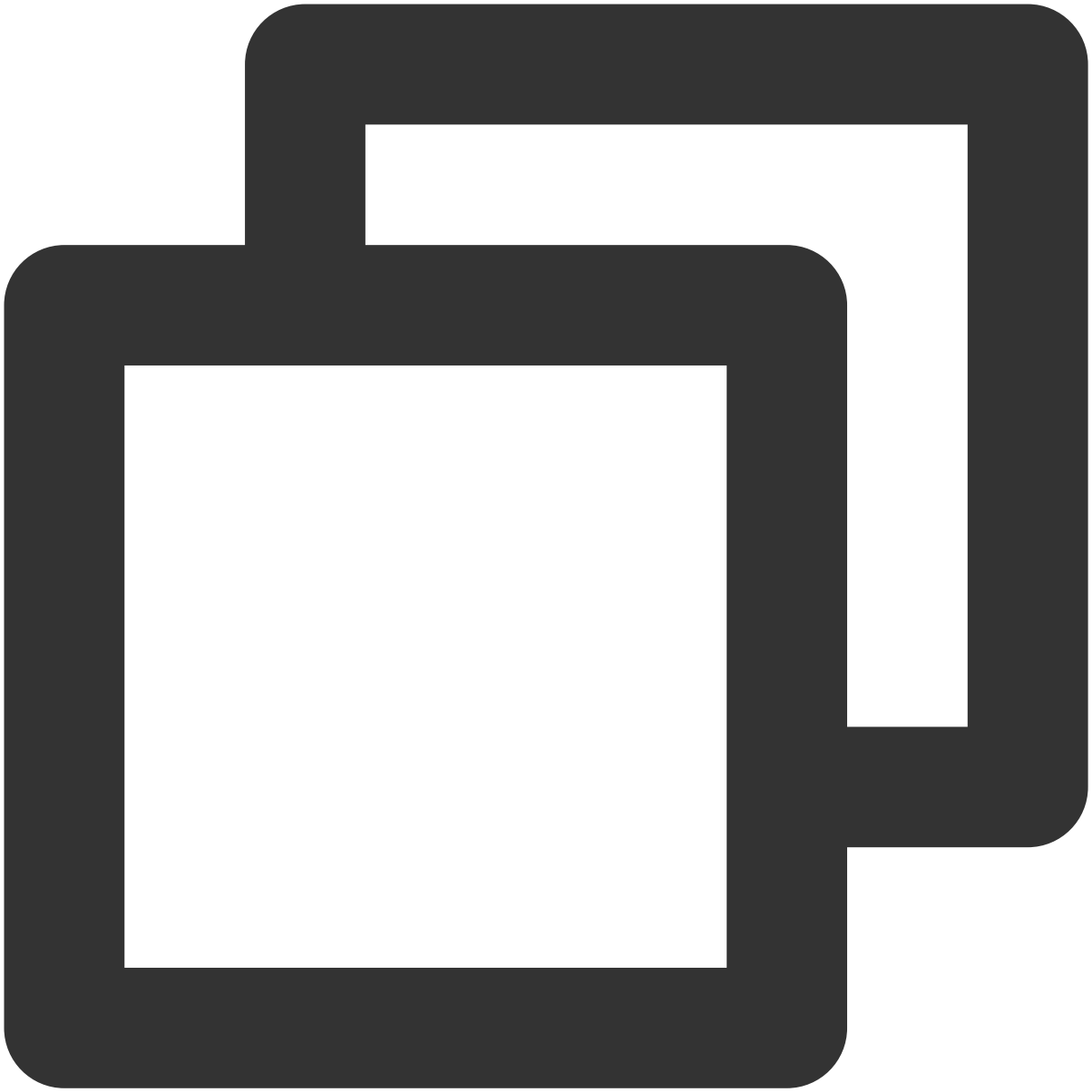
```
{
    "UserAttrs": [
        {
            "To_Account": "xiaojun013",
            "Attrs": [
                "sex",
                "city"
            ]
        },
        {
            "To_Account": "xiaojun012",
```

```
            "Attrs": [
                "sex",
                "city"
            ]
        }
    ]
}
```

Request Fields

Field	Type	Required	Description
To_Account	String	Yes	Target user account.
Attrs	Array	Yes	Attribute set. Note that you only need to specify the attribute names here. For more information on the format and meanings of `Attrs` , see Setting Application Attribute Names.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description

ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
90033	Invalid attribute.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags

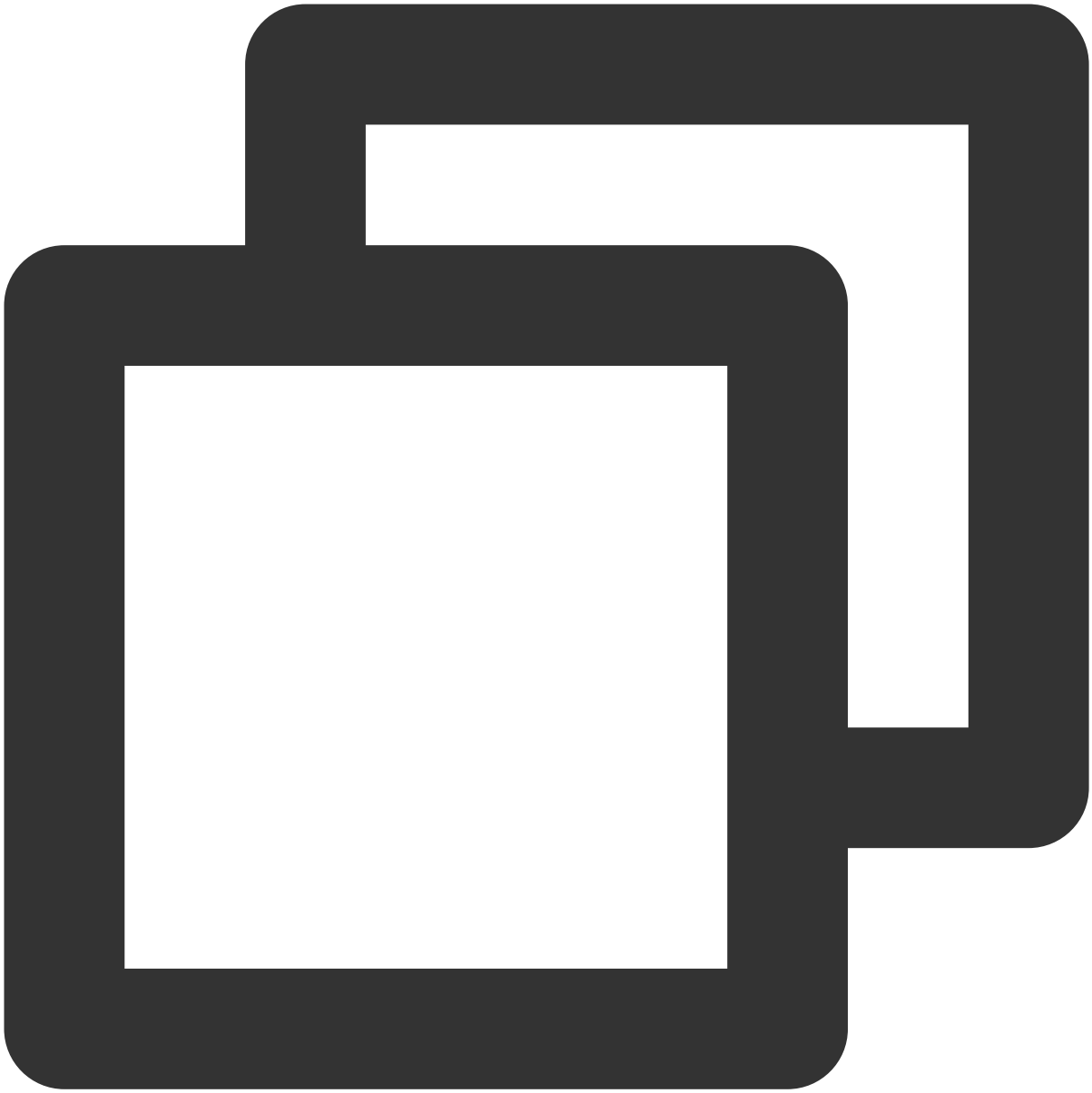
Obtaining User Tags

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to obtain user tags. Up to 100 users' tags can be obtained at a time.

Sample Request URL



```
https://xxxxxx/v4/timpush/get_tag?usersig=xxx&identifier=admin&sdkappid=88888888&ra
```

Request Parameters

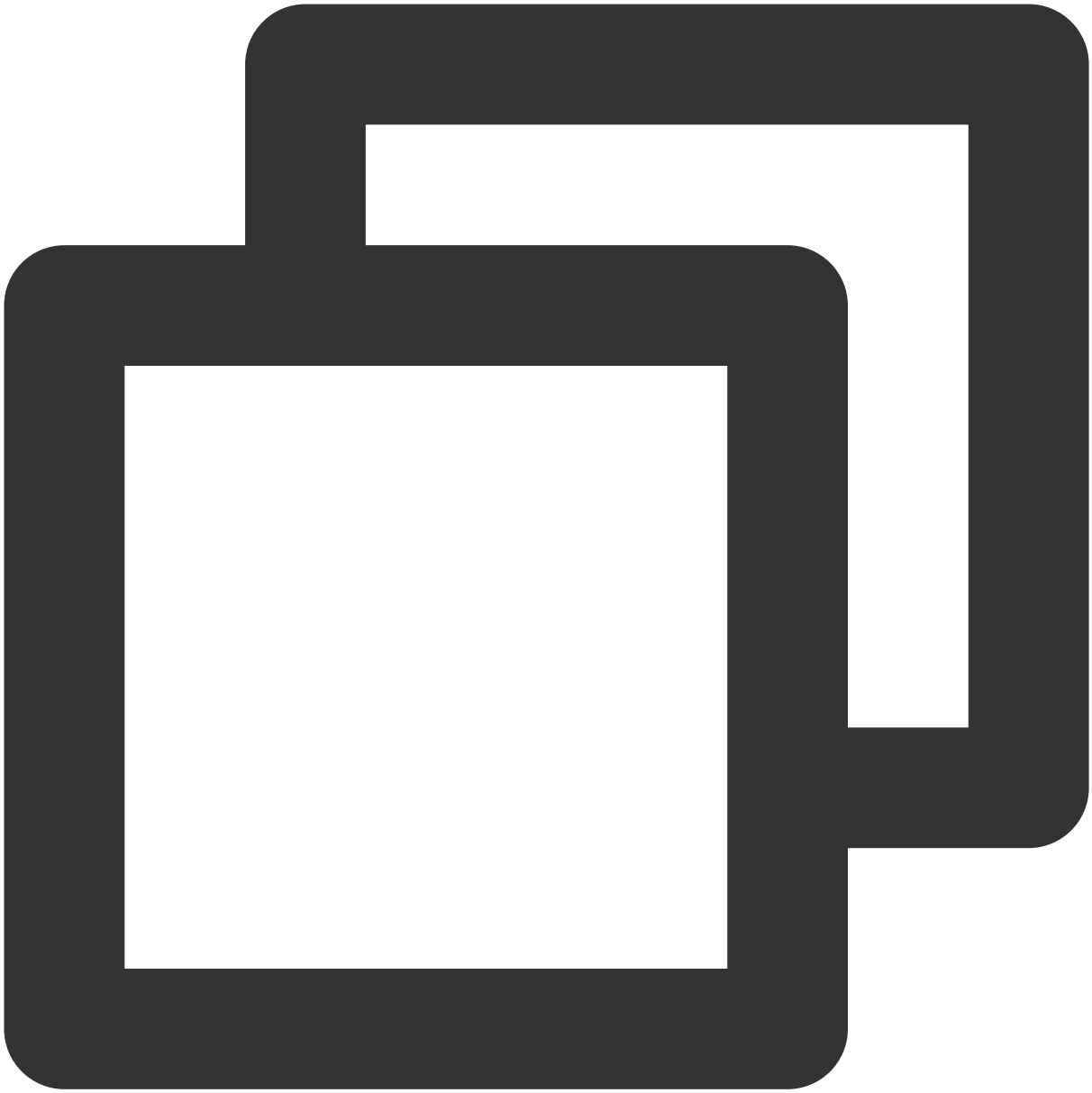
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com`
v4/timpush/get_tag	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets



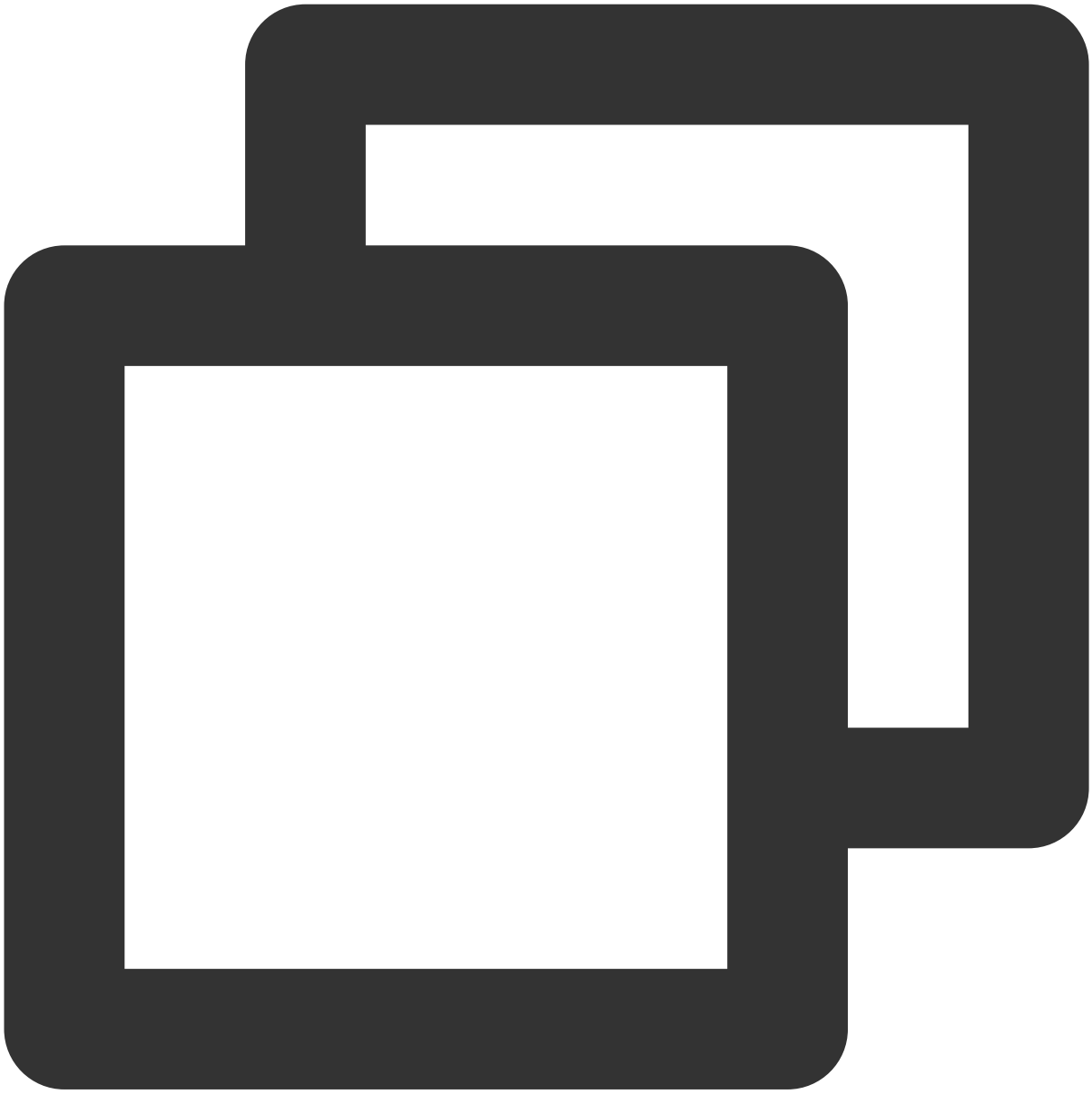
```
{
    "To_Account": [
        "xiaojun012",
        "xiaojun013"
    ]
}
```

Request Fields

Field	Type	Required	Description

To_Account	Array	Yes	List of target user accounts.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0,
    "UserTags": [
        {
```

```
            "To_Account": "xiaojun012",
            "Tags": ["a", "b"]
        },
        {
            "To_Account": "xiaojun013",
            "Tags": ["a", "c"]
        }
    ]
}
```

Response Fields

Field	Type	Description
ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.
UserTags	Array	List of user tags.
To_Account	String	User account.
Tags	Array	Tag content.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags
Adding User Tags
Deleting User Tags
Deleting All User Tags
Recalling Push

Adding User Tags

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to add tags for users.

Note:

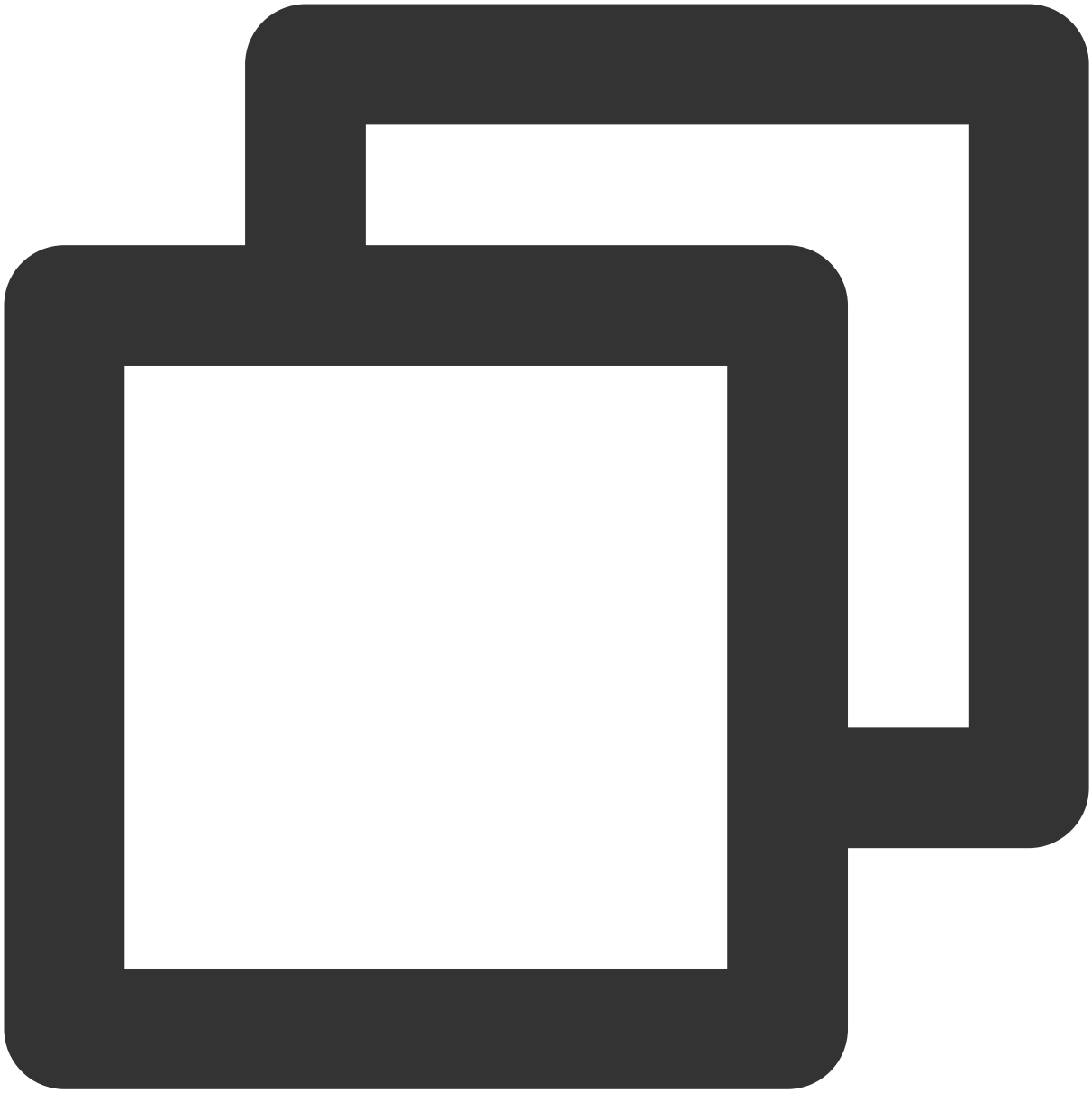
Up to 100 users' tags can be added at a time. In the request body, a maximum of 10 tags can be added for each user.

A maximum of 100 tags can be set for each user. If a user has more than 100 tags, you need to delete old tags before adding new ones for the user.

The maximum number of tags that can be set for an application is 1,000, that is, the deduplicated total number of tags for all users is up to 1,000.

The maximum length of a single tag is 50 bytes.

Sample Request URL



```
https://xxxxxx/v4/timpush/add_tag?usersig=xxx&identifier=admin&sdkappid=88888888&ra
```

Request Parameters

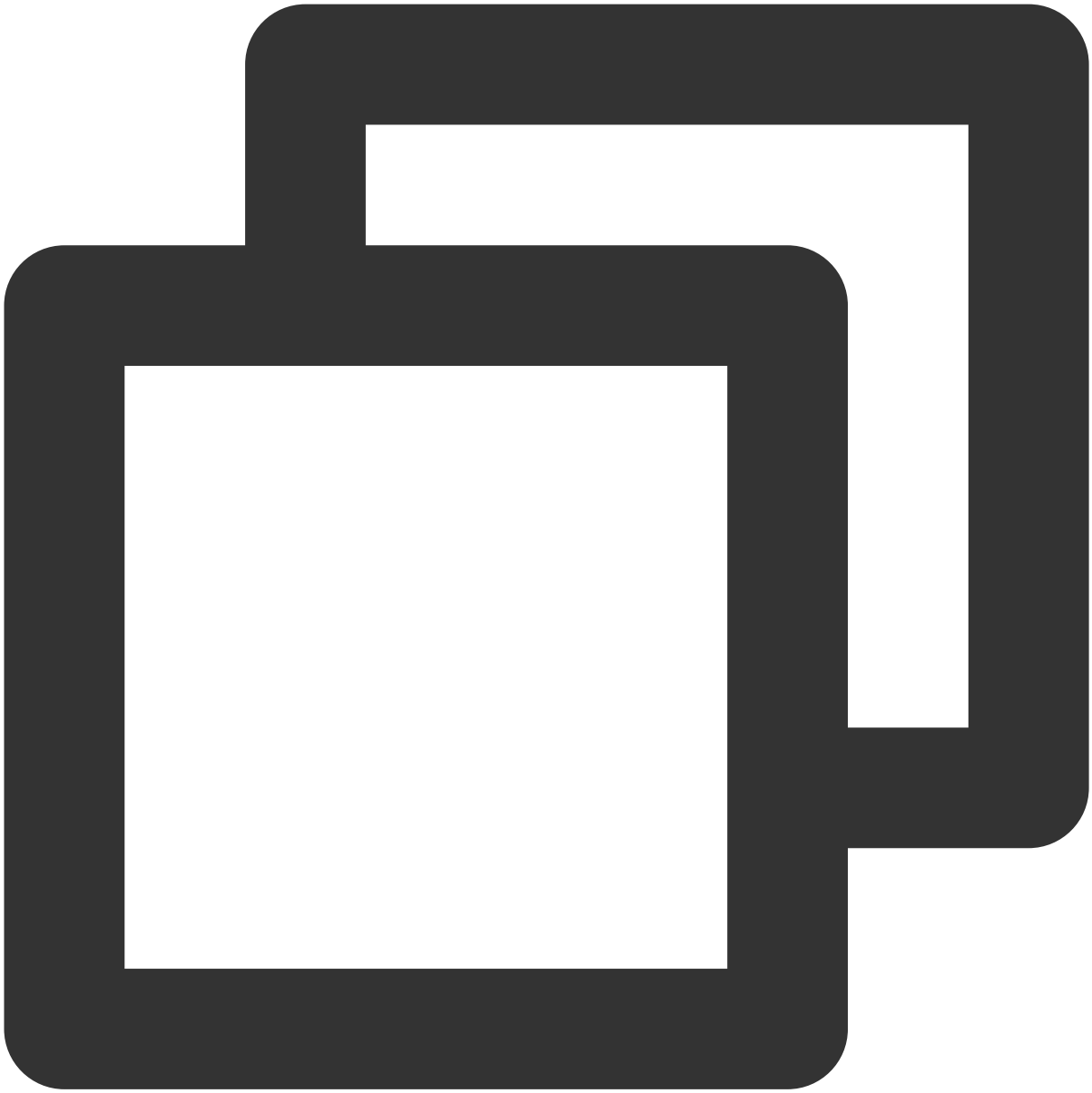
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/add_tag	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets



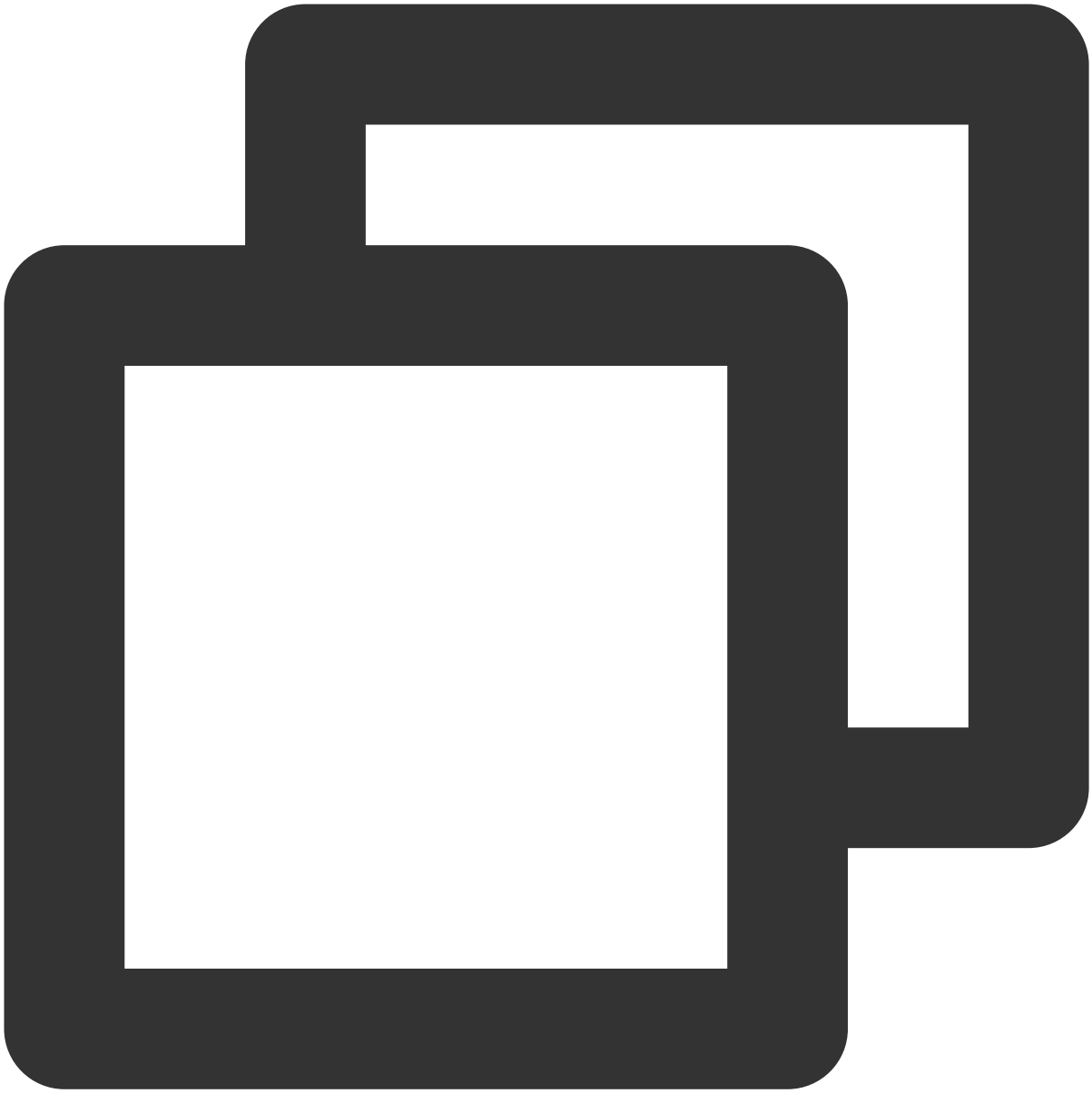
```
{
    "UserTags": [
        {
            "To_Account": "xiaojun012",
            "Tags": ["a", "b"]
        },
        {
            "To_Account": "xiaojun013",
            "Tags": ["a", "b"]
        }
    ]
```

```
}
```

Request Fields

Field	Type	Required	Description
To_Account	String	Yes	Target user account.
Tags	Array	Yes	Tag set.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description

ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags
Adding User Tags
Deleting User Tags
Deleting All User Tags

Recalling Push

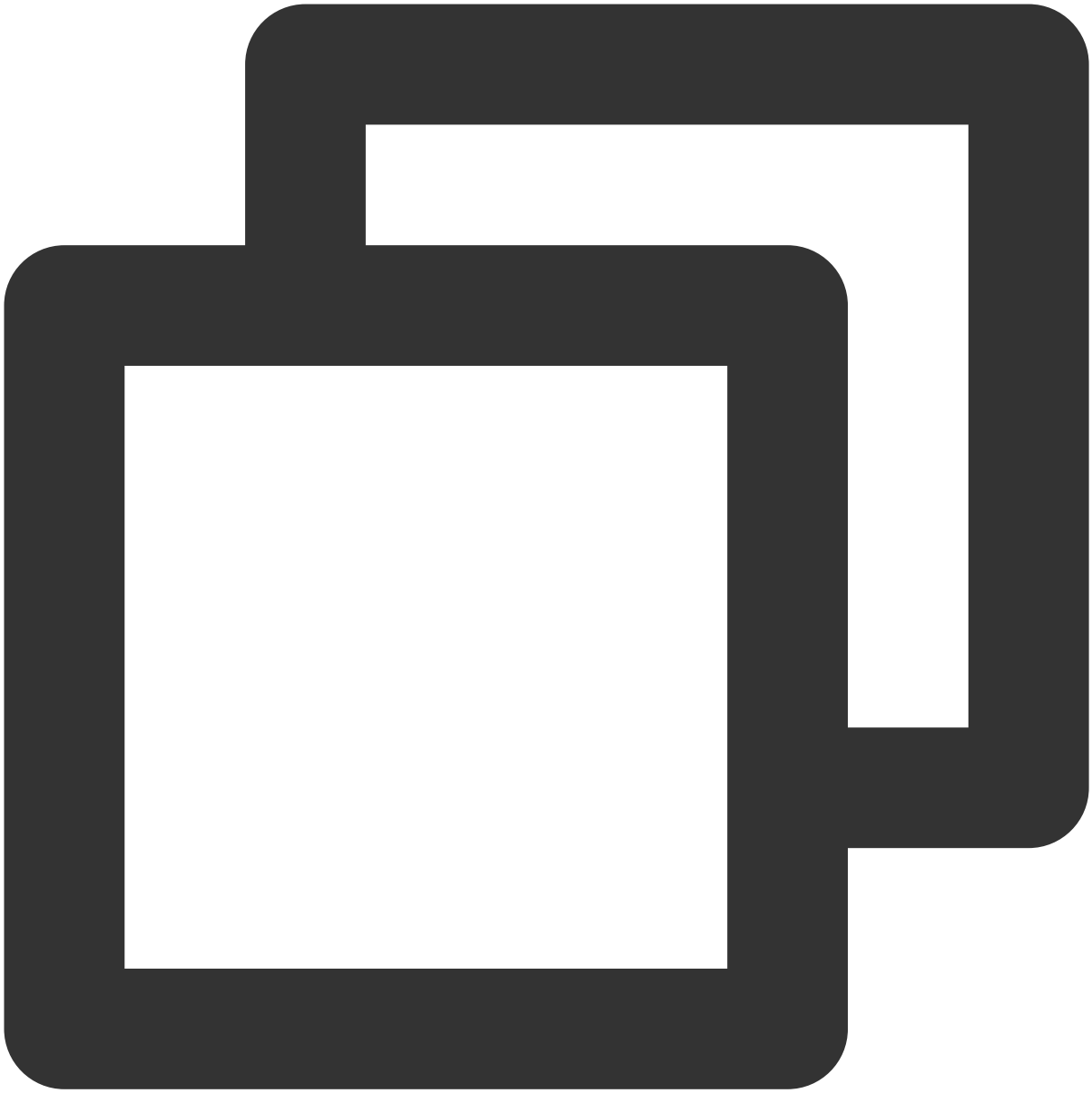
Deleting User Tags

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to delete tags for users. Up to 100 users' tags can be deleted at a time.

Sample Request URL



```
https://xxxxxx/v4/timpush/remove_tag?usersig=xxx&identifier=admin&sdkappid=88888888
```

Request Parameters

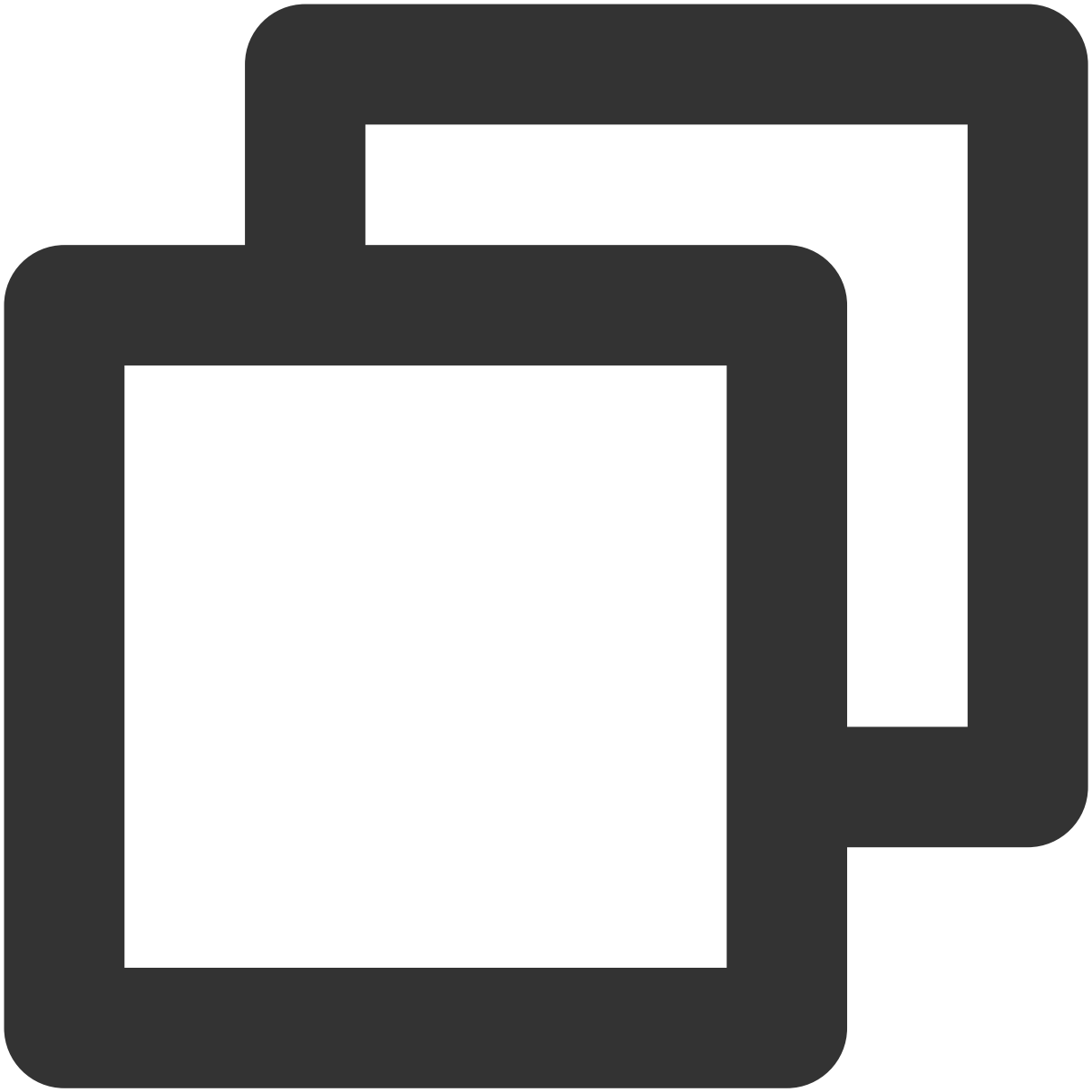
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/remove_tag	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets



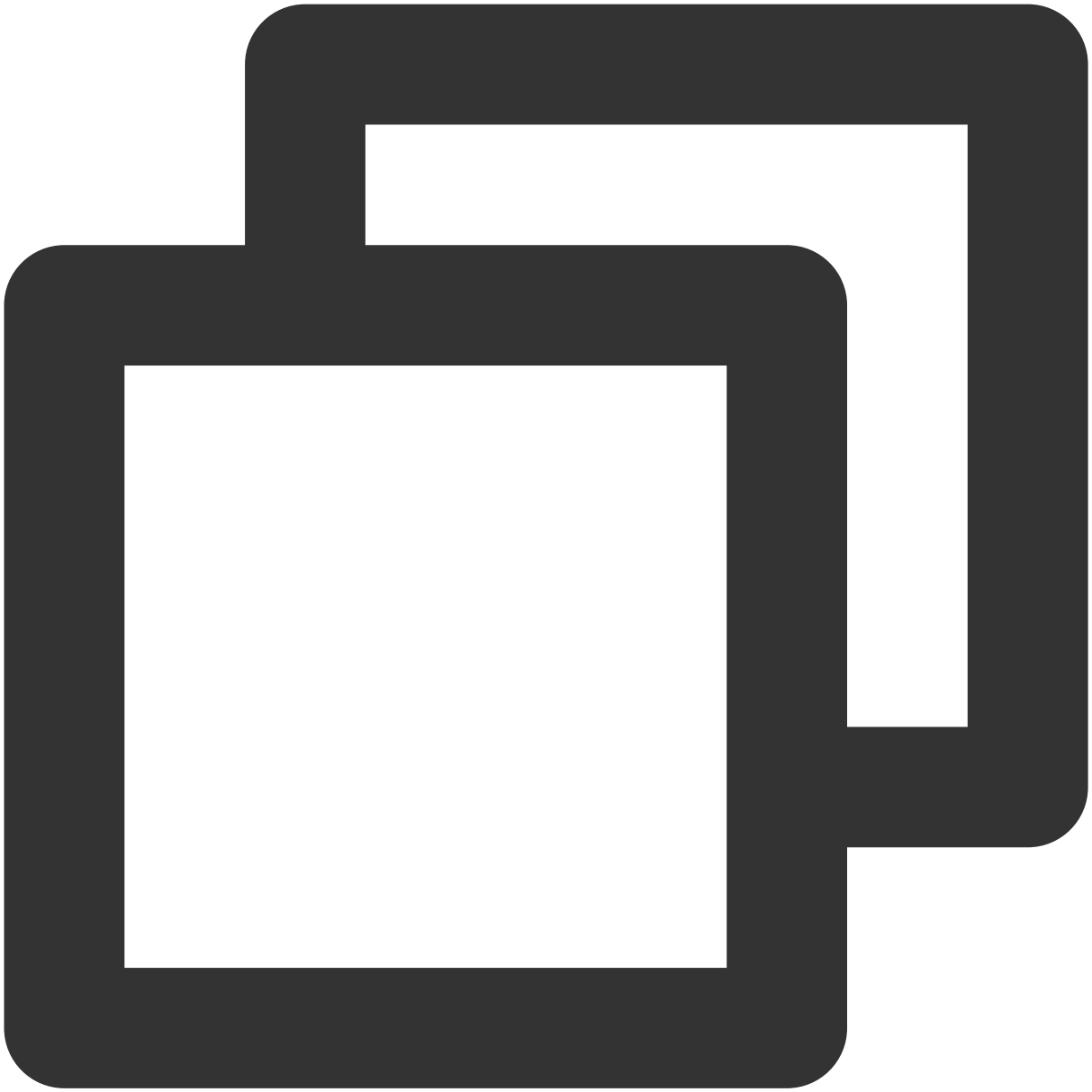
```
{
    "UserTags": [
        {
            "To_Account": "xiaojun012",
            "Tags": ["a", "b"]
        },
        {
            "To_Account": "xiaojun013",
            "Tags": ["a", "b"]
        }
    ]
```

```
}
```

Request Fields

Field	Type	Required	Description
To_Account	String	Yes	Target user account.
Tags	Array	Yes	Tag set.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description

ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes
Obtaining User Tags
Adding User Tags
Deleting User Tags

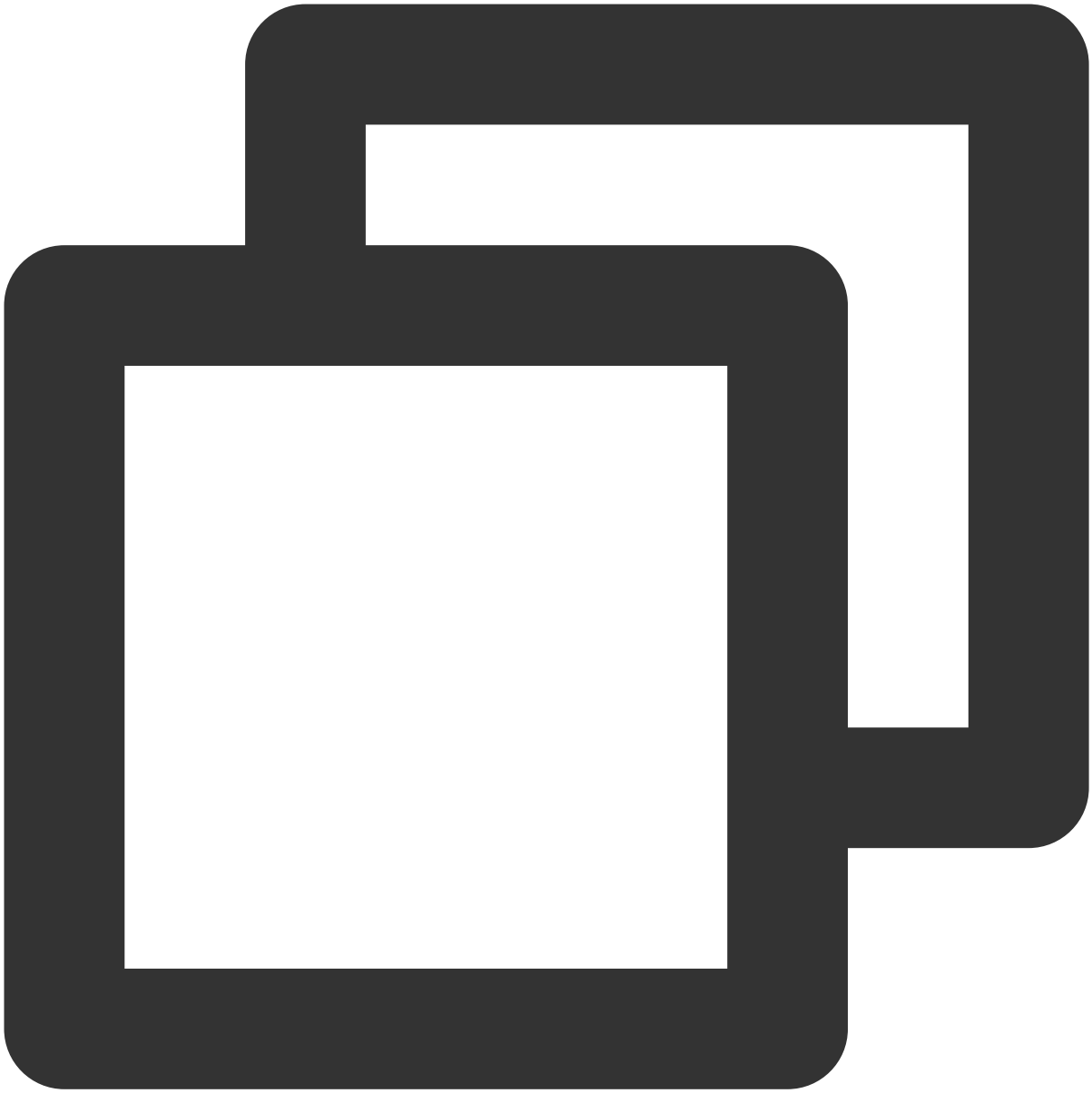
Deleting All User Tags

Last updated：2024-07-16 11:40:03

Feature Overview

This API is used by the administrator to delete all tags for users. Up to 100 users' tags can be deleted at a time.

Sample Request URL



```
https://xxxxxx/v4/timpush/clear_all_tags?usersig=xxx&identifier=admin&sdkappid=8888
```

Request Parameters

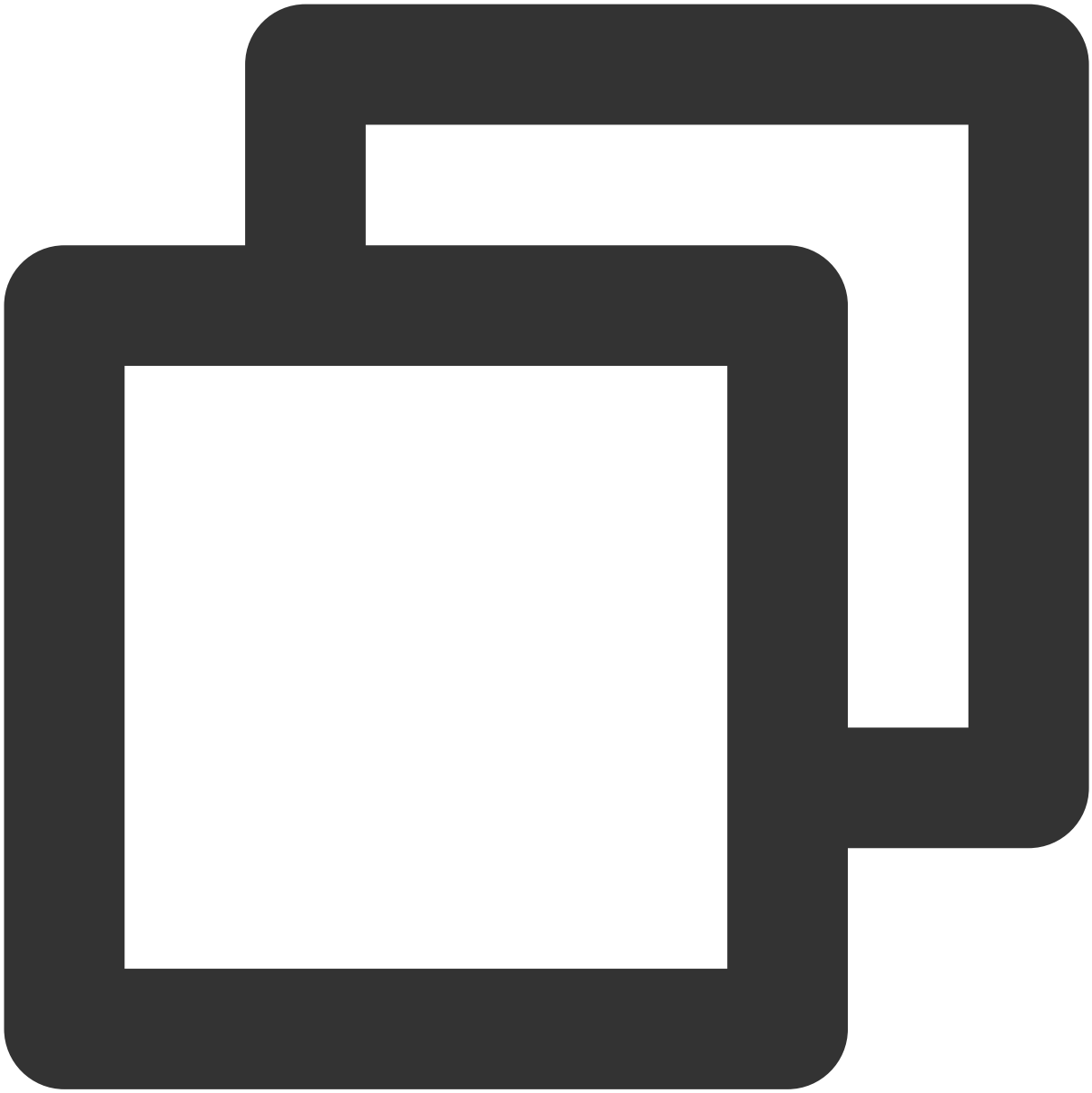
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID.

	China: `console.tim.qq.com` Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/clear_all_tags	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

100 times/second

Sample Request Packets



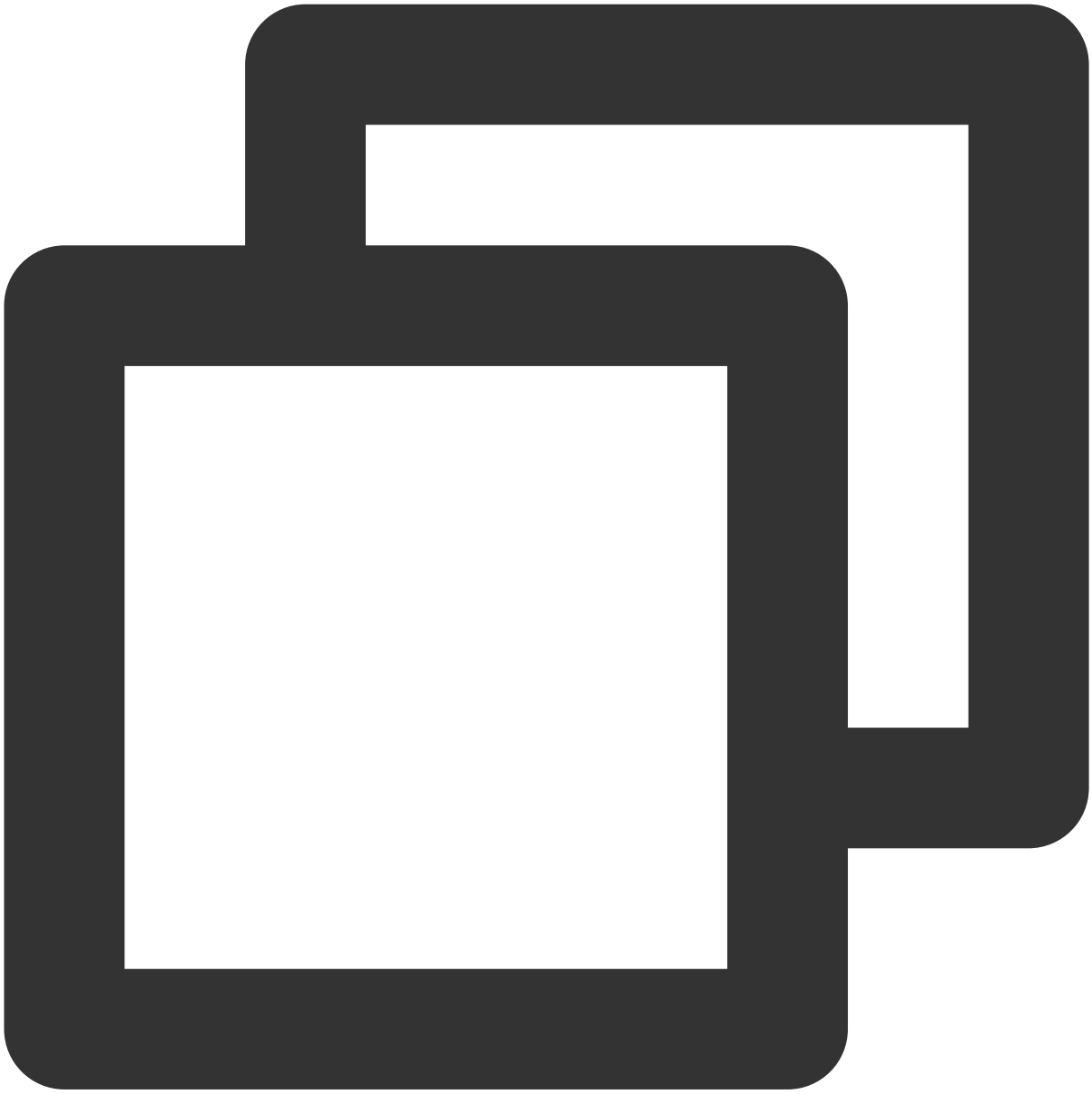
```
{
    "To_Account": [
        "xiaojun012",
        "xiaojun013"
    ]
}
```

Request Fields

Field	Type	Required	Description

To_Account	Array	Yes	Target user account.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description
ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90018	The number of requested accounts exceeds the limit.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users
Setting Application Attribute Names
Obtaining Application Attribute Names
Setting User Attributes
Deleting User Attributes
Obtaining User Attributes

Recalling Push

Last updated：2024-07-16 11:40:03

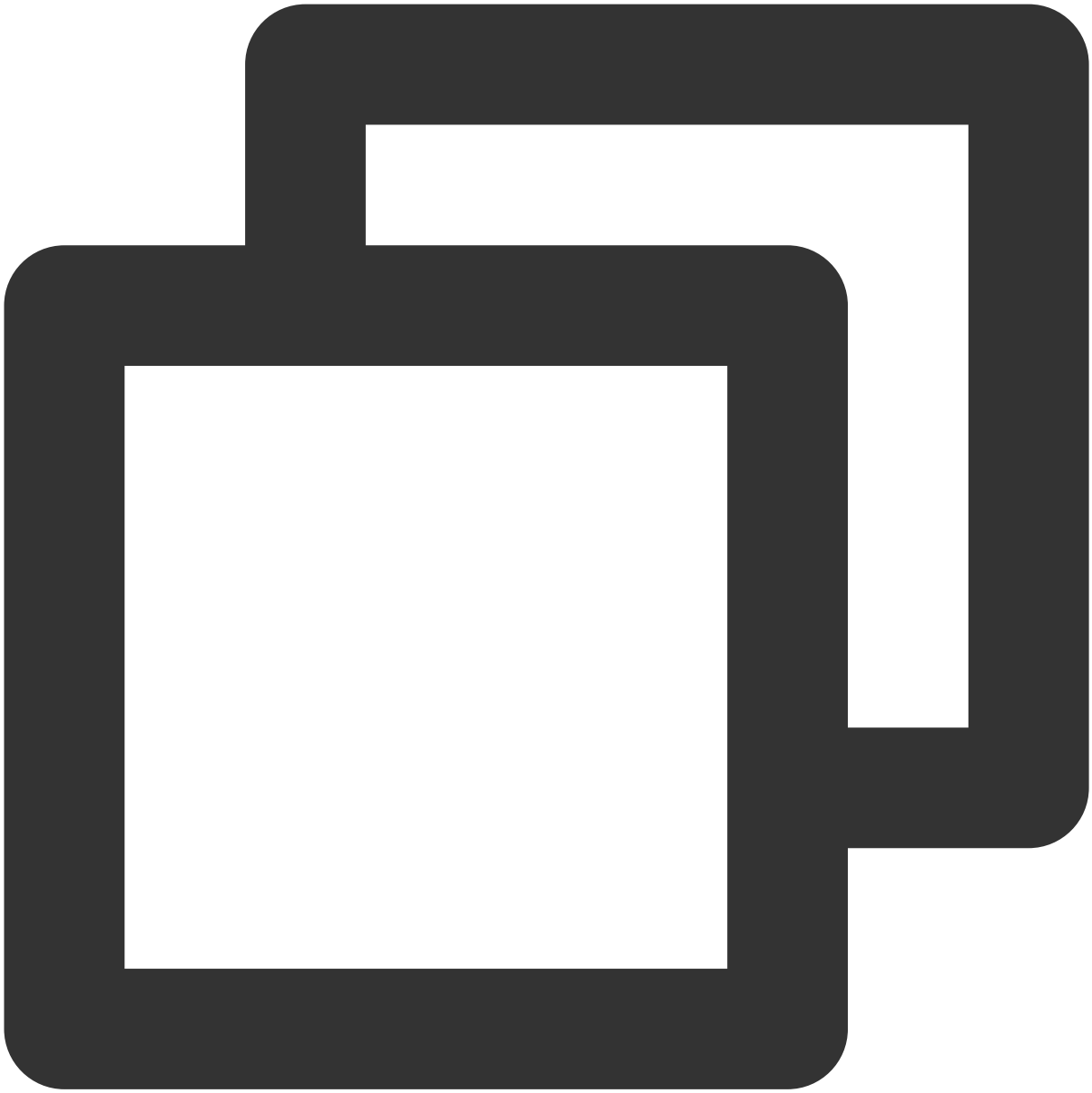
If there is an error in the content pushed to all/tagged users, it will have a negative impact on the product when end users view or click the content. In this case, it is necessary to handle it in a timely manner. You can choose to recall the push.

Feature Overview

Termination: The delivery of push tasks takes some time, and the delivery to accounts for which the tasks have not been delivered will be terminated.
Recall: For accounts for which the tasks have been delivered, unread and roaming messages can be recalled.
Overwrite: If an account has received an offline push, that push can be overwritten.
This API supports the the termination, recall, and overwrite of push tasks for all or tagged users. In the following, termination/recall/overwrite will be referred to as recall by default.
The validity period for recall is 24 hours, calculated from the time of task initiation. Push tasks that exceed 24 hours cannot be recalled.

API Call Description

Sample Request URL



```
https://xxxxxx/v4/timpush/revoke?usersig=xxx&identifier=admin&sdkappid=88888888&ran
```

Request Parameters

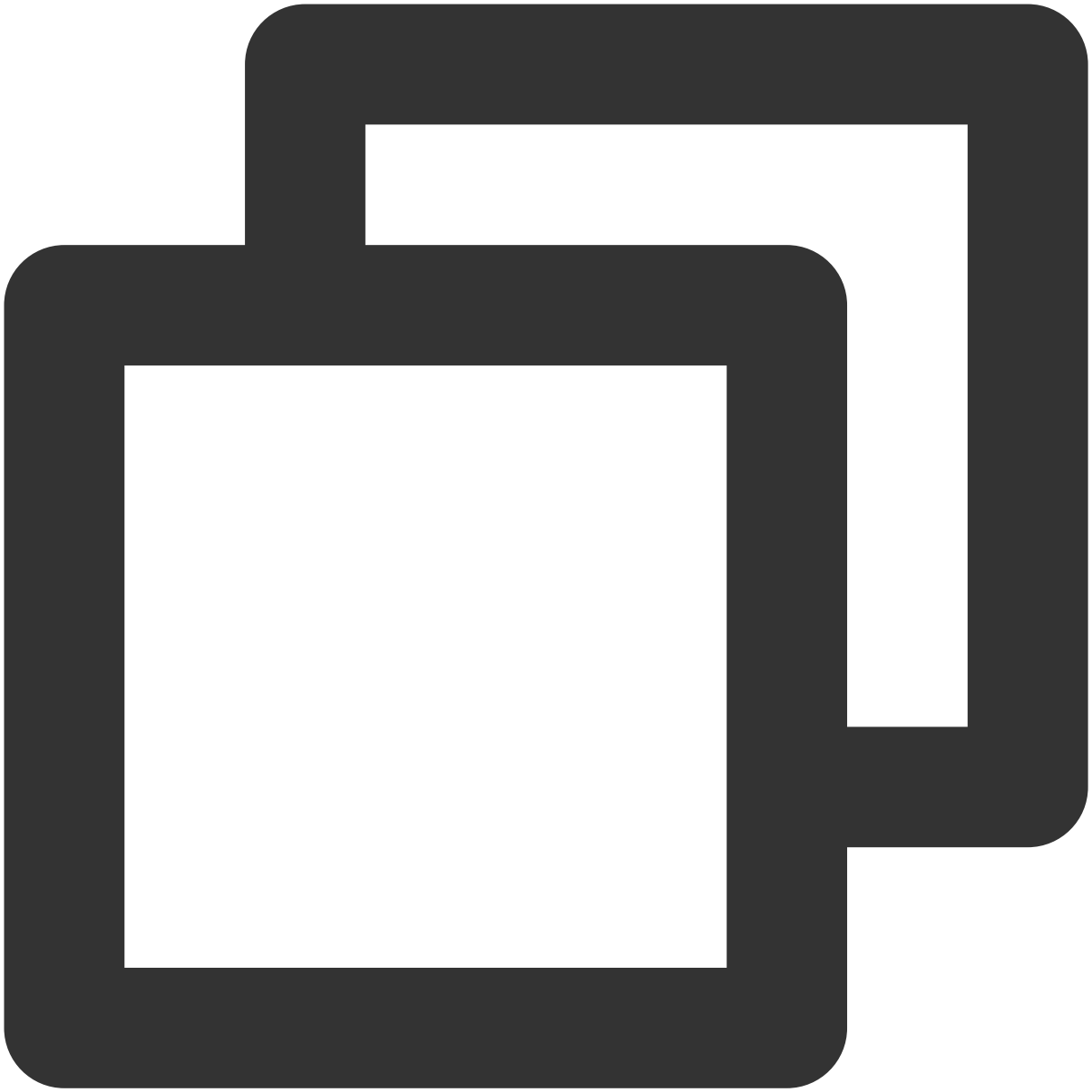
Parameter	Description
https	The request protocol is HTTPS and the request method is POST.
xxxxxx	The dedicated domain name corresponding to the country/region of the SDKAppID. China: `console.tim.qq.com`

	Singapore: `adminapisgp.im.qcloud.com` Seoul: `adminapikr.im.qcloud.com` Frankfurt: `adminapiger.im.qcloud.com` Silicon Valley: `adminapiusa.im.qcloud.com` Jakarta: `adminapiidn.im.qcloud.com`
v4/timpush/revoke	Request API.
usersig	The signature generated by the app admin account. For details, see Generating UserSig.
identifier	The app admin account.
sdkappid	The SDKAppID assigned by the Chat console when an application is created.
random	A random 32-bit unsigned integer.
contenttype	The value is always `json` .

Maximum Call Frequency

1 time/second

Sample Request Packets



```
{
    "TaskId": "660cc447_537ed82a_200000cd7ee17f5_84035729_bc614e",  // TaskId of pu
     "OfflinePushInfo": {   // If roaming/unread messages are not specified to be s
                            // OfflinePushinfo must be included during recall.
        "Title": "recall title",
        "Desc": "The other party has recalled a message."
    }
}
```

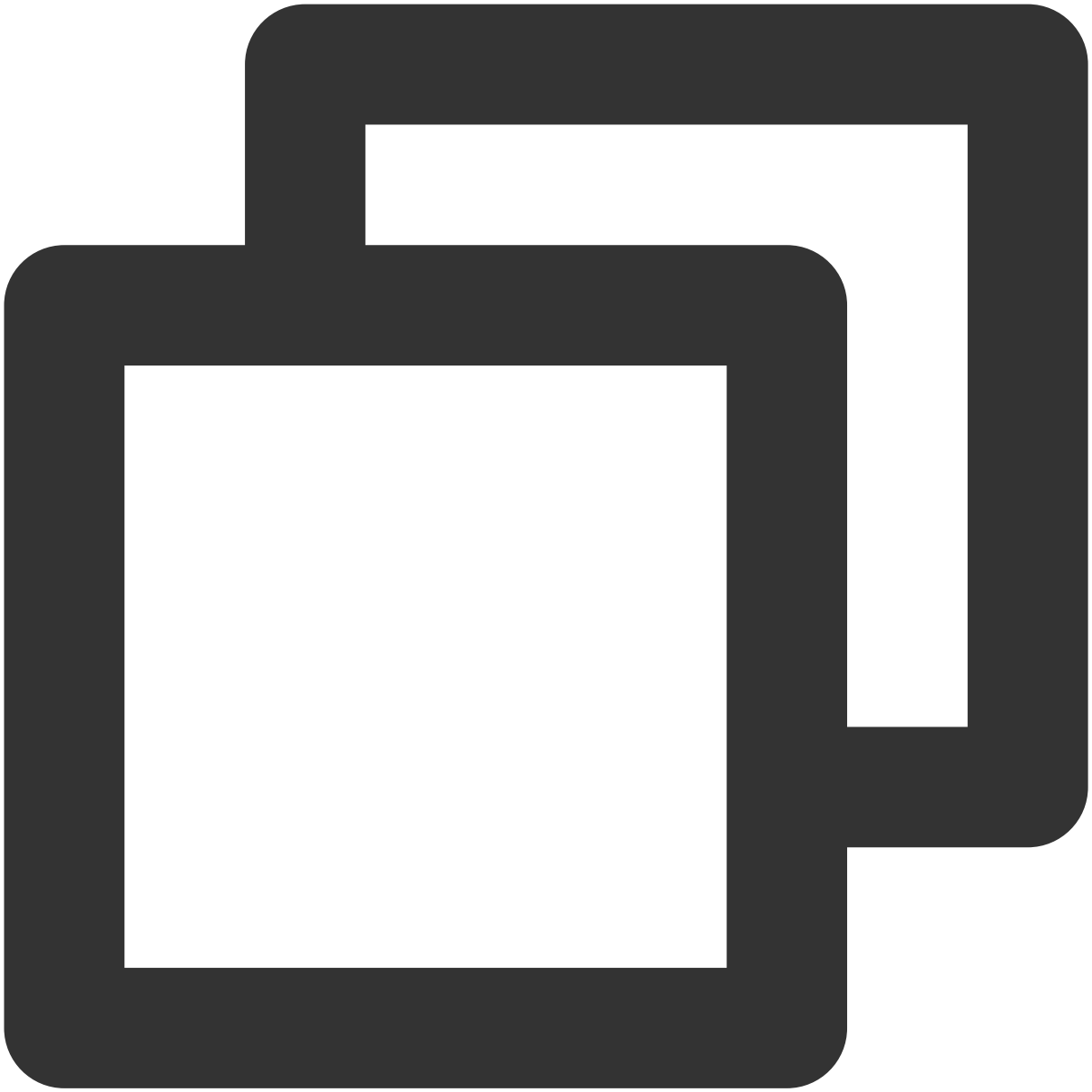
Note:

1. Vendors that support offline push overwrite: APNS/Google FCM/Huawei/Honor. Offline push from other vendors does not support overwrite. (Google FCM's notification mode supports overwrite, while the data mode currently does not support.)
2. During recall, if the recipient is in the foreground and the offline push (notification message) is read by default, the offline push will not be overwritten.

Request Fields

Field	Type	Required	Description
TaskId	String	Yes	Push task ID.
OfflinePushInfo	Object	No	The information to be pushed offline. For more information, see Message Formats. **Note:** If OfflinePushinfo.PushFlag=1 or OfflinePushInfo is not set, offline push will not be overwritten.

Sample Response Packets



```
{
    "ActionStatus": "OK",
    "ErrorInfo": "",
    "ErrorCode": 0
}
```

Response Fields

Field	Type	Description

ActionStatus	String	Processing result. `OK` : succeeded. `FAIL` : failed.
ErrorCode	Integer	Error code.
ErrorInfo	String	Error information.

Error Codes

Unless a network error (such as error 502) occurs, the HTTP return code for this API is always 200. **`ErrorCode` and `ErrorInfo` in the response packets represent the actual error code and error information.** For common error codes (60000 to 79999), see Error Codes.

The following table describes the error codes specific to this API:

Error Code	Description
90001	Failed to parse the JSON format. Check whether the request packets meet JSON specifications.
90009	The request requires app admin permissions.
90049	Recall TaskId is illegal and there is no push record. The returned TaskId can only be used for recall when the task is pushed through the timpush/push API.
90050	Repeated recall. Push tasks that have already been recalled cannot be called repeatedly.
90051	Recall is too frequent. The recall frequency limit is 1 time per second.
90052	The recall validity period has been exceeded. The recall must be within 24 hours, and push tasks that exceed 24 hours cannot be recalled.
90053	Invalid recall. Push with OnlineOnlyFlag=0, but the recall does not include OfflinePushInfo.
91000	Internal service error. Try again.

API Debugging Tool

Use the RESTful API online debugging tool to debug this API.

References

Pushing to All/Tagged Users

Advanced Features
Custom Definition Badge

Last updated：2024-08-01 11:33:30

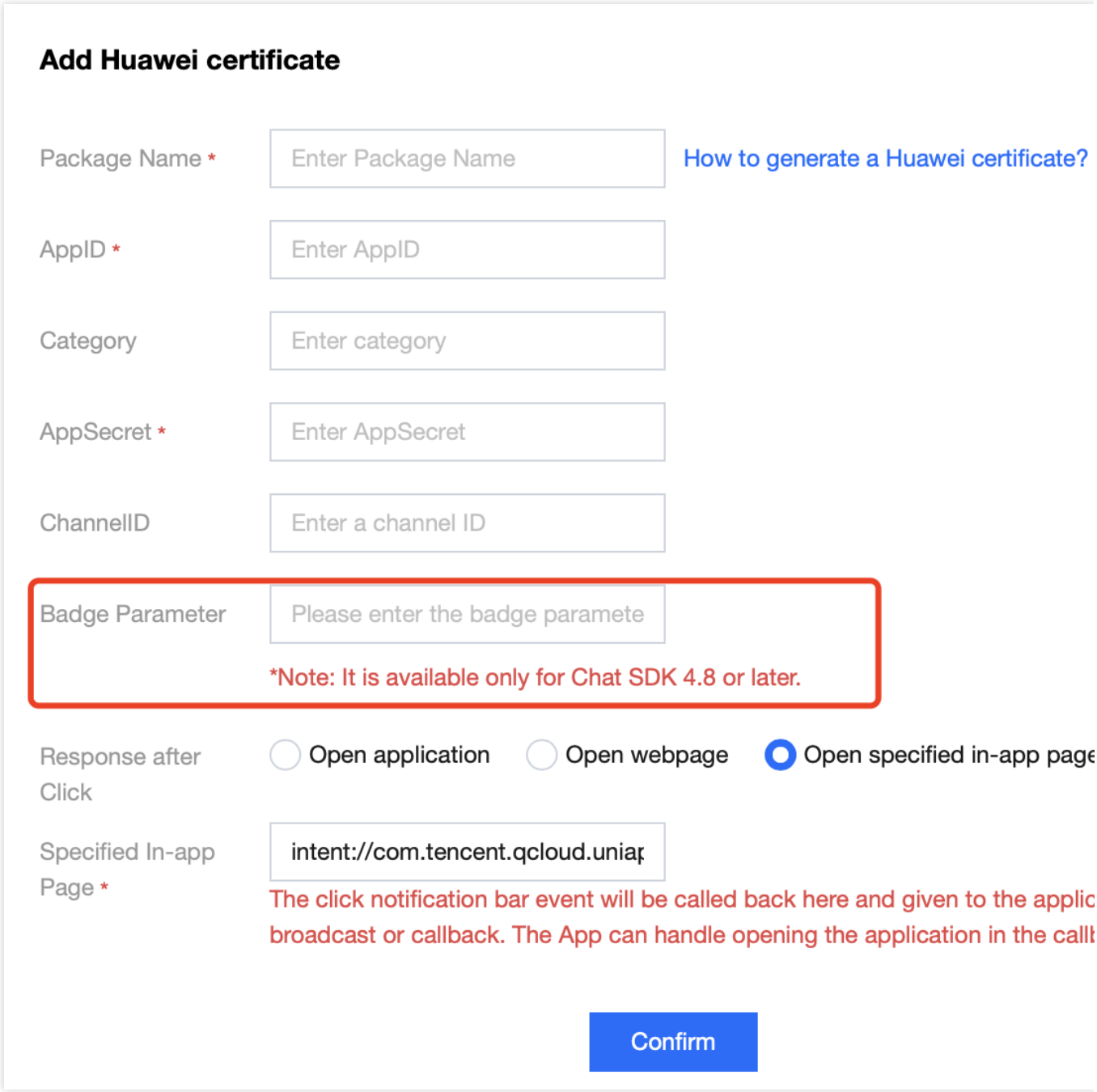
Android

Supported Vendors

Huawei.

Configuration Method

To configure the Huawei badge parameters in the console, set them to the application's startup class, for example, "com.tencent.qcloud.tim.demo.SplashActivity". The component will automatically parse and update the badge; otherwise, it will not update the badge.



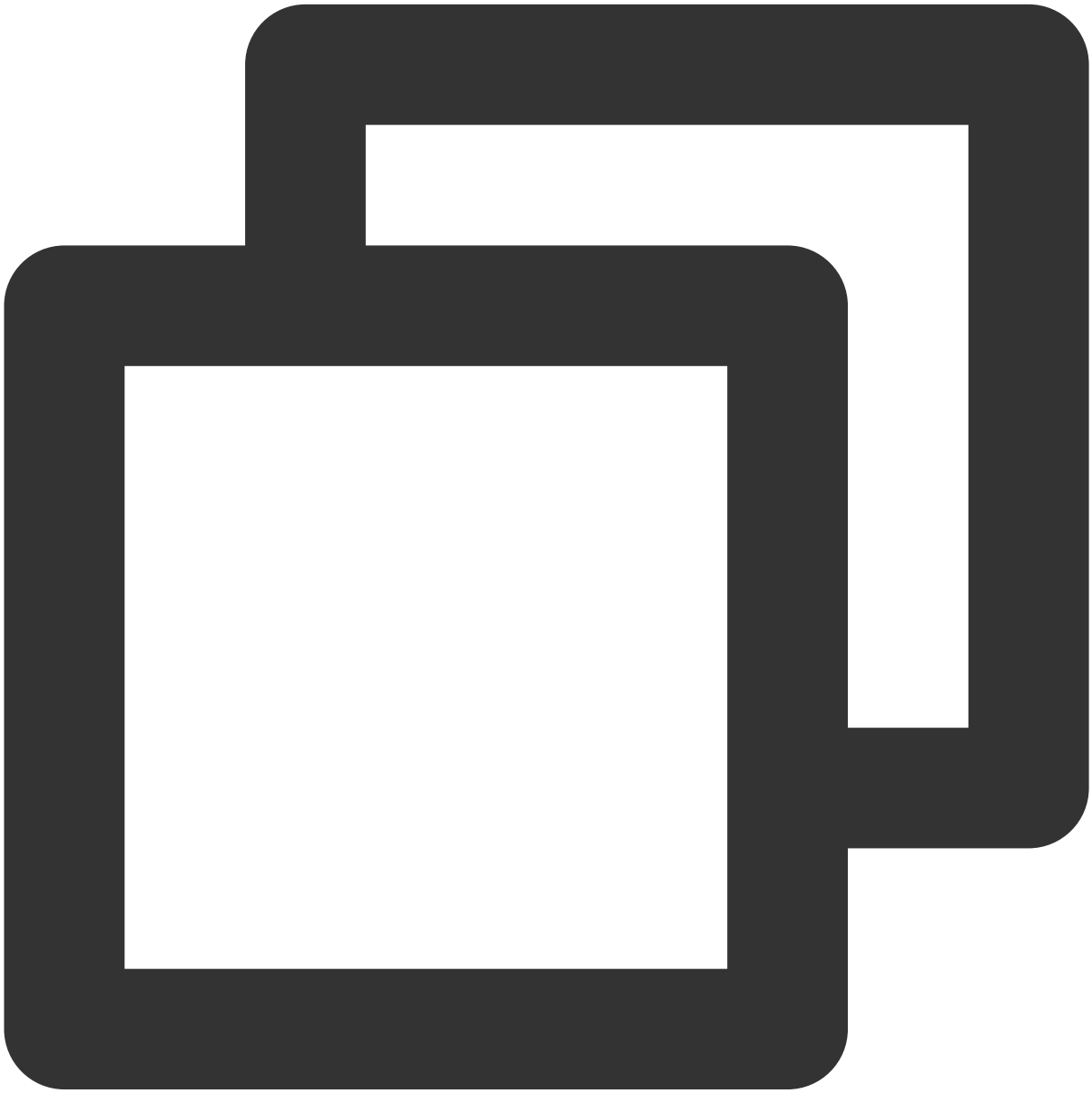
iOS

By default, when the App goes into the background, the IMSDK will set the total number of unread IM messages as the badge. If the App is integrated with offline push, when a new offline push notification is received, the App badge will increment by 1 based on the baseline badge (default is the total number of unread IM messages, or the custom-defined badge if one has been set).

Configuration Method

If you want to customize the badge, follow these steps:

1. The App calls the - (void)setAPNSListener:(id<V2TIMAPNSListener>)apnsListener interface to set the listener.
2. The App implements the - (uint32_t)onSetAPPUnreadCount interface and returns the custom-defined badge number.



```
// 1. Set the listener
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    // Listen for push notifications
    [V2TIMManager.sharedInstance setAPNSListener:self];
    // Listen for unread conversation counts
    [[V2TIMManager sharedInstance] setConversationListener:self];
```

```
    return YES;
}

// 2. Save the unread count after it changes
- (void)onTotalUnreadMessageCountChanged:(UInt64)totalUnreadCount {
    self.unreadNumber = totalUnreadCount;
}

// 3. Report custom-defined unread count after the app is pushed to the background
/** After the application enters the background, customize the app's unread count.
 *  <pre>
 *
 *   - (uint32_t)onSetAPPUnreadCount {
 *       return 100;  // Custom-defined unread count
 *   }
 *
 *  </pre>
 */
- (uint32_t)onSetAPPUnreadCount {
    // 1. Get the custom-defined badge
    uint32_t customBadgeNumber = ...

    // 2. Add the IM message unread count
    customBadgeNumber += self.unreadNumber;

    // 3. Report to the IM server via IMSDK
    return customBadgeNumber;
}
```

Flutter

For configuration, please refer to the iOS and Android sections below. The methods called are also available in the Flutter version of the IM SDK, with the same names.

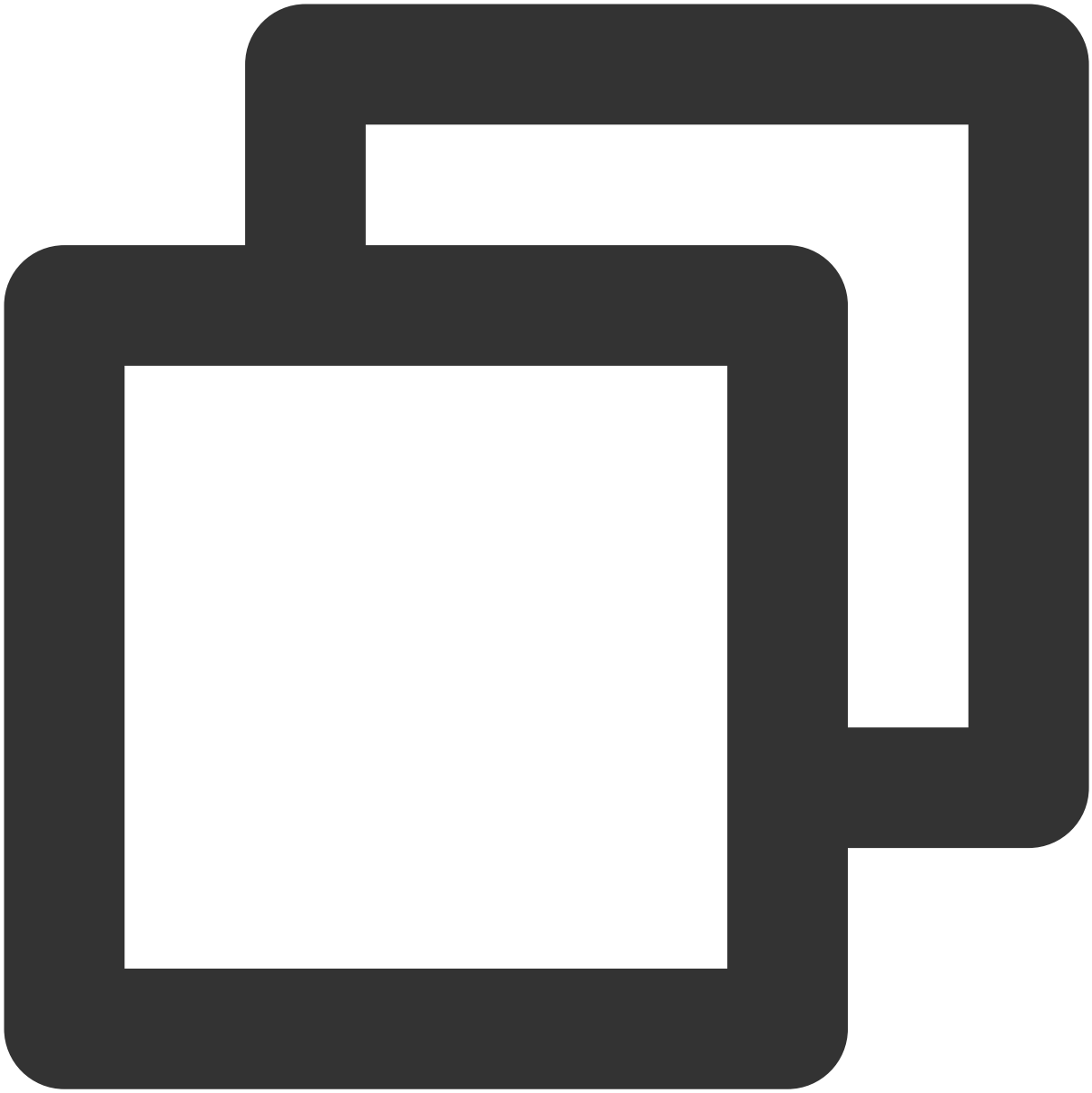
Custom Definition Ringtone

Last updated：2024-08-01 11:36:20

Android

Pre-8.0 systems

1. Custom ringtone resource files for Android should be added to the project's raw directory; for iOS, link them into the Xcode project.
2. Please use the message invocation interfaces setAndroidSound() and setIOSSound().



```
V2TIMOfflinePushInfo v2TIMOfflinePushInfo = new V2TIMOfflinePushInfo();
v2TIMOfflinePushInfo.setAndroidSound("Ringtone Name");
v2TIMOfflinePushInfo.setIOSSound("Ringtone Name.mp3");

String msgID = V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, isGroup ?
 V2TIMMessage.V2TIM_PRIORITY_DEFAULT, false, v2TIMOfflinePushInfo, new V2TIMSendCal
@Override
 public void onProgress(int progress) {

 }
```

```
@Override
 public void onError(int code, String desc) {

 }

@Override
 public void onSuccess(V2TIMMessage v2TIMMessage) {

 }
});
```

Note:

Supported in IMSDK v6.1.2155 or above.

The interface supports Huawei, Xiaomi, FCM, and APNS.

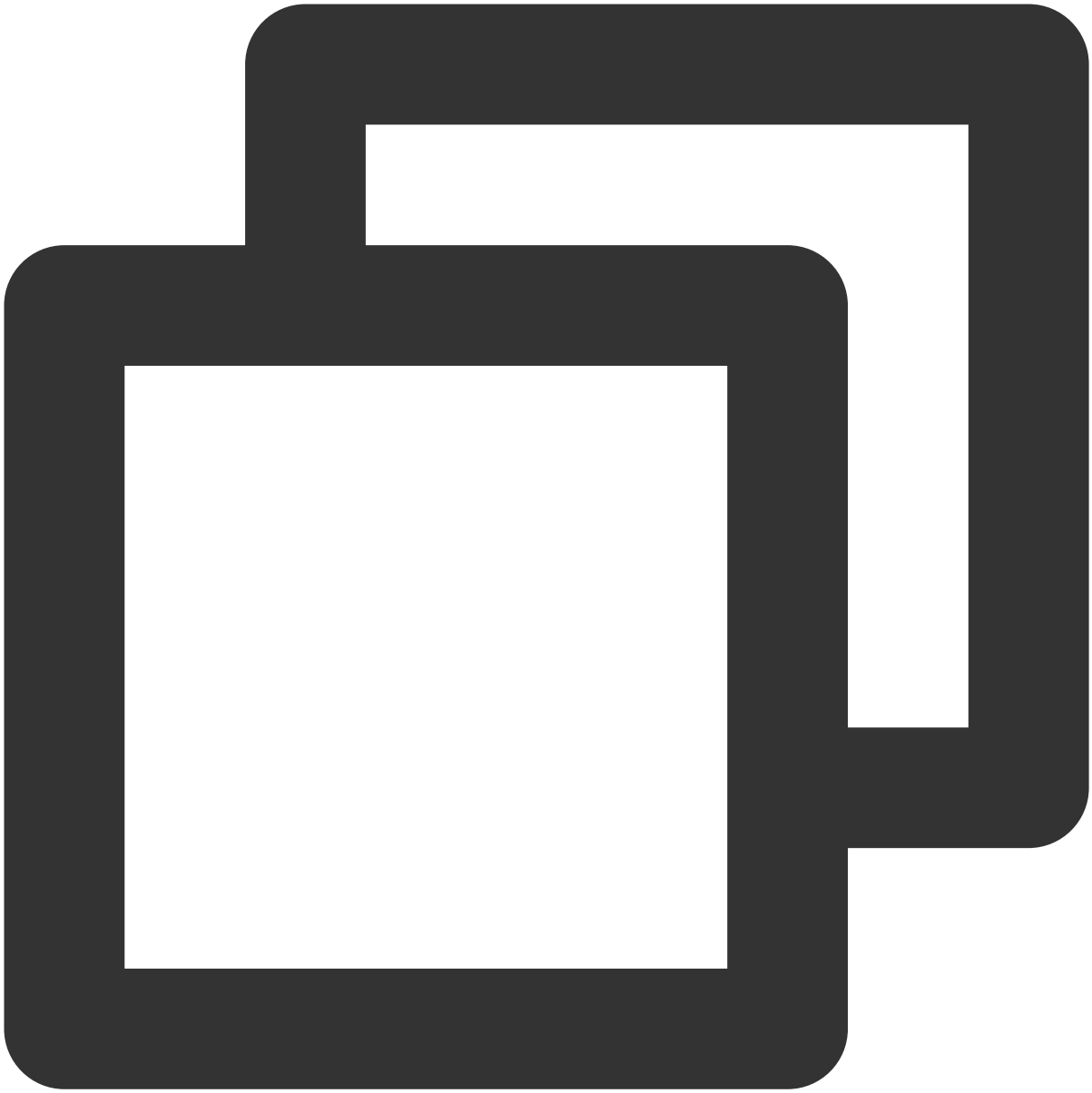
System version 8.0 and later

Huawei and APNs

Huawei, APNS use setAndroidSound() and setIOSSound() to set the offline push ringtone prompt.

OPPO

1. Locally create a custom notification channel



```
// Example
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    NotificationManager nm = (NotificationManager) context.getSystemService(Context
    NotificationChannel notificationChannel =
            new NotificationChannel("channelId", "channelName", NotificationManager
    notificationChannel.enableLights(true);
    notificationChannel.enableVibration(true);
    notificationChannel.setShowBadge(true);
    notificationChannel.setLockscreenVisibility(Notification.VISIBILITY_PUBLIC);

    // "android.resource://package_name/raw/private_ring"
```

```
        notificationChannel.setSound(Uri.parse("sound"), null);
        nm.createNotificationChannel(notificationChannel);
    }
```

2. Use the created channel

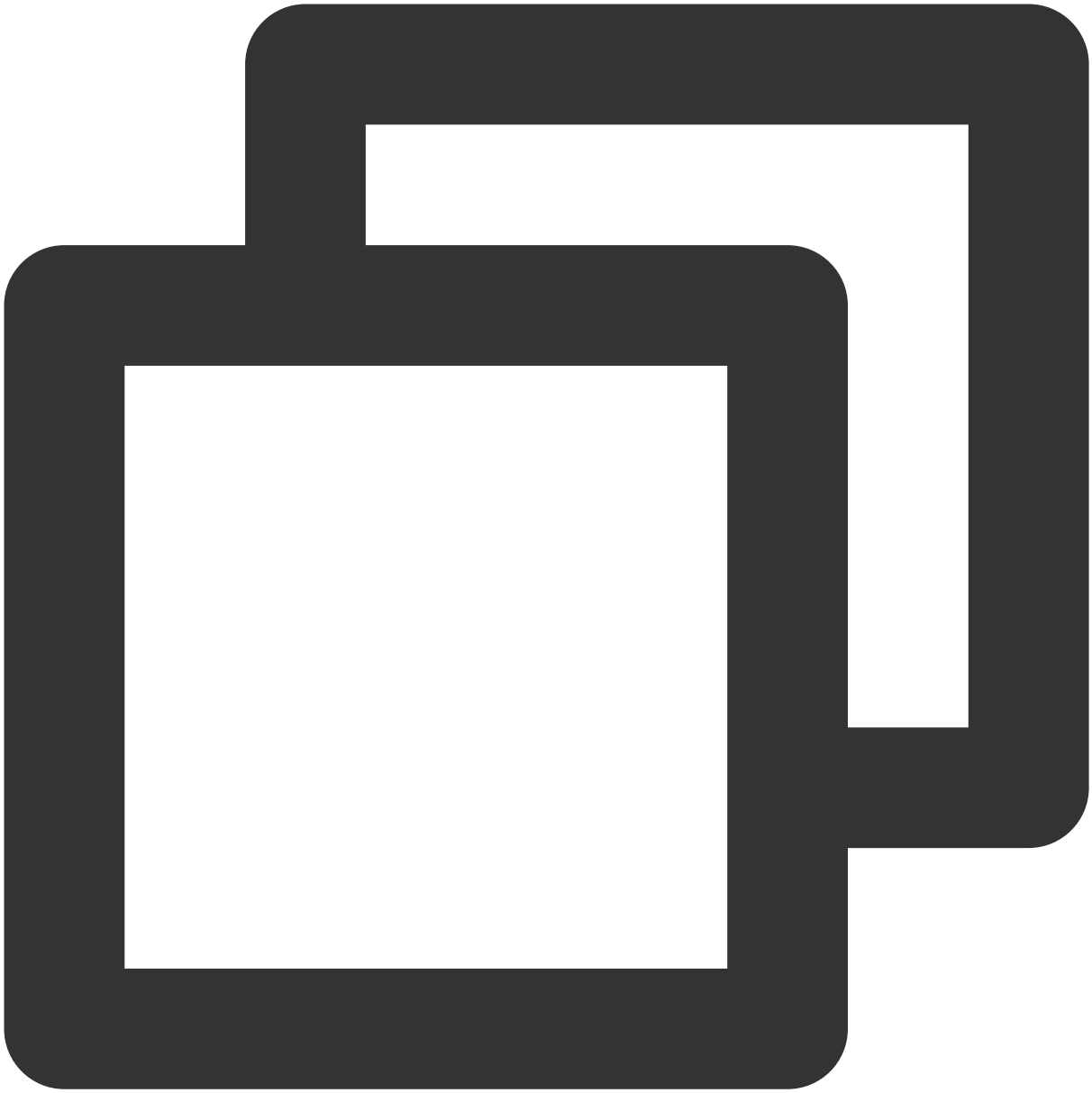
Create Private Message Channel - .

Push messages carry the channel ID field. For details, see setAndroidOPPOChannelID. For console settings, see Certificate Editing ChannelID field. Setting one of the two is sufficient.

Mi

1. log in to the Manufacturer Console to create a channel and configure, where the ringtone file needs to be added to the raw directory of your local Android Studio project.

2. Send a message specifying the channel ID of the custom ringtone; for details, please refer to setAndroidXiaoMiChannelID.

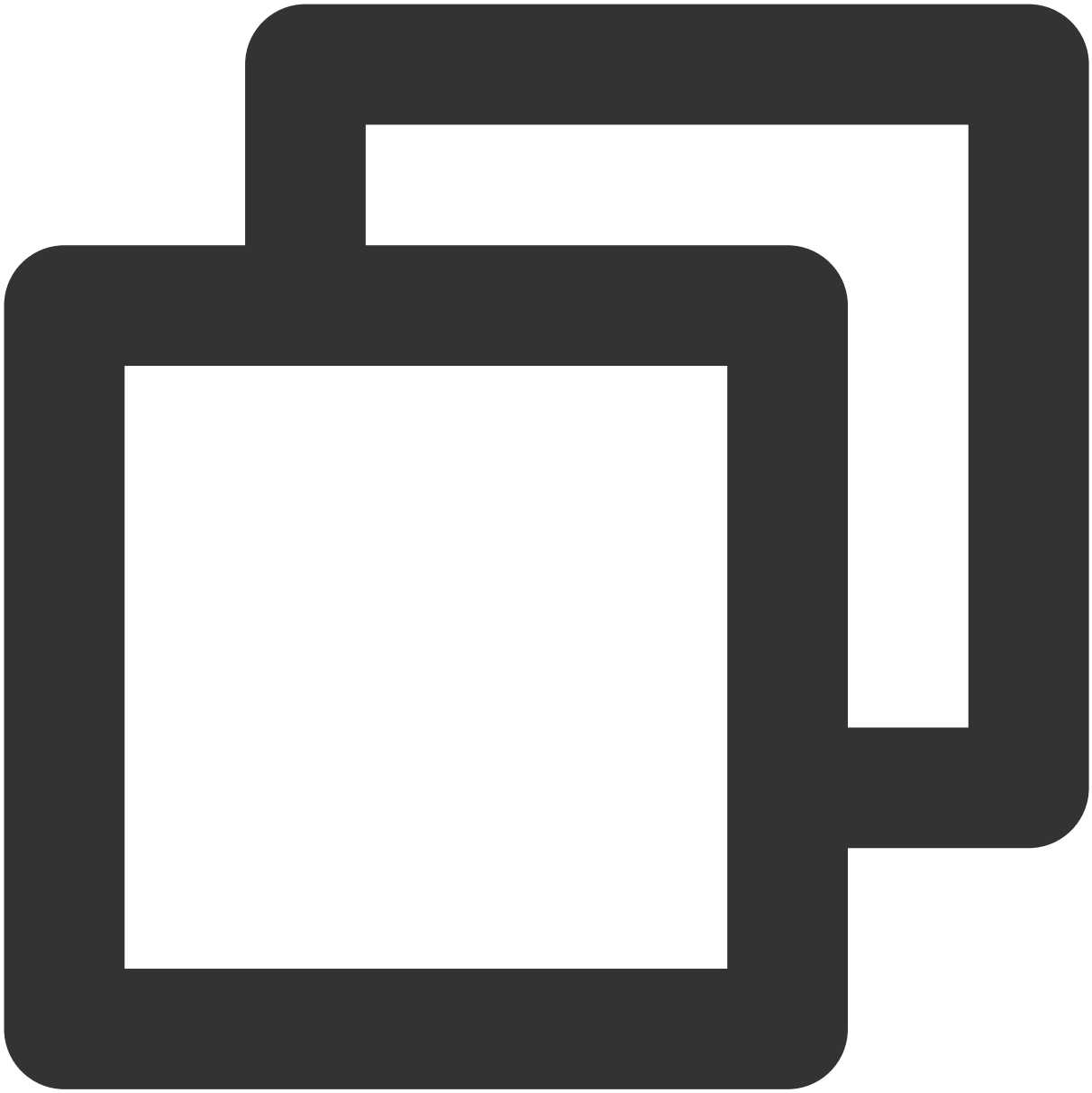


```
V2TIMOfflinePushInfo v2TIMOfflinePushInfo = new V2TIMOfflinePushInfo();
v2TIMOfflinePushInfo.setAndroidXiaoMiChannelID("Channel ID Applied by Manufacturer"

String msgID = V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, isGroup ?
 V2TIMMessage.V2TIM_PRIORITY_DEFAULT, false, v2TIMOfflinePushInfo, new V2TIMSendCal
@Override
 public void onProgress(int progress) {
    TUIChatUtils.callbackOnProgress(callBack, progress);
 }

@Override
```

```
  public void onError(int code, String desc) {
     TUIChatUtils.callbackOnError(callBack, TAG, code, desc);
  }

@Override
  public void onSuccess(V2TIMMessage v2TIMMessage) {

  }
});
```

FCM

1. Custom ringtone resource files should be added to the project's raw directory. To configure FCM's custom ringtone parameters, call before registering for push services. For details, see configFCMPrivateRing.
2. Send a message specifying the channel ID for the custom ringtone; for details, please refer to setAndroidFCMChannelID.



```
V2TIMOfflinePushInfo v2TIMOfflinePushInfo = new V2TIMOfflinePushInfo();
v2TIMOfflinePushInfo.setAndroidFCMChannelID(PrivateConstants.fcmPushChannelId);

String msgID = V2TIMManager.getMessageManager().sendMessage(v2TIMMessage, isGroup ?
 V2TIMMessage.V2TIM_PRIORITY_DEFAULT, false, v2TIMOfflinePushInfo, new V2TIMSendCal
@Override
 public void onProgress(int progress) {
    TUIChatUtils.callbackOnProgress(callBack, progress);
 }

@Override
```

```
  public void onError(int code, String desc) {
     TUIChatUtils.callbackOnError(callBack, TAG, code, desc);
  }

 @Override
  public void onSuccess(V2TIMMessage v2TIMMessage) {

  }
 });
```

Note:

Supported in IMSDK v7.0.3754 or above.

FCM custom ringtones or setting channel ID is supported only in Certificate Mode.

iOS

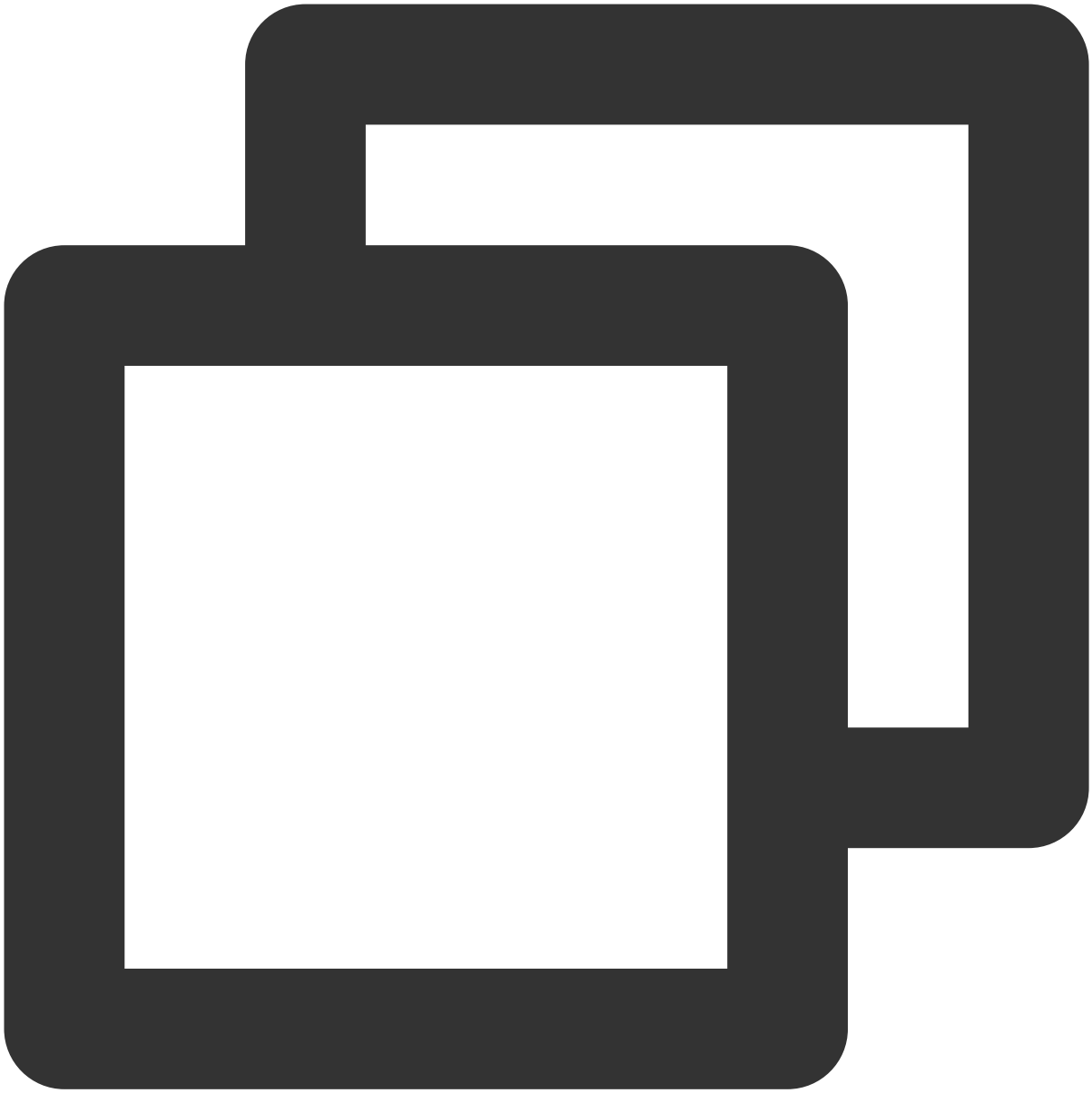
1. When calling sendMessage to send a message, set the V2TIMOfflinePushInfo's `iOSSound` field. Pass in the voice file name to `iOSSound` .

Note:

Offline push notification sound settings (effective only for iOS), when `iOSSound = kIOSOfflinePushNoSound`, it indicates no sound will be played upon receiving.

When `iOSSound = kIOSOfflinePushDefaultSound`, it indicates the system sound will be played upon receiving.

To customize `iOSSound`, you first need to link the audio file into the Xcode project, and then set the audio filename (including the extension) to `iOSSound`.



```
V2TIMOfflinePushInfo *pushInfo = [[V2TIMOfflinePushInfo alloc] init];
pushInfo.title = @"push title";
pushInfo.iOSSound = @"phone_ringing.mp3"; // your voice file's name
[[V2TIMManager sharedInstance] sendMessage:msg receiver:receiver groupID:groupID pr

} fail:^(int code, NSString *msg) {

}];
```

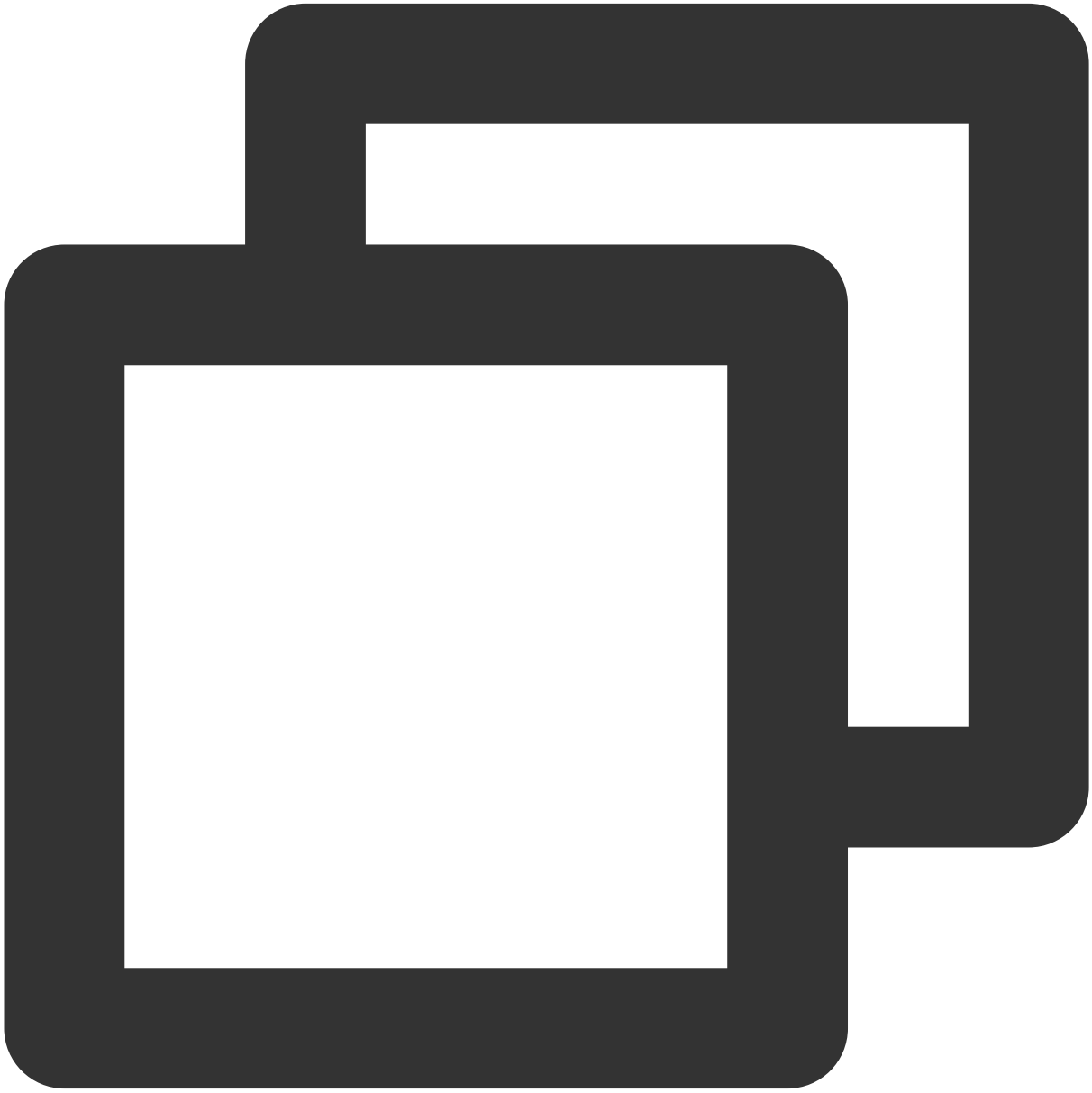
2. When calling sendMessage to send a message, set the V2TIMOfflinePushInfo's `AndroidSound` field; pass in the voice file name to `AndroidSound` .

Note:

Offline push notification sound settings (effective only for Android, supported only in imsdk 6.1 and above) are supported only on Huawei and Google phones for setting ringtone prompts.

For Xiaomi ringtone settings, please refer to: Server-side Java SDK documentation.

To customize `AndroidSound`, you first need to place the audio file in the raw directory of the Android project, and then set the `AndroidSound` with the audio filename (without the extension).



```
V2TIMOfflinePushInfo *pushInfo = [[V2TIMOfflinePushInfo alloc] init];
```

```
pushInfo.title = @"push title";
pushInfo.AndroidSound = @"phone_ringing"; // your voice file's name
[[V2TIMManager sharedInstance] sendMessage:msg receiver:receiver groupID:groupID pr

} fail:^(int code, NSString *msg) {

}];
```

uniapp

Note:

The receiver needs to integrate TencentCloud-TIMPush.

For the sender, **@tencentcloud/chat ≥ 3.3.2**.

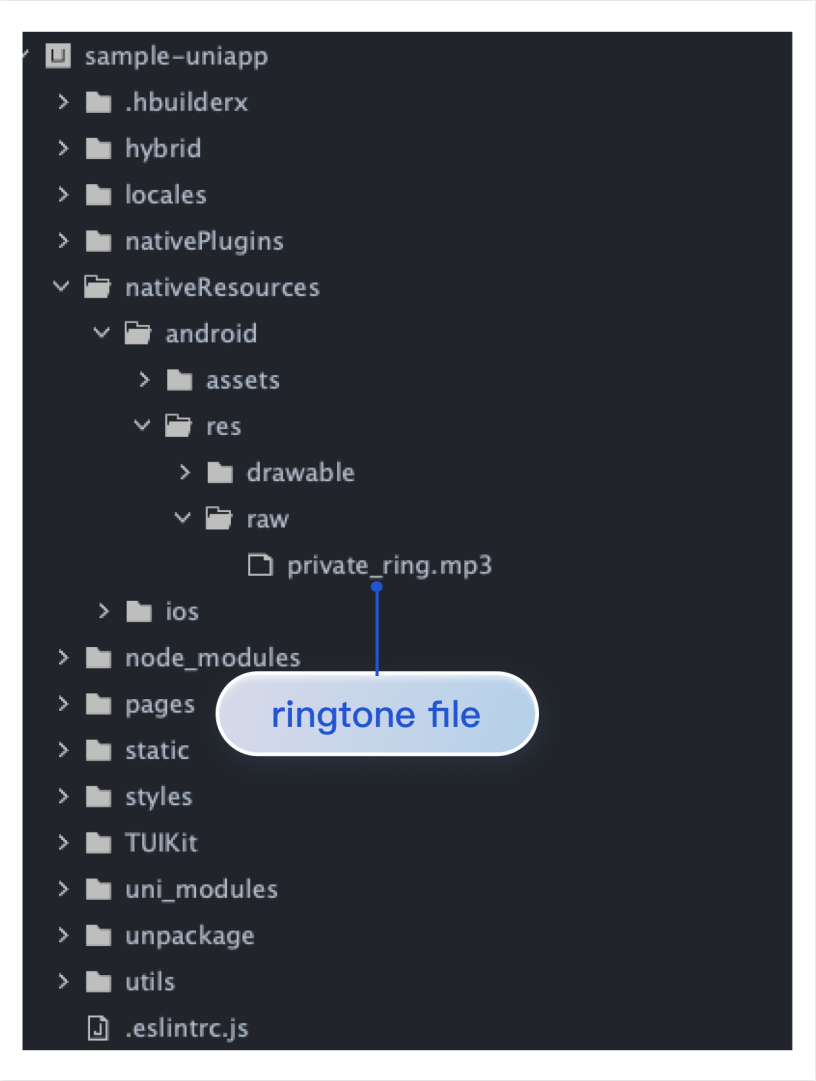
Supports Huawei, Xiaomi, FCM, and APNS.

Receiver

Android

iOS

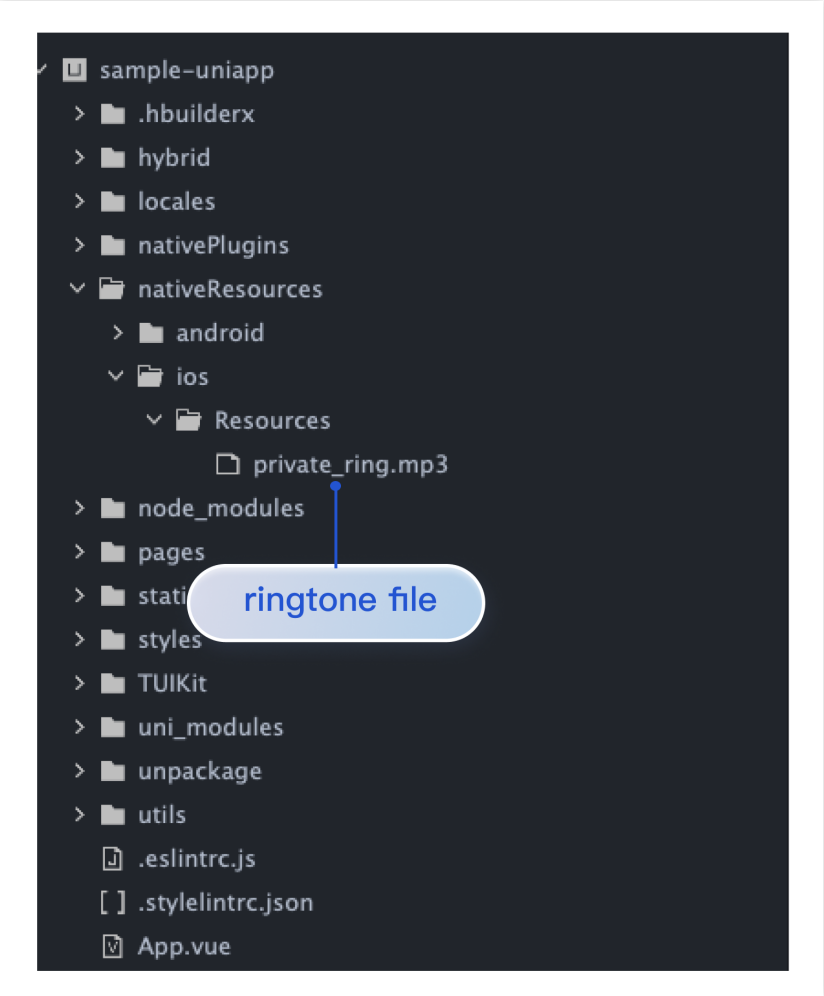
Custom ringtone resource files should be added to the project **nativeResources/android/res/raw** directory, as shown:



Note:

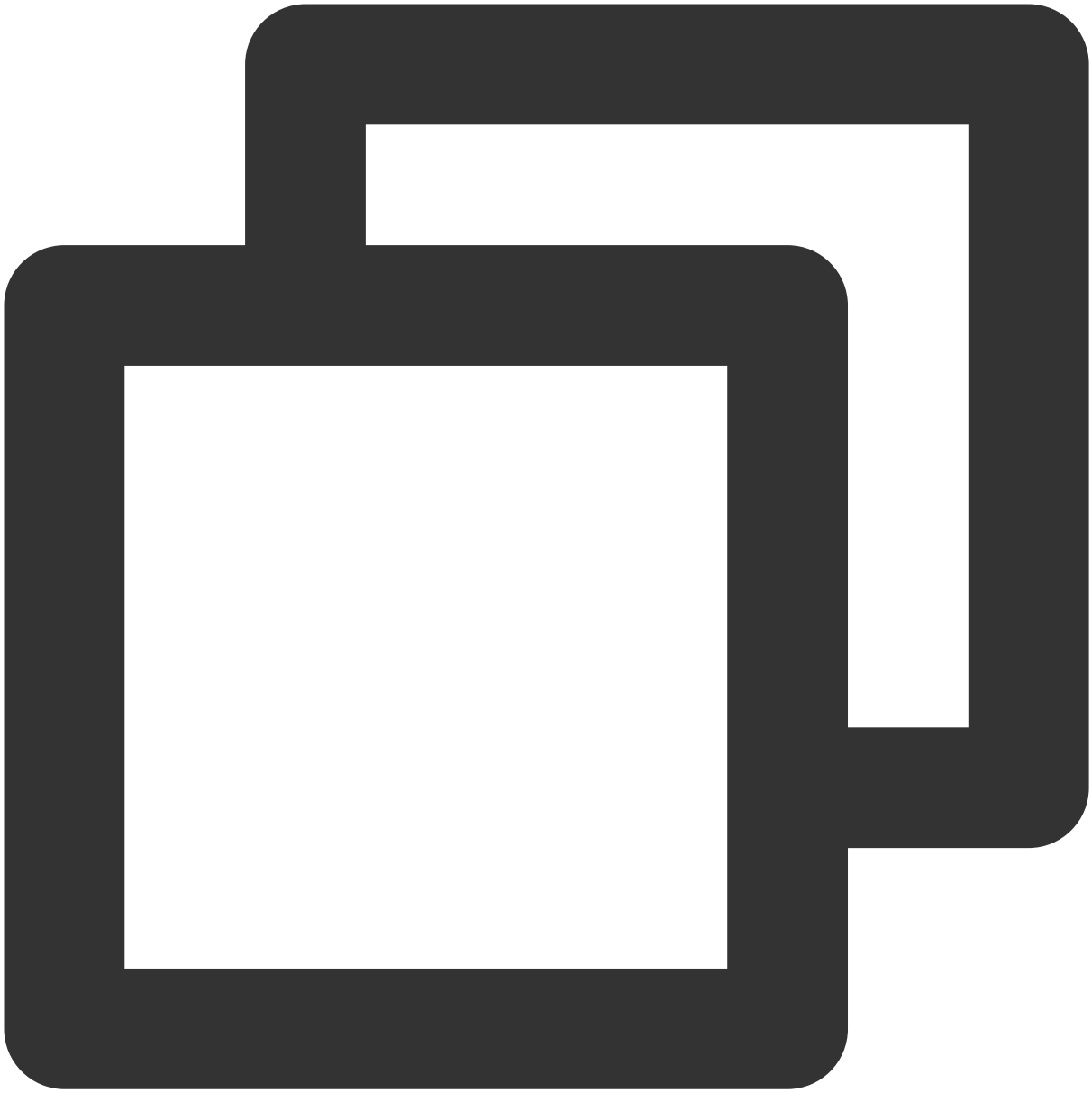
For iOS custom ringtones, the uniapp must be the official package.

Custom ringtone resource files should be added to the project **nativeResources/ios/Resources** directory, as shown:



Sender

1. Upgrade @tencentcloud/chat to the latest version.

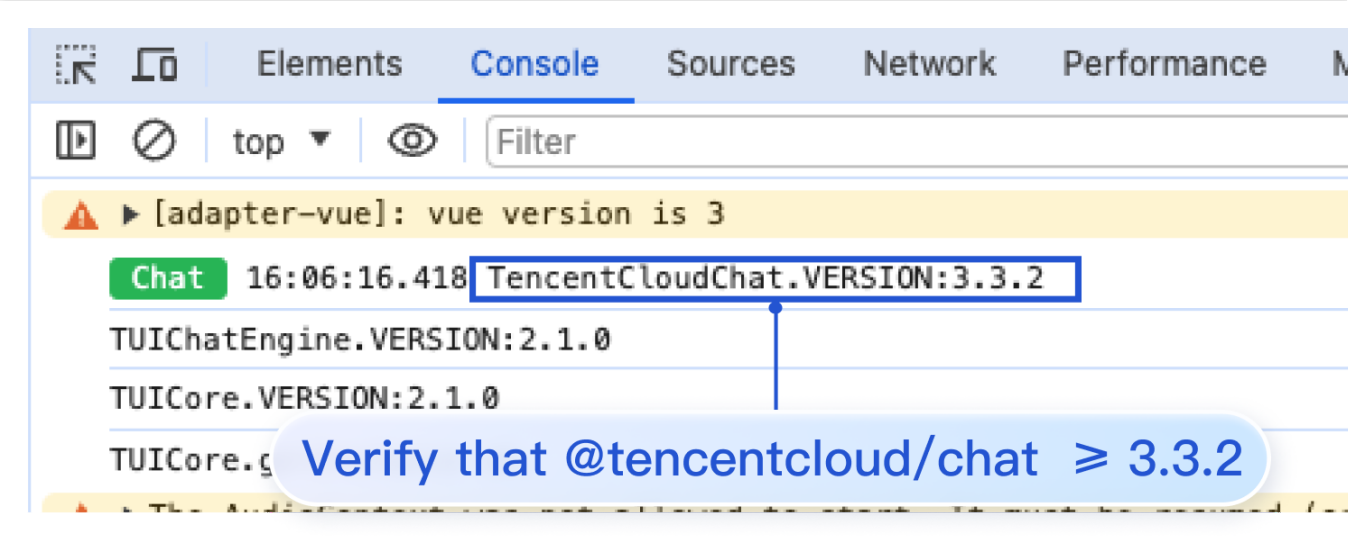


```
npm install @tencentcloud/chat@latest
```

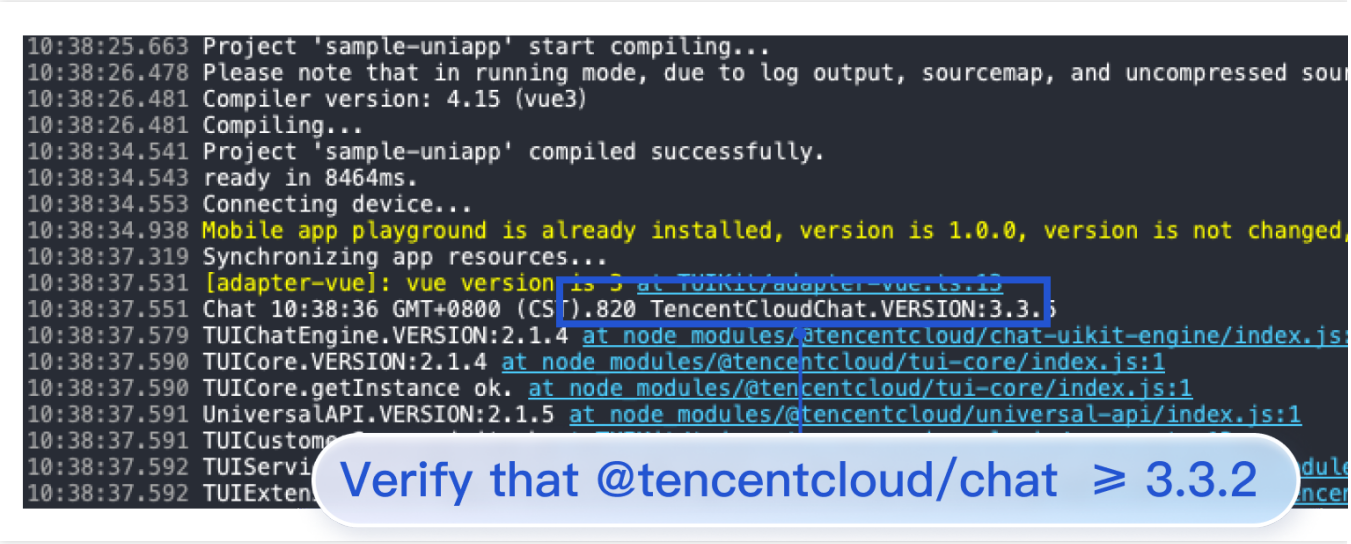
web

uniapp

Check the TencentCloudChat.VERSION in the browser console to confirm **@tencentcloud/chat ≥ 3.3.2** as shown:



Check the TencentCloudChat.VERSION in the HBuilder logs to confirm **@tencentcloud/chat ≥ 3.3.2** as shown:



2. Send a message, set related parameters for the custom ringtone in offlinePushInfo.

Note:

On Xiaomi phones with Android 8.0 and above, setting androidInfo.XiaoMiChannelID is mandatory, please refer to:Xiaomi Custom Ringtone.

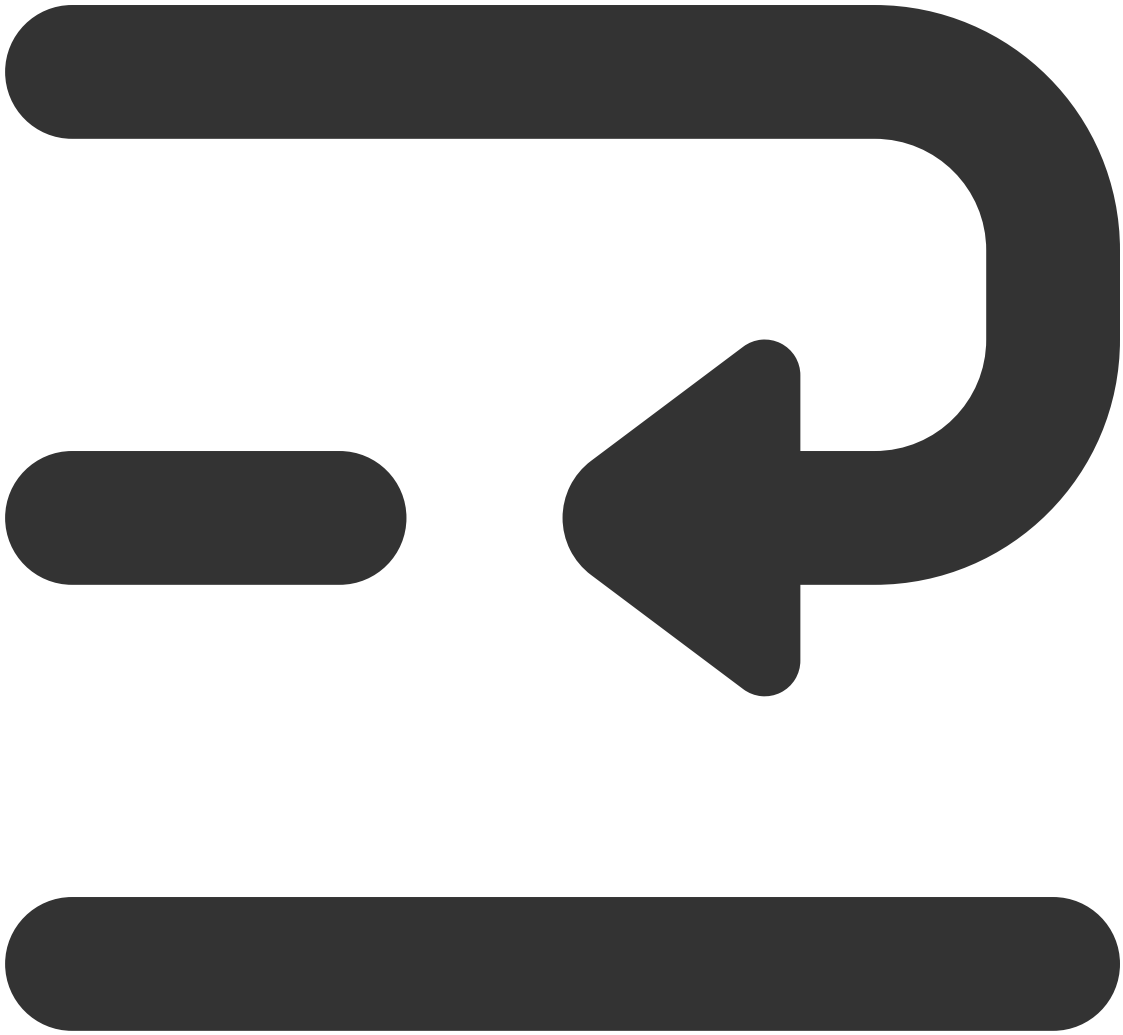
Google Phone FCM Push on Android 8.0 and above systems requires setting androidInfo.FCMChannelID for sound notifications.

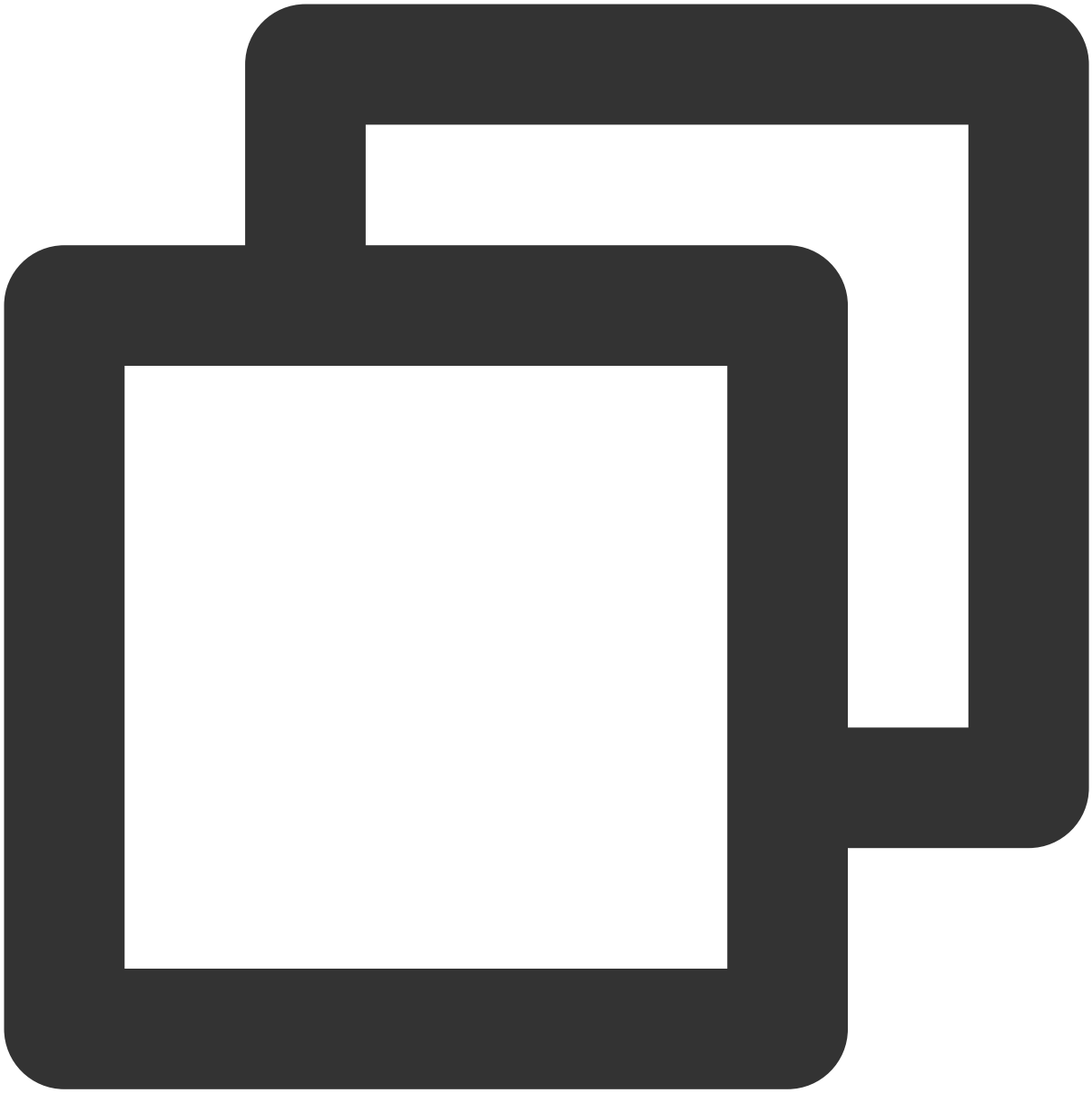
Integration (UI Included)

Integration Without UI

Note:

When apnsInfo.sound = TUIChatEngine.TYPES.IOS_OFFLINE_PUSH_NO_SOUND, it means no sound will be played upon receiving.
When apnsInfo.sound = TUIChatEngine.TYPES.IOS_OFFLINE_PUSH_DEFAULT_SOUND, it means the system sound will be played upon receiving.
When using TUIChatService in UIKit to send messages, set related offlinePushInfo parameters. For example, to send a regular text message, the code is as follows:





```
// Send Plain Text Message
let promise = TUIChatService.sendTextMessage(
{
   payload: {   text: 'Hello world!' }
},
{
  // If the recipient is offline, the message will be stored for roaming and an off
   offlinePushInfo: {
     androidInfo: { // Android Push Configuration
        sound: 'private_ring.mp3', // Custom ringtone for Android
        XiaoMiChannelID: '', // On Xiaomi Mobile Phone with Android 8.0 and above,
```

```
          FCMChannelID: '', // Google Phone FCM on Android 8.0 and above systems requ
        },
        apnsInfo: { // APNs Push Configuration
      // apnsInfo.sound = TUIChatEngine.TYPES.IOS_OFFLINE_PUSH_NO_SOUND, indicates n
      // apnsInfo.sound = TUIChatEngine.TYPES.IOS_OFFLINE_PUSH_DEFAULT_SOUND, indica
          sound: 'private_ring.mp3', // Custom ringtone for iOS
        }
    }
}
);
promise.catch((error) => {
    // Business side can handle errors by catching exceptions through promise.catch
});
```

Reference Documentation:

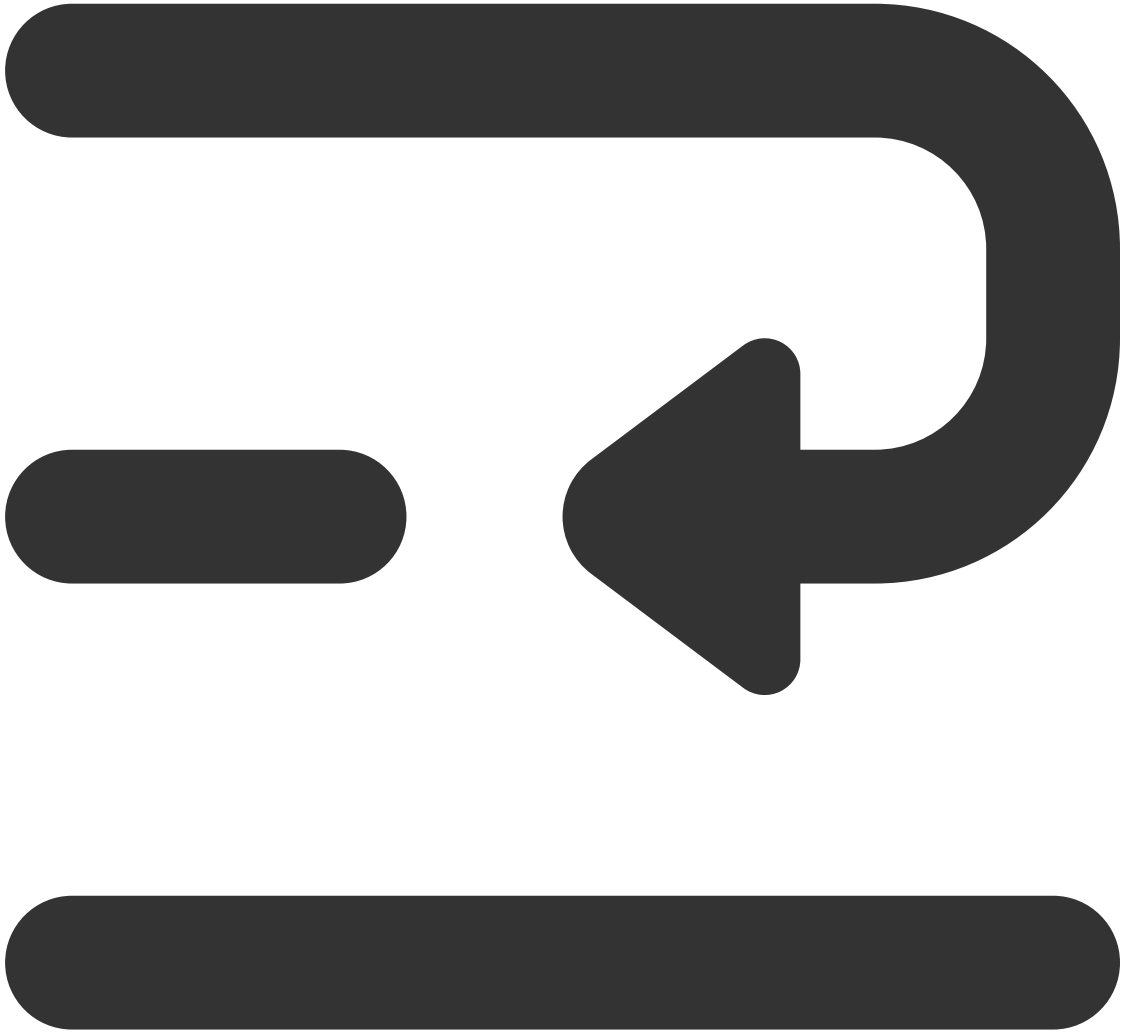
uikit - TUIChatService documentation related to sending messages

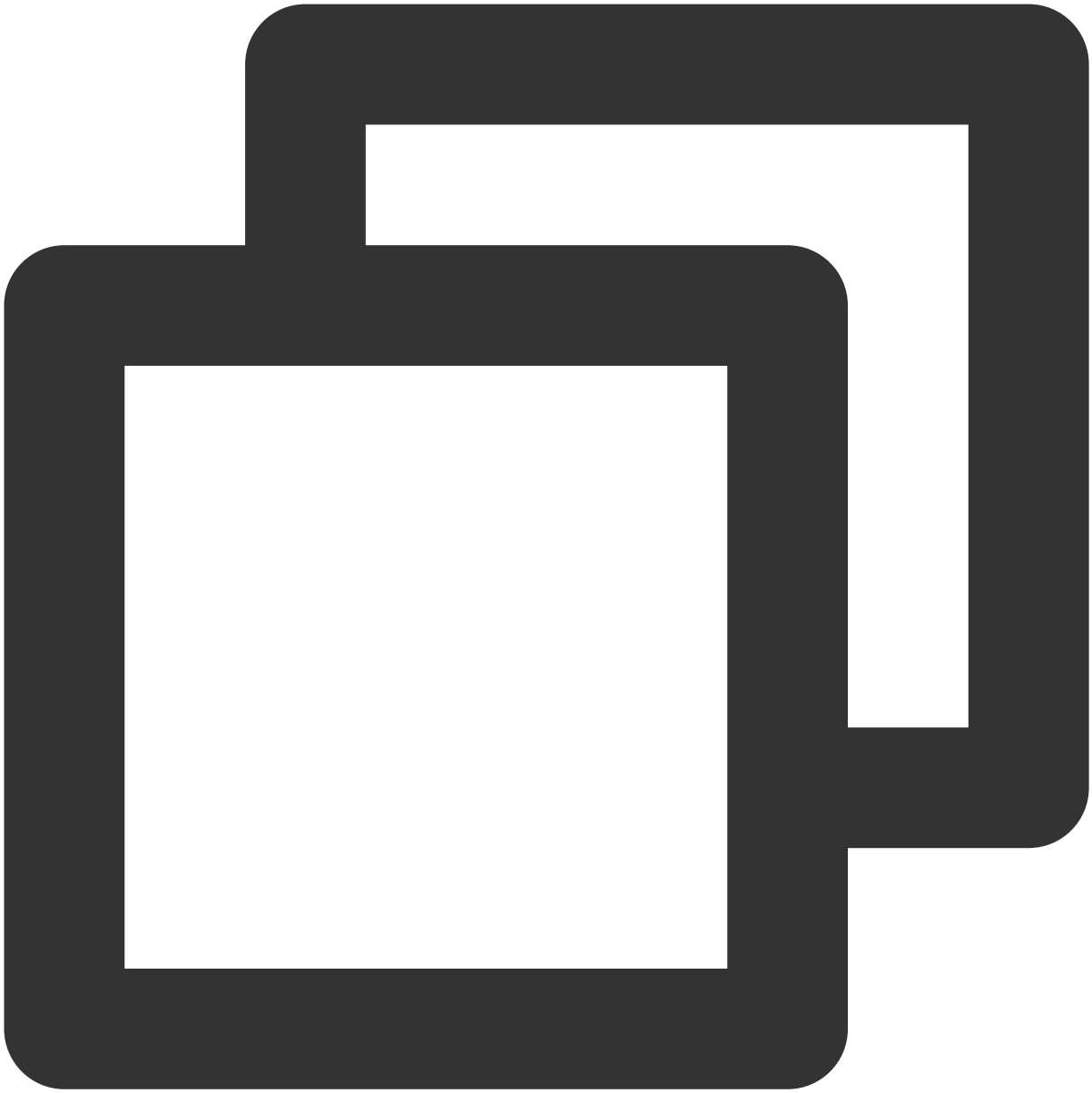
Note:

When apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_NO_SOUND, it means no sound will be played upon receiving.

When apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_DEFAULT_SOUND, it means the system sound will be played upon receiving.

When sending a message in chat, set the related fields for offlinePushInfo as follows:





```
// Message Sending Options
chat.sendMessage(message, {
  // If the recipient is offline, the message will be stored for roaming and an off
   offlinePushInfo: {
     androidInfo: { // Android Push Configuration
        sound: 'private_ring.mp3', // Custom ringtone for Android
        XiaoMiChannelID: '', // On Xiaomi Mobile Phone with Android 8.0 and above,
        FCMChannelID: '', // Google Phone FCM on Android 8.0 and above systems requ
      },
      apnsInfo: { // APNs Push Configuration
     // apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_NO_SOUND, indicate
```

```
      // apnsInfo.sound = TencentCloudChat.TYPES.IOS_OFFLINE_PUSH_DEFAULT_SOUND, ind
         sound: 'private_ring.mp3', // Custom ringtone for iOS
       }
    }
});
```

Reference document: Chat SDK - sendMessage documentation

Flutter

Note:

The interface supports Huawei, Xiaomi, FCM, and APNS.

Custom ringtone resource files for Android should be added to the project's raw directory; for iOS, link them into the Xcode project.

When calling sendMessage to send a message, set the fields offlinePushInfo, `iOSSound`, and `androidSound`.

For specific manufacturer configurations, please refer to the content of the Android and iOS modules below. The methods called are all available under the same name in the Flutter version of the IM SDK.

Customized Icon

Last updated：2024-06-13 10:21:45

Android

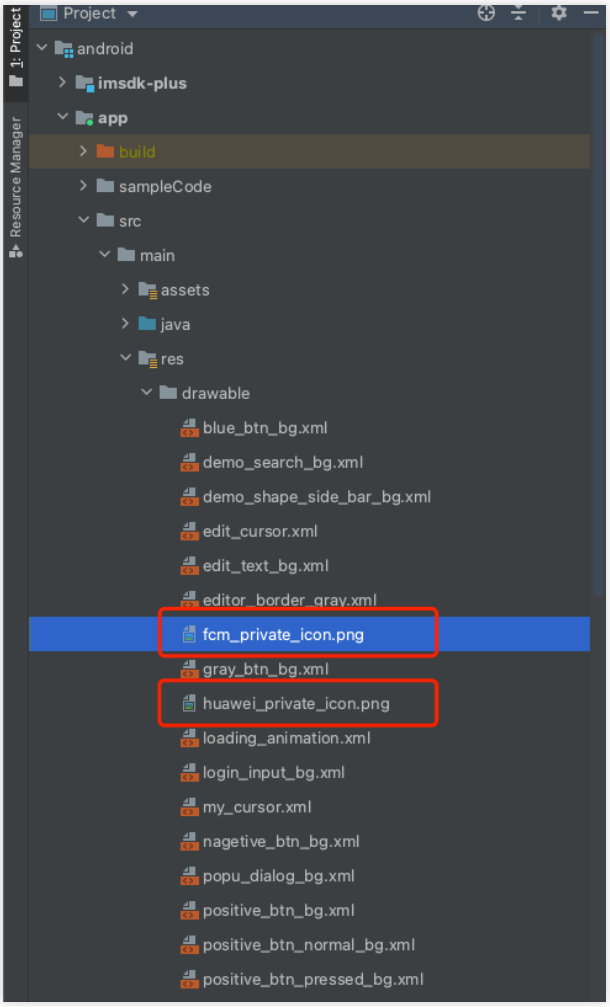
Supported Vendors

Huawei and Google FCM support Customized Icon; other manufacturers do not support Customized Icon, and by default, use the App Icon.

Configuration Method

Method 1

The component will automatically parse and update the Customized Icon related to Application Project Configuration; otherwise, it will use the default App Icon of the application.
Huawei:huawei_private_icon.png
FCM:fcm_private_icon.png

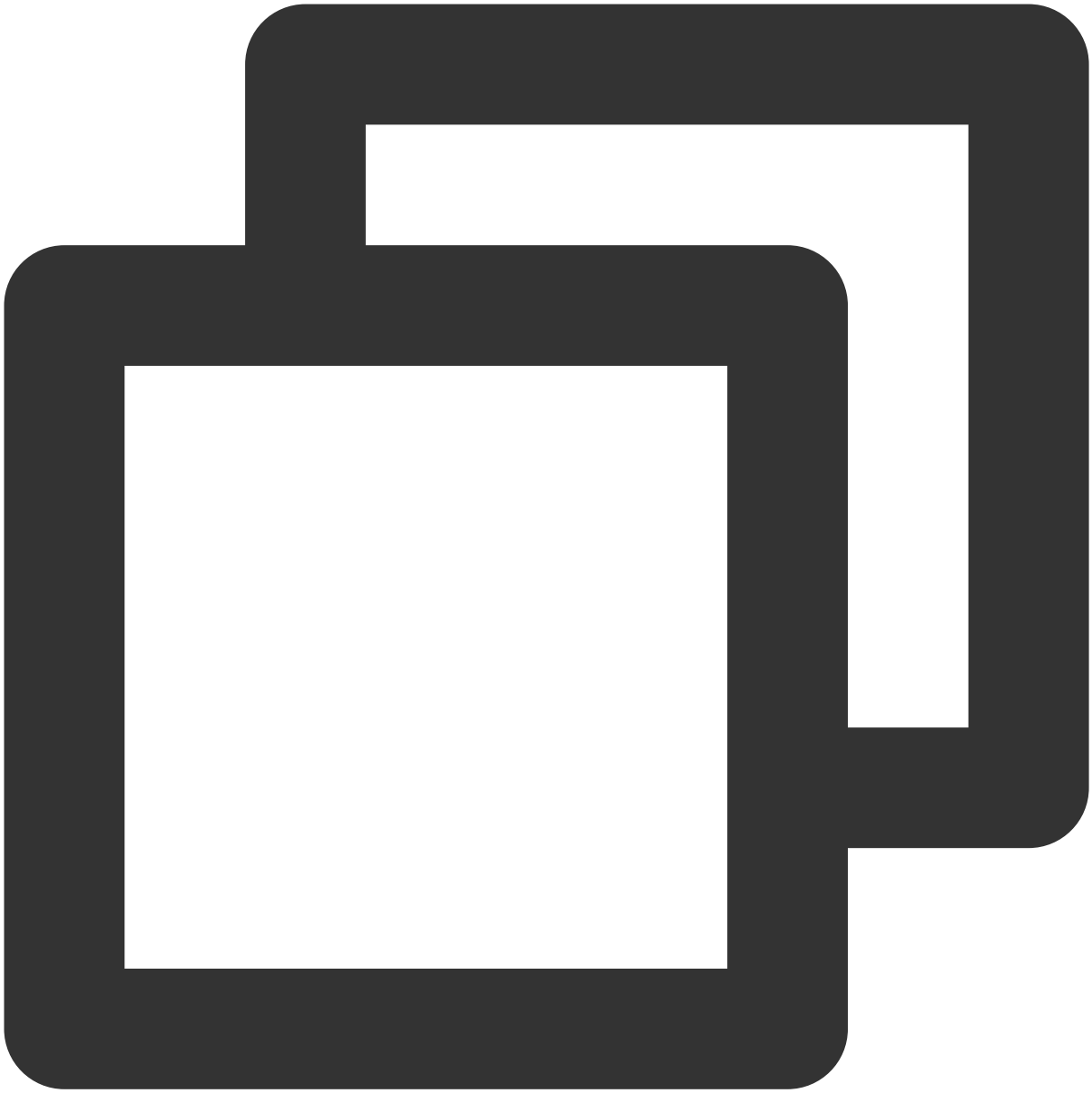


Method 2

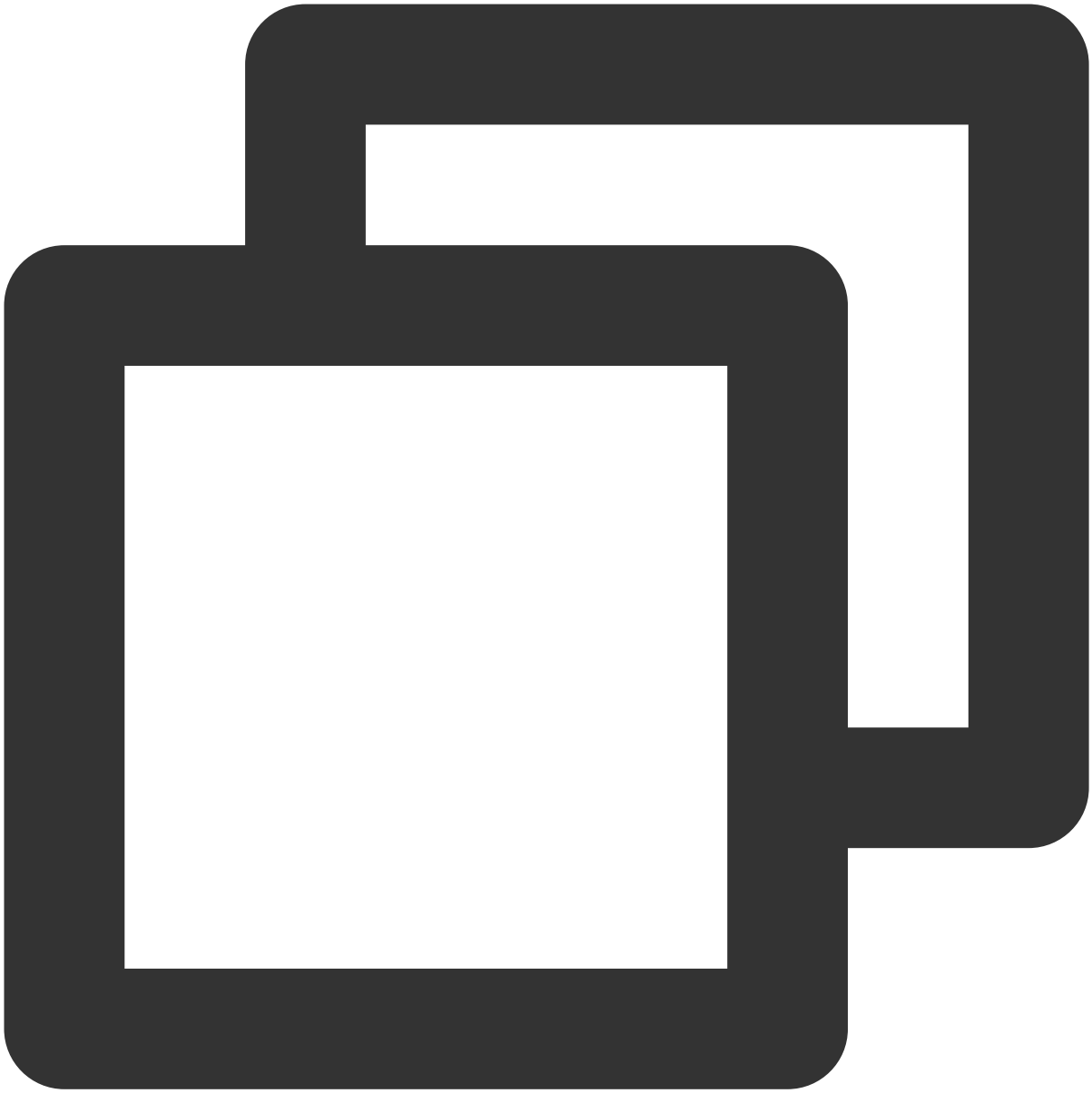
Effective when configured in the main project's Manifest File:

Huawei

Google FCM



```
<meta-data
    android:name="com.huawei.messaging.default_notification_icon"
    android:resource="@drawable/Icon Resource Name" />
```



```
<!-- [START fcm_default_icon] -->
<!-- Set custom default icon. This is used when no icon is set for incoming notific
     See README(https://goo.gl/l4GJaQ) for more. -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/Icon Resource Name" />

<!-- Set color used with incoming notification messages. This is used when no color
     notification message. See README(https://goo.gl/6BKBk7) for more. -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_color"
```

```
    android:resource="@android:color/white" />
<!-- [END fcm_default_icon] -->
```

Note:

FCM Icon Requirements:

Small icon must be a PNG image with Alpha transparency channel.

Background must be transparent.

Avoid leaving excessive padding around the icon.

It is recommended to use 46 x 46px uniformly. Smaller images will be fuzzy, while larger ones will be automatically scaled down by the system.

iOS

Custom definitions are not supported; the App Icon is used by default.

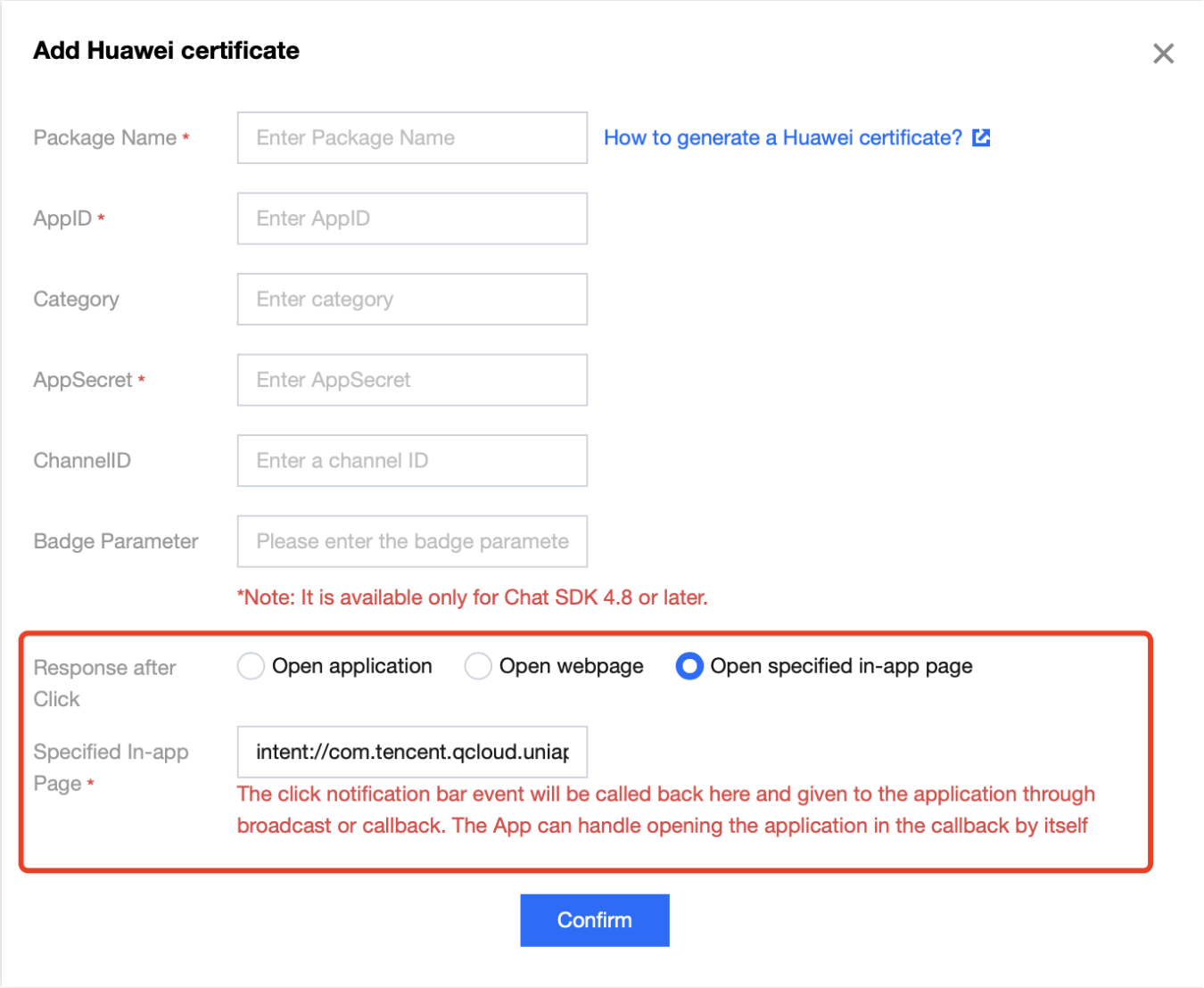
Custom Definition Click Redirect

Last updated：2024-06-13 10:21:45

Flutter

Step 1: Manufacturer Configuration

Refer to Manufacturer Configuration > Flutter, complete the manufacturer configuration. Please note, for subsequent actions after clicking, use the default configuration.



Step 2: Client Code Configuration

Refer to Client Code Configuration, complete the configuration.

Step 3: Handle Message Click Callback and Parse Parameters

Please define a function for receiving push message click callback events.

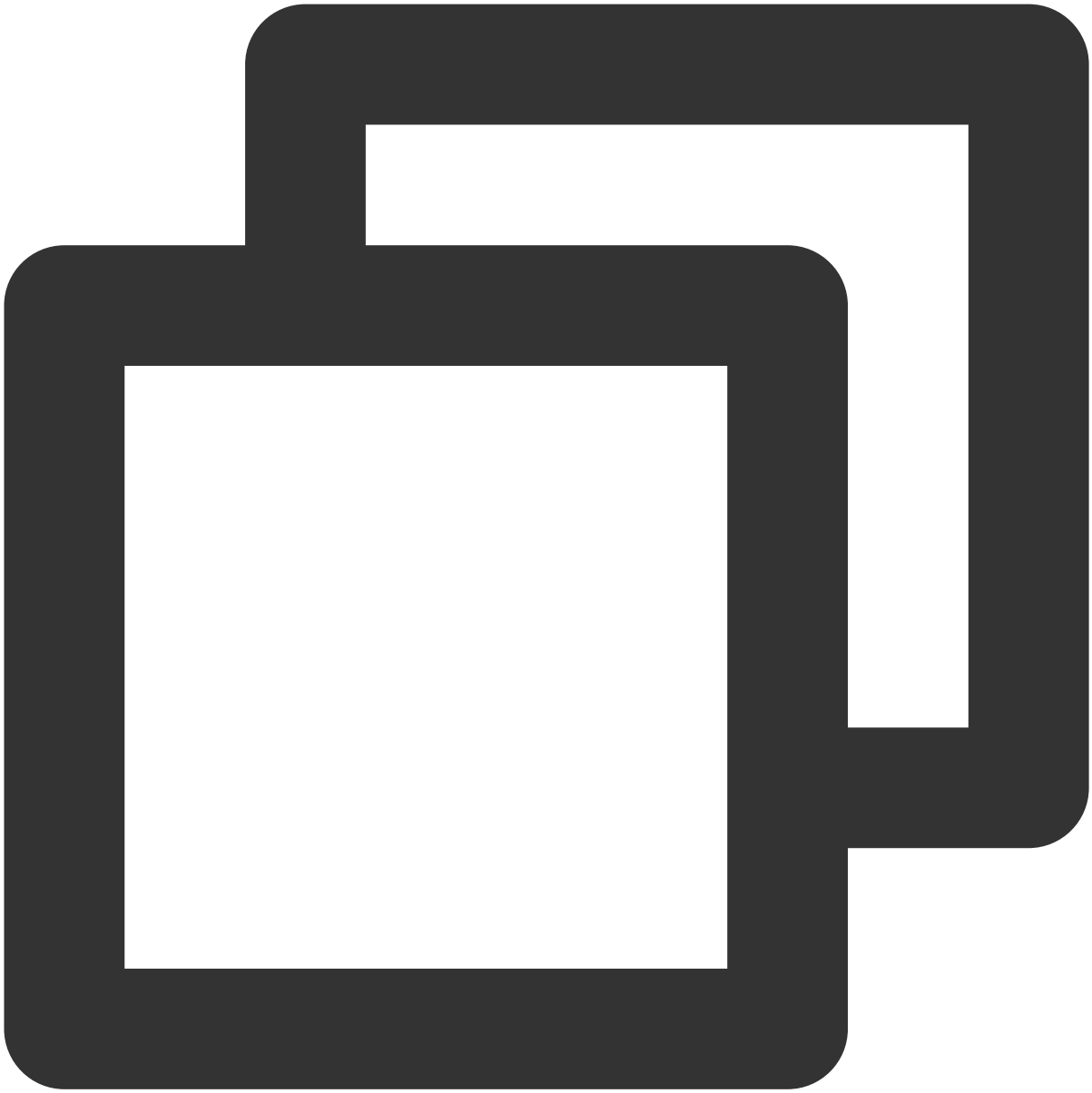
Define the function with the parameter format `{required String ext, String? userID, String? groupID}` .

Here, the ext field contains the full ext information carried by the message, specified by the sender. If not specified, there is a default value. You can use this to navigate to the corresponding page based on parsing this field.

The userID and groupID fields, for this plugin, attempt to automatically parse the ext Json String and obtain the individual chat partner userID and group chat groupID information carried within. If you haven't defined the ext field yourself, and the ext field is specified by default by the SDK or UIKit, you can use the default parsing here. If the parsing attempt fails, they will be null.

You can define a function to receive this callback and use it to navigate to the corresponding Session Page or your Business Page.

Example below:



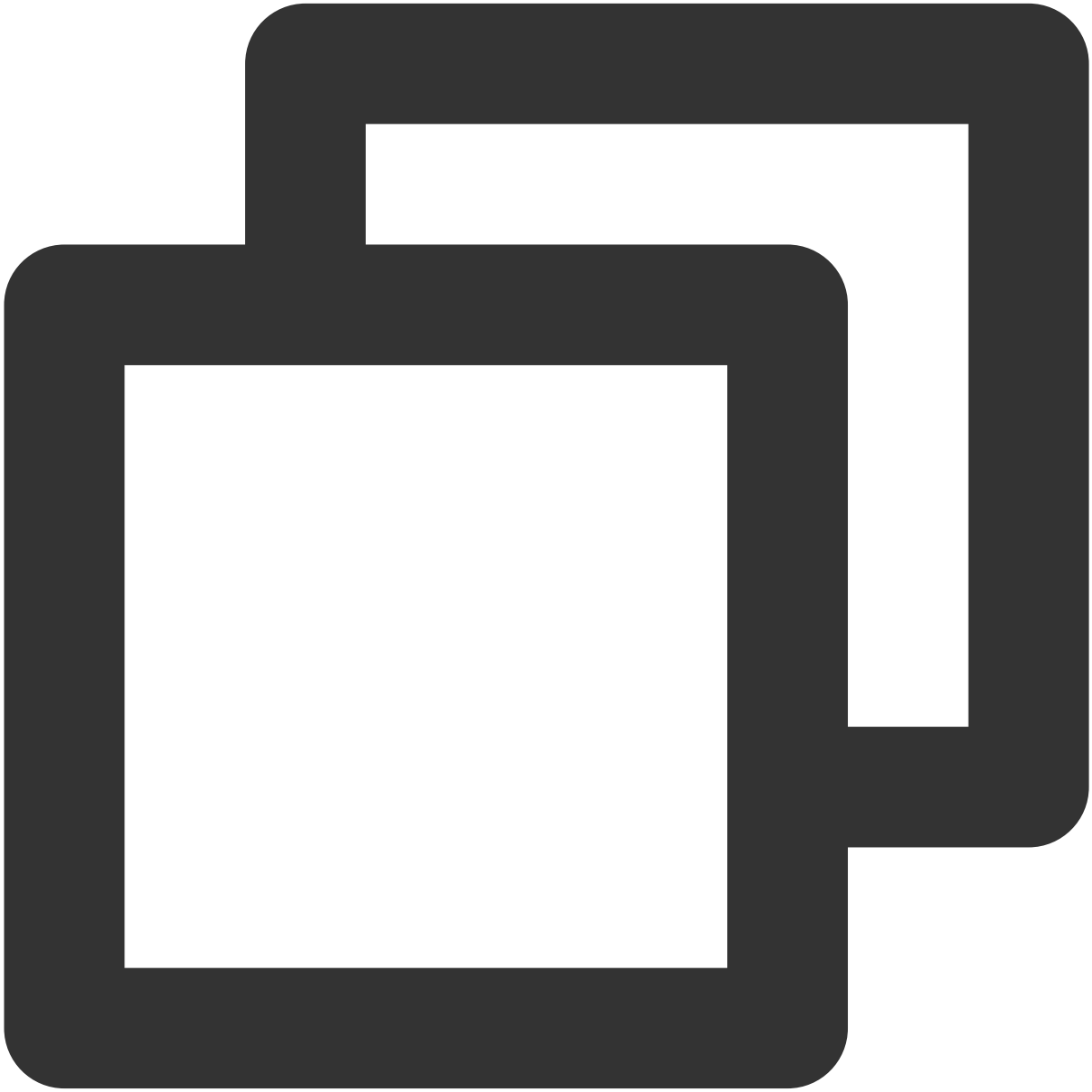
```
void _onNotificationClicked({required String ext, String? userID, String? groupID})
  print("_onNotificationClicked: $ext, userID: $userID, groupID: $groupID");
  if (userID != null || groupID != null) {
    // Navigate to the corresponding Message Page based on userID or groupID.
  } else {
    // Use your own parsing method based on the ext field to navigate to the corres
  }
}
```

Step 4: Mount the Listen for Callbacks

After completing the IM Login, and before using other plugins (such as CallKit), immediately register the Push Plugin.

Invoke the `TencentCloudChatPush().registerPush` method, pass in the Click Callback Function defined in the background.

Additionally, you can optionally pass in the `apnsCertificateID` for the iOS Push Certificate ID and `androidPushOEMConfig` for Android Push Vendor Configuration. These configurations have been specified in previous steps. If no changes are necessary, they need not be passed in again.

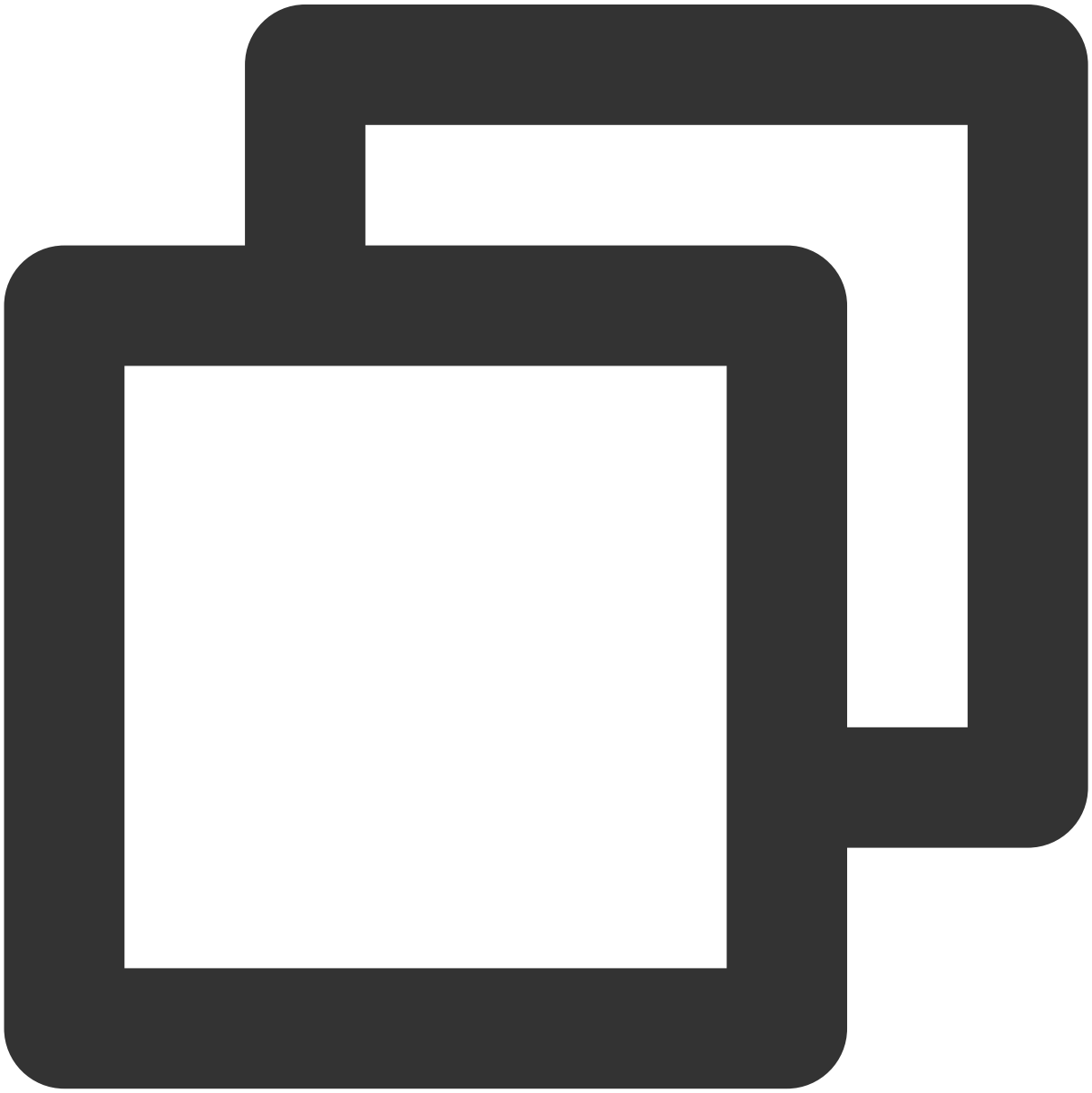


```
TencentCloudChatPush().registerPush(onNotificationClicked: _onNotificationClicked);
```

Note:

If your application needs to use the **push plugin for business message notifications**, and it does not immediately start and log in to the IM module after launching<3>, or you need to handle business navigation through message click callbacks before <5>logging in to the IM module</5>, it is recommended that you call the <7>TencentCloudChatPush().registerOnNotificationClickedEvent</7> method as soon as possible to manually mount the message click callback, in order to timely retrieve message parameters.

In such scenarios, you can execute this function before calling `TencentCloudChatPush().registerPush` and place it as early as possible in your code.

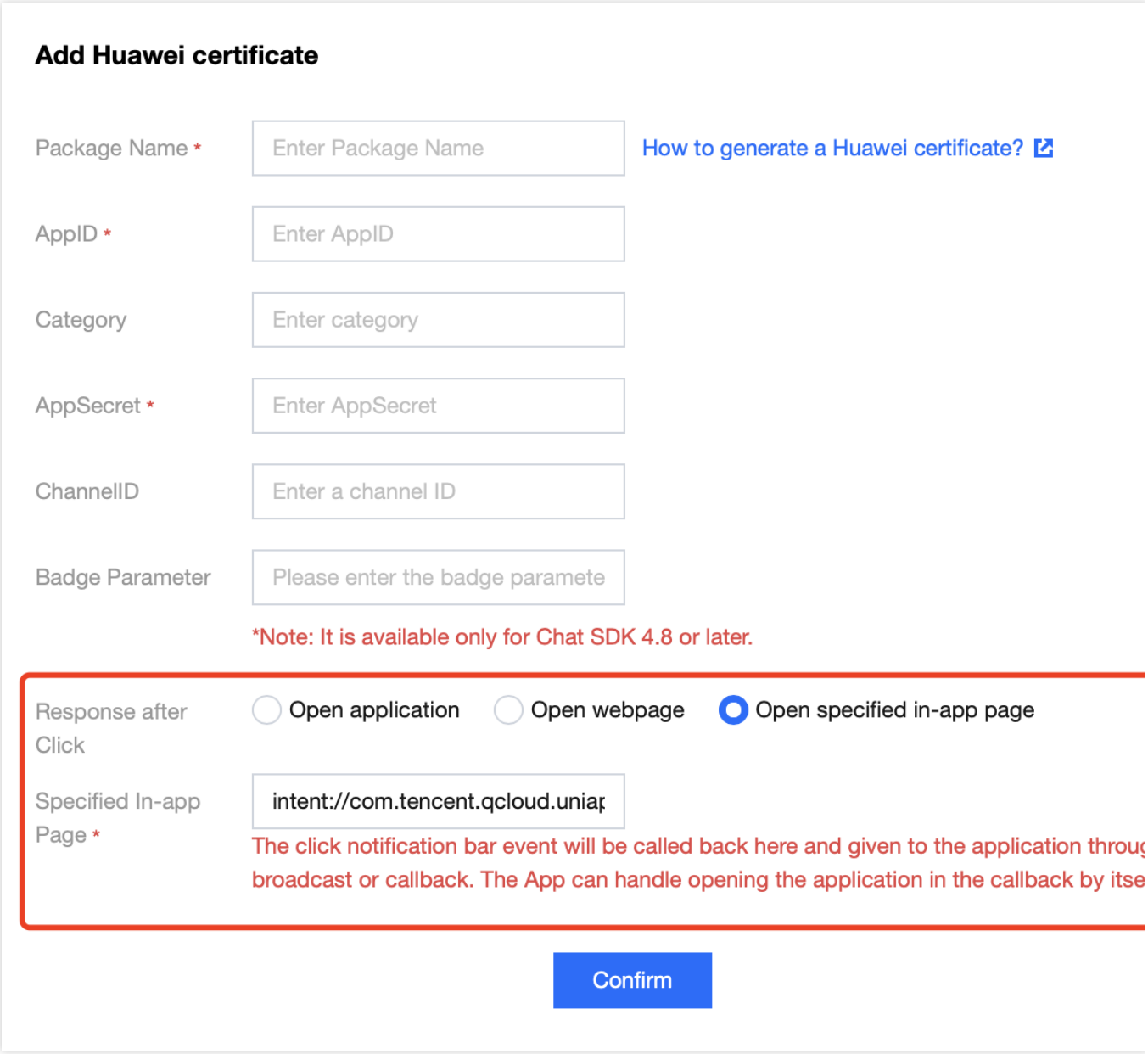


```
TencentCloudChatPush().registerOnNotificationClickedEvent(onNotificationClicked: _o
```

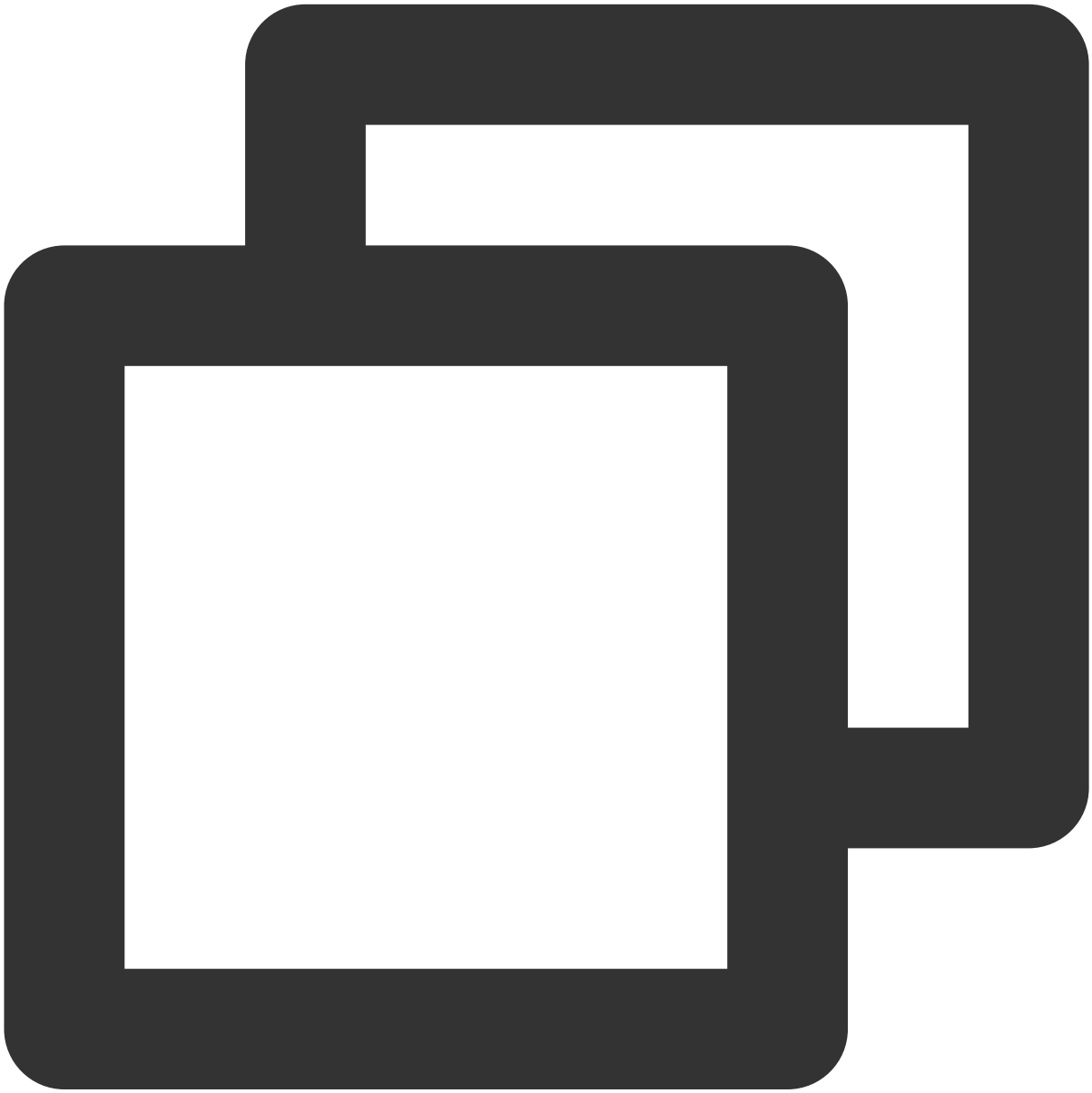
Android

After receiving an offline push, the notification bar will display the push message as shown. Clicking on the notification bar can custom open the application interface.

1. For console configuration, click on 'Subsequent Actions' and configure as follows, choose **Open the specified interface within the app**, and the plugin users will by default fill in the jump parameters.



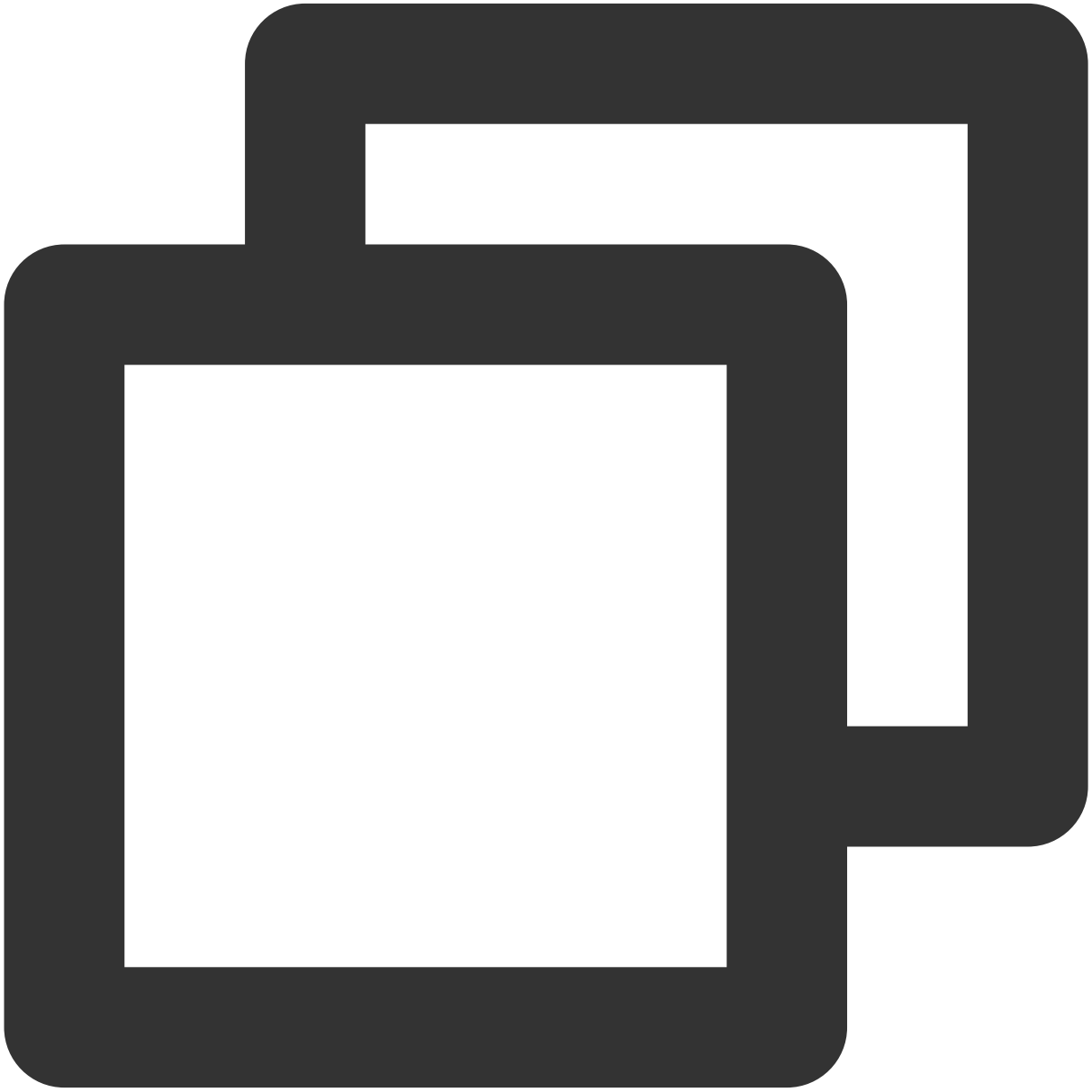
2. For registration and callback handling of click events, it's recommended to place the registration in the application's Application oncreate() function. The component will notify the application via callback or broadcast, and the application can configure its custom jump page in the callback.
The callback method is as follows:



```
TUICore.registerEvent(TUIConstants.TIMPush.EVENT_NOTIFY, TUIConstants.TIMPush.EVENT
        @Override
        public void onNotifyEvent(String key, String subKey, Map<String, Object> pa
            Log.d(TAG, "onNotifyEvent key = " + key + "subKey = " + subKey);
            if (TUIConstants.TIMPush.EVENT_NOTIFY.equals(key)) {
                if (TUIConstants.TIMPush.EVENT_NOTIFY_NOTIFICATION.equals(subKey))
```

```
                    if (param != null) {
                        String extString = (String)param.get(TUIConstants.TIMPush.N
                        TUIUtils.handleOfflinePush(extString, null);
                    }
                }
            }
        }
    });
```

The broadcast method is as follows:



```
// Dynamic Broadcast Registration
```

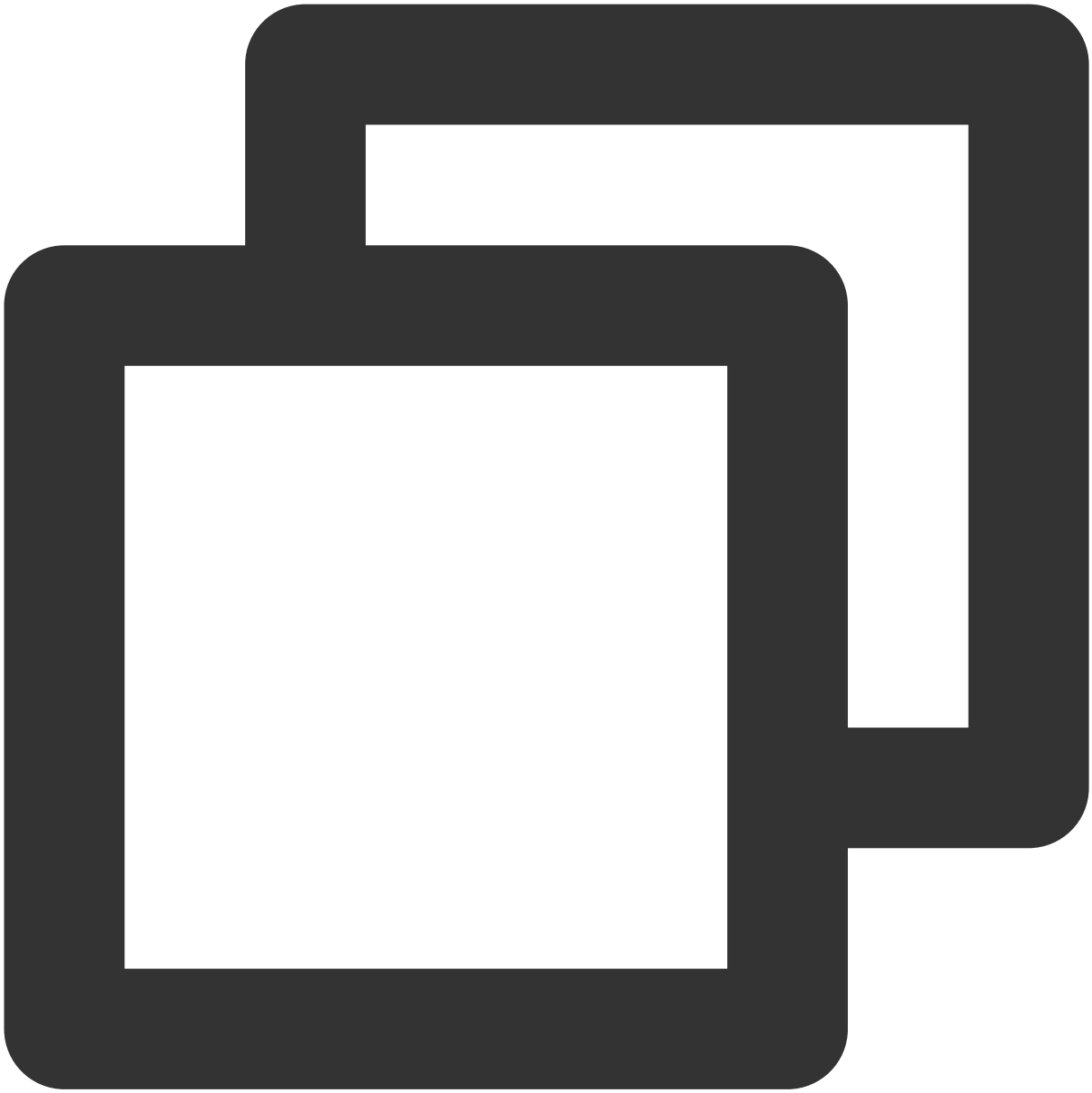
```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(TUIConstants.TIMPush.NOTIFICATION_BROADCAST_ACTION);
LocalBroadcastManager.getInstance(context).registerReceiver(localReceiver, intentFi

// Broadcast Receiver
public class OfflinePushLocalReceiver extends BroadcastReceiver {
    public static final String TAG = OfflinePushLocalReceiver.class.getSimpleName()

    @Override
    public void onReceive(Context context, Intent intent) {
        DemoLog.d(TAG, "BROADCAST_PUSH_RECEIVER intent = " + intent);
        if (intent != null) {
            String ext = intent.getStringExtra(TUIConstants.TIMPush.NOTIFICATION_EX
            TUIUtils.handleOfflinePush(ext, null);
        } else {
            DemoLog.e(TAG, "onReceive ext is null");
        }
    }
}
```

iOS

Please set the V2TIMOfflinePushInfo's `ext` field when calling sendMessage to send a message. When the user receives an offline push and starts the App, they can get the `ext` field in the `AppDelegate -> didReceiveRemoteNotification` system callback, and then jump to the specified UI interface according to the content of the `ext` field.

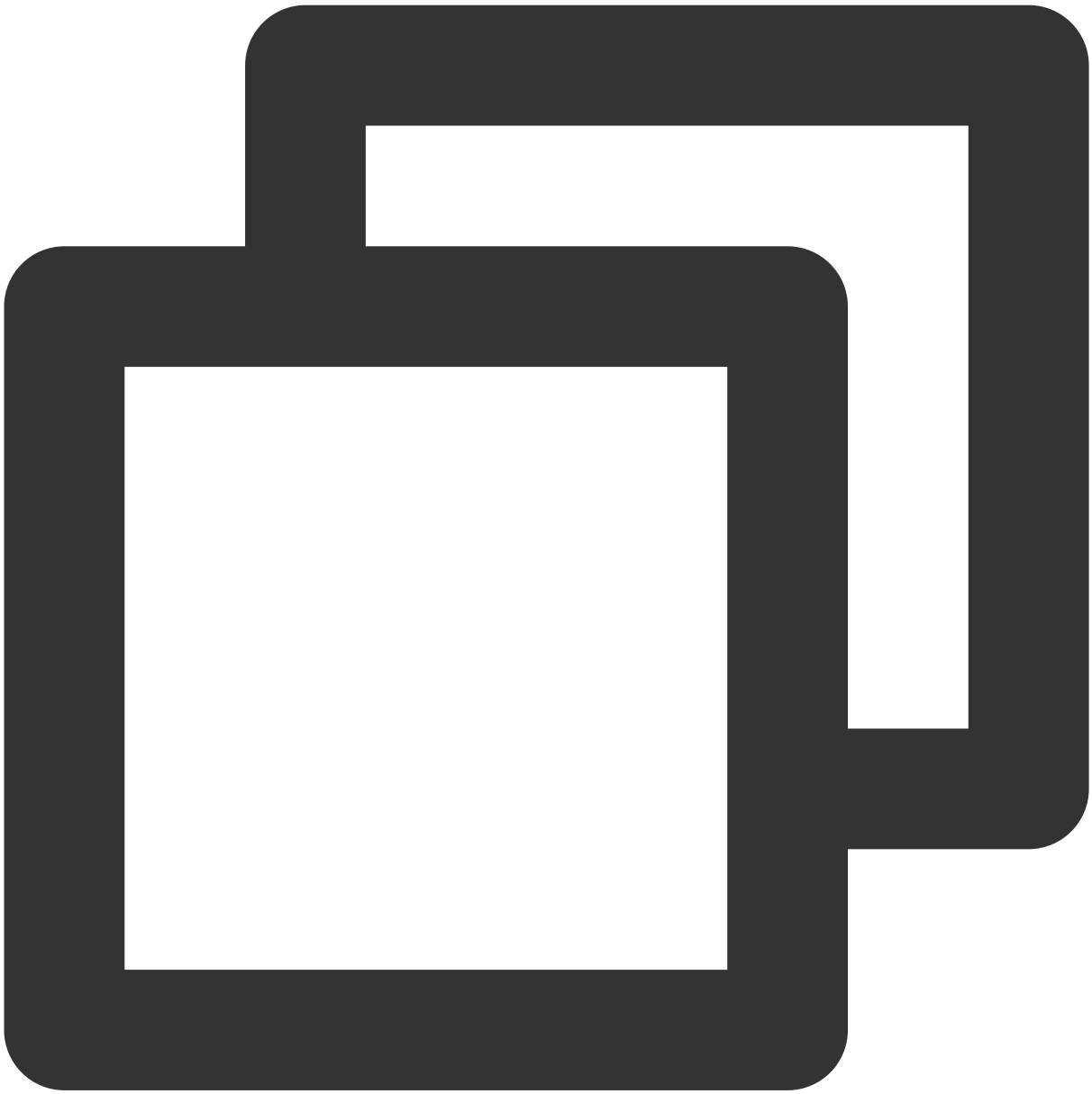
Taking the scenario of **Denny sending a message to Vinson** as an example.

The sender, Denny, needs to set the push extension field `ext` when sending the message:



```
// Denny sets offlinePushInfo and specifies the ext field when sending the message
V2TIMMessage *msg = [[V2TIMManager sharedInstance] createTextMessage:@"Text Message
V2TIMOfflinePushInfo *info = [[V2TIMOfflinePushInfo alloc] init];
info.ext = @"jump to denny";
[[V2TIMManager sharedInstance] sendMessage:msg receiver:@"vinson" groupID:nil prior
onlineUserOnly:NO offlinePushInfo:info progress:^(uint32_t progress) {
} succ:^{
} fail:^(int code, NSString *msg) {
}];
```

Recipient: Vinson's App may be offline, but can still receive APNS offline push. When Vinson clicks on the push message, the App will launch:



```
// After launching the APP, Vinson will receive the following callback
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDi
fetchCompletionHandler:(void (^)(UIBackgroundFetchResult result))completionHandler
    // Parsing the push extension field desc
    if ([userInfo[@"ext"] isEqualToString:@"jump to denny"]) {
        // Jump to the chat interface with Denny
    }
}
```

Push Message Categorization

Last updated：2024-06-13 10:21:45

Every vendor has a message categorization mechanism, and different types have different push strategies. If the push requirement is of the IM type and timely delivery is desired, it is necessary to set your application as the corresponding push type according to vendor rules. It will be categorized as a system message type or an important message type with high priority. Conversely, offline push will not be pushed to the device in a timely manner.

Note:

Only IM type messages need to be configured as system message types or important message types for timely delivery. Some marketing or advertising type pushes, which do not require immediate delivery, can arrive at the device within a certain period, and do not need to be configured as a system message type.

Vendors have restrictions on the daily push quantity and push frequency for applications, which can be viewed in the vendor console regarding the daily limit on push quantity and restrictions.

Do not arbitrarily configure message types. Configuration that does not meet the standard may result in the vendor freezing your account.

Huawei

Starting from EMUI 10.0, Huawei Push intelligently categorizes notification messages into two levels: **Service and Communication** and **Information Marketing**. Versions earlier than EMUI 10.0 did not categorize notifications, having only one level where all messages were displayed through the **Default Notifications** channel, equivalent to Service and Communication on EMUI 10.0. The daily push quantity for Information Marketing messages will be subject to an upper limit management based on the application type from January 5, 2023, while Service and Communication messages will not be limited in daily push quantity.

Customized Self-Categorization Push Method

Apply for Self-Categorization Benefits.

Push messages carry the category field, for details see setAndroidHuaWeiCategory. For console settings, see the Category field in Certificate Editing. Setting either one is sufficient.

For more information, see Message Categorization Standards or Push Quantity Management Rules.

HONOR

Honor phone push is related to the system version.

Currently, the Honor channel only supports Honor devices in China with Magic UI 4.0 or above and those overseas with Magic UI 4.2 or above.

Honor devices below the versions mentioned can use the Huawei Manufacturer for push access.
For more information, see Product Description.

vivo

Push messages are classified into system messages and operational messages, with different push effectiveness and policies. System messages also undergo a secondary correction through the vendor's intelligent classification; if it's incorrectly identified as a non-system message, it will automatically be corrected as an operational message. False judgments can be corrected by email application feedback. Moreover, message pushing is also subject to a daily limit, determined by the app's subscription statistics with the manufacturer.

Customized Self-Categorization Push Method

Push messages carry the category field. For details, see setAndroidVIVOCategory. For console settings, refer to Certificate Editing Category field. Setting one of the two is sufficient.
For more details, see Push Message Classification Description or Push Message Restriction Description.

OPPO

Push messages are classified into private messages and public messages with different push effectiveness and policies. Private messages target users who pay a certain level of attention and wish to receive information promptly. Application for private message channel benefits requires an email application. The number of pushes via the public trust channel is limited.

Customized Self-Categorization Push Method

Create Private Message Channel.
Push messages carry the channel ID field. For details, see setAndroidOPPOChannelID. For console settings, see Certificate Editing ChannelID field. Setting one of the two is sufficient.
For specific details, see Message Classification Description or Push Service Restriction Explanation.

Mi

Push messages are divided into "Private Message" and "Public Trust Message" categories, with the default channel being Public Trust Message. The daily push quantity of Public Trust Messages will be subject to upper limit management. Public Trust Messages are suitable for pushing Hot News, New Product Promotions, Platform Announcements, Community Topics, Prize-Winning Activities, etc., mostly content of universal interest to users. Private Messages are suitable for pushing Chat Messages, Personal Order Changes, Courier Notifications,

Transaction Reminders, IoT System Notifications, etc., relating to private notifications, with no restrictions on the number of notification messages pushed. Implementation of Message Classification Management requires channel application and access in the Manufacturer Console.

Custom Method for Self-Classified Push:

Channel Application and Access.
Push messages carry the channel ID field. For details, see setAndroidXiaoMiChannelID. For console settings, see Certificate Editing ChannelID field. Setting one of the two is sufficient.
For details, see Push Message Classification New Rules or Push Message Restriction Description.

Meizu

The number of push messages is limited.
For details, see Push Access Guide.

FCM

The frequency of push uplink messages is limited.
For details, see Message Frequency Limit.
Note:
The setting of push message Channel ID and the category field has two methods: API interface and console certificate settings, each with its scope of effect, and the API setting has a higher priority than the console setting.

Release Notes

Last updated：2024-08-01 12:04:20

TIMPush 8.0.6897 @2024.07.15

Added intelligent detection available channel strategy.

Added push registration timeout protection mechanism.

Resolved small icon settings issue.

Fixed the issue where configuring the jump option to the homepage couldn't launch the application.

Optimized model recogni

TIMPush 7.9.5668 @2024.04.09

Added OfflinePushExtInfo to support Transparent Push Feature.

Upgraded Vivo push package version.

Resolved occasional crash issues on OPPO and Vivo models.

Fixed occasional mismatches of FCM Data mode for multiple push notifications and transparent data.

Optimized WorkManager compilation issues.

UniApp supports Reach Reporting, supports Process Termination Callback for click events.

TIMPush - Android 7.8.5484 @2024.02.02

Released TIMPush-UniApp.

FCM PUSH supports Pass-through Message.

Optimization of Push Registration and Reporting Logic.

TIMPush - iOS 7.8.5483 @2024.02.02

iOS support for Minimum version updated to 10.0 and above.

TIMPush 7.7.5283 @2023.12.28

Add custom Definition push configuration file feature.

Upgrade Honor Push Package.

Optimize Vivo placeholder configuration.

Fix issue where some functions did not callback after iOS hosted UNUserNotificationCenterDelegate.

TIMPush 7.7.5282 @2023.12.18

Upgrade Xiaomi and Meizu Push Packages.

Optimize Context occasionally empty issue.

Optimize registerPush manual invocation of API logic.

TIMPush 7.6.5011 @2023.11.13

Release Push Plugin.

FAQS

Last updated：2024-06-13 10:21:45

Android

How do I troubleshoot if I cannot receive offline push messages?

Special Case Investigation

OPPO devices

There are several common reasons why OPPO devices do not receive push notifications:
According to the requirements of the official OPPO Push website, ChannelID must be configured on OPPO devices running Android 8.0 or above, otherwise, push notifications cannot be displayed. For the method of configuration, refer to setAndroidOPPOChannelID.
The notification bar display feature is disabled by default for applications installed on the OPPO device. If this is the case, check the switch status.

Google FCM

If push notifications are not received, verify whether the certificates have been correctly uploaded to the IM console. For the troubleshooting path, see the document Configuring in the IM console > Google FCM, and check against the diagram to see if it has been added correctly.

Xiaomi and vivo

Xiaomi and vivo: The app needs to be listed in the App Marketplace before it can be delivered through the Manufacturer Channel. It is usually indicated that the app is in the blocklist, prohibiting message sending. Or, the App has closed the push channel.

Sending messages as Custom Definition Messages

The offline push of Custom Definition Messages is different from ordinary messages. We cannot parse the content of Custom Definition Messages, and the push content cannot be determined. Therefore, by default, they are not pushed offline. If you have offline push requirements, you need to set the desc field of offlinePushInfo during sendMessage, and the desc information will be displayed by default during push.

Effects of Device Notification Bar Settings

The immediate effect of offline push is the Notification Bar Prompt, so like other notifications, it's affected by the device notification settings. Taking Huawei as an example:
Phone Settings > **Notifications** > **Lock Screen Notifications** > **Hide or Do Not Display Notifications**, will affect the display of offline push notifications when the screen is locked.

Phone Settings > **Notifications** > **More Notification Settings** > **Status Bar Notification Icons**, will affect the display of offline push notification icons in the status bar.
Phone Settings > **Notifications** > **App Notification Management** > **Allow Notifications**, toggling on or off will directly affect the display of offline push notifications.
Phone Settings > **Notifications** > **App Notification Management** > **Notification Ringtone** and **Phone Settings** > **Notifications** > **App Notification Management** > **Silent Notifications**, will affect the sound of offline push notification.

Message Type

Vendor push services all have message categorization mechanisms, and different types will have different push strategies. For details, please see Push Message Categorization.

Vendor Features

Offline push depends on the vendor's capabilities, and some simple characters may be filtered by the vendor and not be pushed through transparently. It is recommended to use meaningful content for pushing.

Self-service Push Troubleshooting Tool

Whether device push plugin integration is working properly

Test whether messages can be properly pushed offline by using the Offline Testing Tool in the IM Console.

Push Link Troubleshooting

Use the Troubleshooting Tool to view the details of the full push link.

How to troubleshoot if the jump interface is unsuccessful?

Click to perform a Configuration Check

See Custom Click-through Redirect for details, check:
Configure the subsequent actions after clicking in the console as follows, select **Open the specified interface within the app**, plugin users will by default fill in the jump parameters, check if the parameters have been modified.
Registration and callback processing for click events. It is recommended to register during the application Application's oncreate() function, which needs to be placed early in the application lifecycle.
Check if the click callback correctly handles the redirect logic.

Device System Permission Restrictions

If the application process does not exist and clicking the notification bar to jump to the application interface, it is necessary to pull the application from the background to the foreground. Some manufacturers' systems will check whether the App has enabled **background self-start** or **floating window** permissions. If not enabled, the system side will intercept and handle it, leading to failure.

Different manufacturers, and even different Android versions from the same manufacturer, offer different permissions and permission names for applications. Through testing, only the background pop-up interface permission needs to be enabled for Mi 6, while Redmi requires both the background pop-up interface and floating window display permissions, necessitating different treatments for different manufacturers.

Push Link Troubleshooting

Use the Troubleshooting Tool to view the details of the full push link.

Others

Huawei

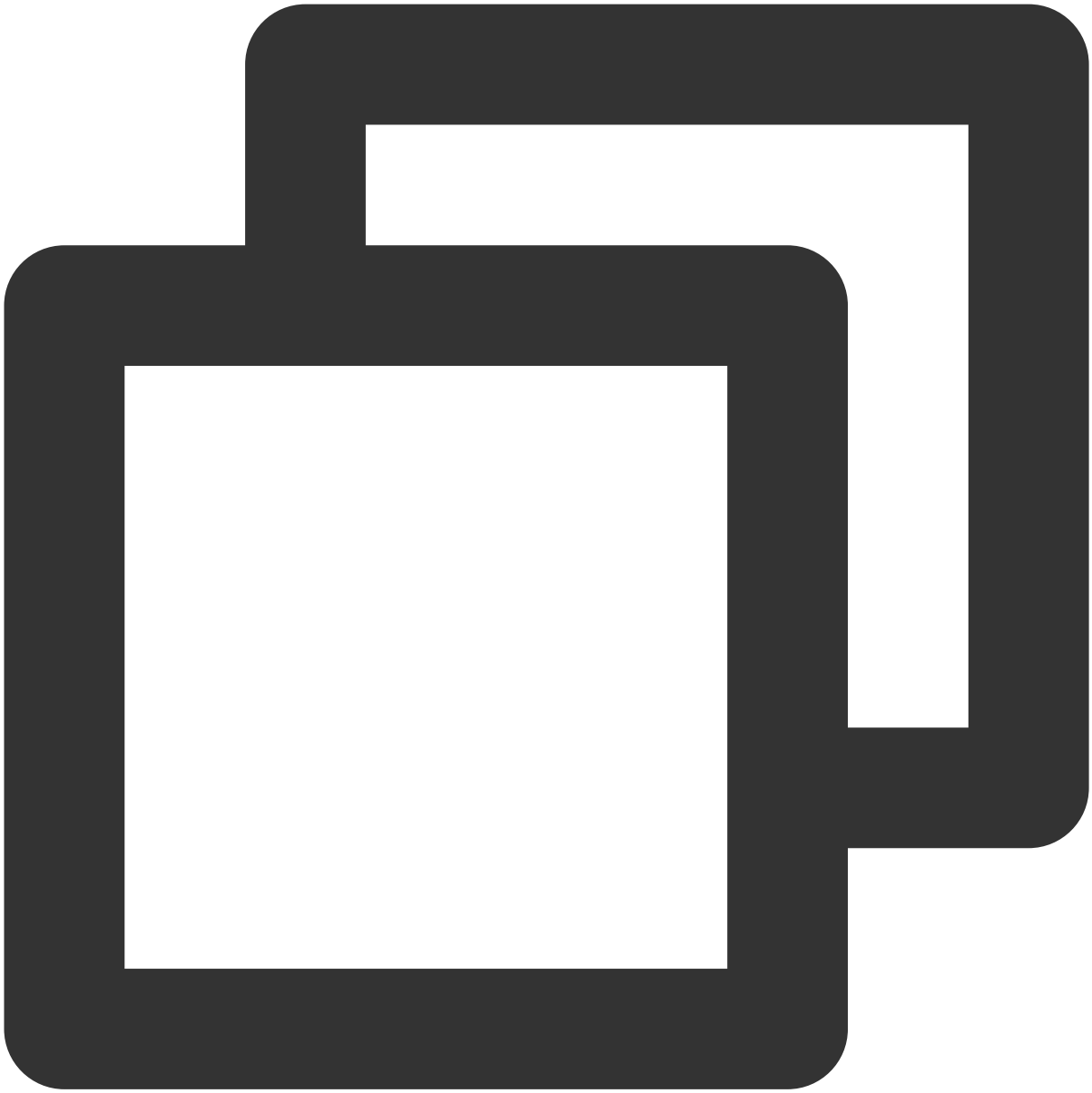
Huawei Push Certificate ID <= 11344, using Huawei Push v2 Version Interface, does not support reach and click acknowledgment. If you need statistical data feature, please regenerate and update the Certificate ID. AndroidInfo.ExtAsHuaweiIntentParam, passing "1" means treating the pass-through content Ext as an Intent parameter, "0" means treating the pass-through content Ext as an Action parameter. The restapi using "1" currently does not support click event statistics.

Statistical Data Feature

Only the push data details of the last device logged in are recorded; multi-platform login scenarios are not supported.

The Debug version of the App feature works normally, but anomalies occur with the Release version of the App feature

This issue is very likely caused by obfuscation. You can add the following obfuscation rules:



```
-keep class com.tencent.qcloud.** { *; }
-keep class com.tencent.timpush.** { *; }
```

Resolving conflicts between integrating TIMPush and other competitors

The reason is that the application itself or the third-party push client it integrates or depends on conflicts with the third-party client in TIMPush. It is necessary to keep only one in use. For specific methods, please refer to: TIMPush Integration Conflict Resolution.

iOS

Why doesn't offline push work for common messages?

First, check that the app runtime environment is the same as the certificate environment. Otherwise, offline push messages will not be received.
Next, check if the app and the certificate environment are set to production. If they are in a development environment, applying for a `deviceToken` from Apple might fail. This issue has not been found in the production environment, so please switch to production for testing.

Why doesn't offline push work for custom messages?

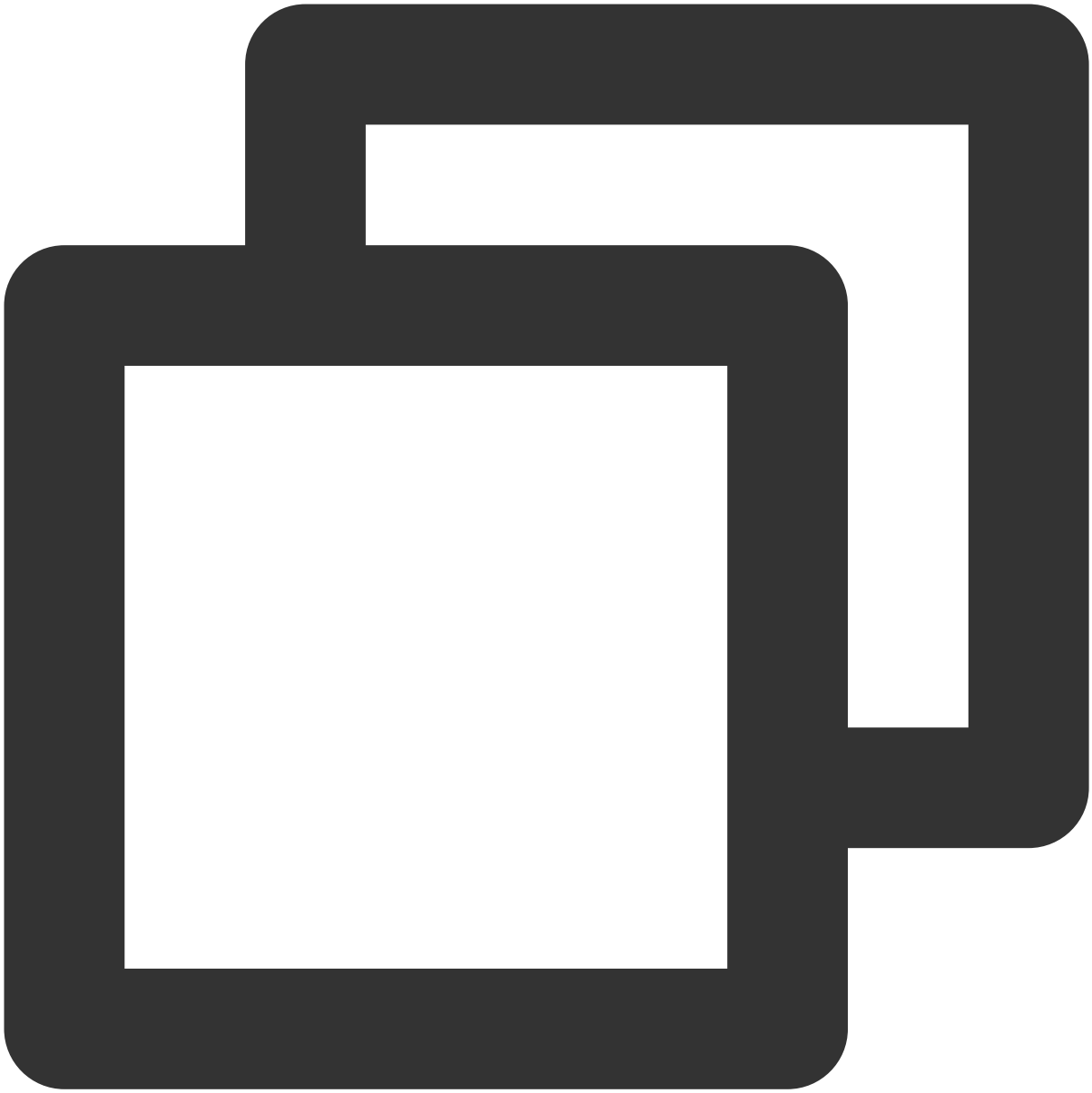
The offline push for custom messages differs from that for regular messages. We cannot parse the content of custom messages, making it impossible to determine the push content. Therefore, custom messages are not pushed by default. If you require push, you need to set the offlinePushInfo `desc` field during sendMessage. The `desc` information will by default be displayed during push.

How do I disable the receiving of offline push messages?

To disable receiving offline push messages, you can set the `config` parameter of the setAPNS API to `nil` . This feature has been supported starting from version 5.6.1200.

Push notifications are not received, and the server reports a 'bad devicetoken'.

Apple's deviceToken depends on the current build environment. If the certificate ID and token used when logging into IMSDK and uploading deviceToken to Tencent Cloud do not match, an error will occur.
If you are using the Release environment compilation, the `-application:didRegisterForRemoteNotificationsWithDeviceToken:` callback returns the production environment's token. At this time, set the `businessID` to the production environment's Certificate ID.
If you are using the Debug environment compilation, the `-application:didRegisterForRemoteNotificationsWithDeviceToken:` callback returns the development environment's token. At this time, set the `businessID` to the development environment's Certificate ID.



```
V2TIMAPNSConfig *confg = [[V2TIMAPNSConfig alloc] init];
/* You need to register a developer certificate with Apple, download and generate t
// Push certificate ID
confg.businessID = sdkBusiId;
confg.token = self.deviceToken;
[[V2TIMManager sharedInstance] setAPNS:confg succ:^{
   NSLog(@"%s, succ, %@", __func__, supportTPNS ? @"TPNS": @"APNS");
} fail:^(int code, NSString *msg) {
   NSLog(@"%s, fail, %d, %@", __func__, code, msg);
}];
```

Under the iOS development environment, is the registration occasionally not returning a deviceToken or indicating a failure in requesting the token from APNs?

This problem is caused by instability of APNs. You can fix it in the following ways:

1. Insert a SIM card into the phone and use the 4G network.
2. Uninstall and reinstall the application, restart the application, or shut down and restart the phone.
3. Use a package for the production environment.
4. Use another iPhone.

iOS did not receive the delivery receipt

1. To report push delivery data, enable the console switch to support iOS 10's Extension feature. For more details, please refer to Statistics Push Arrival Rate.
2. To ensure reported data is not lost in abnormal conditions, iOS push data reporting will only occur upon the next log in, which may result in a delay. Please try to log in again.

iOS Certificate Expiration

In the IM console, click editing on the corresponding certificate and update the p12 certificate. It's important to note: do not delete the certificate and recreate it, as recreating the certificate will change the certificate ID, and an update will be required to support it.

iOS Statistical Reporting Feature

To report push reach data, this switch must be enabled to support iOS 10's Extension feature. For more details, please refer to Statistics Push Arrival Rate.

Others

Cannot Receive Push Notifications, Plugin Subscription Expired

Once the push plugin's **trial or purchase expires, the push service will automatically stop (including standard message offline push, broadcast/Tag push, etc.)**. To avoid affecting the normal use of your business, please purchase/renew in advance.

Exchange and Feedback

Welcome to join the Tencent Cloud Chat Community for technical communication and feedback.