Ant Technology

Mobile Gateway Service
User Guide

Document Version: 20250303

蚂蚁集团
ANT GROUP

Legal disclaimer

Document conventions

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

Table of Contents

1.Change history

Document version	Revisions
V20211105	• Added descriptions about the service circuit breaking and dynamic routing toMobile Gateway Service overview. • In the API groups topic, updated the procedure for creating a Duboo API group, and added multi-IDC and registry authentication information. • In the Configure APIs topic, added descriptions about configuring the circuit breaking mechanism and updated the parameter description. • In the Gateway management overview, added descriptions about the circuit breaking mechanism. • Added the Routing rule topic to describe how to configure and manage routing rules.
V20210630	• Updated the Maven dependency version in Service definition and development. • Added a code sample about using SM2/SM3 for signature signing inVerify the backend signature.

2.About Mobile Gateway Service

Mobile Gateway Service (MGS) is a component provided by mPaaS that connects the mobile client and server. This component simplifies the data protocol and communication protocol between the mobile terminal and the server, and can significantly improve development and network communication efficiency.

Features

The gateway serves as a bridge between the client and server. The client accesses the service API in the backend through the gateway. The gateway provides the following functions:

- Automatically generates the RPC call code for the client regardless of network communication, protocols, and data formats.
- Automatically reverse the data returned from the server to generate Objective-C objects, without extra coding.
- Supports data compression, caching, etc.
- Supports unified exception handling, such as pop-up display and toasts.
- Supports RPC interceptors to achieve customized requests and processing.
- Uses the unified security encryption mechanism and anti-tampering request signature verification mechanism.
- Enables traffic restriction and control to protect the backend server.
- Has the circuit breaker feature to protect the backend when the backend system is abnormal.
- Has the dynamic routing feature to support dynamic configuration of routing rules.

Advantages

Mobile Gateway Service has the following advantages:

- Adapts to various terminals and connects heterogeneous backend services with simple configuration.
- Automatically generates mobile SDK to realize frontend-backend separation, improving development efficiency.
- Supports service registration, and discovery and management, and implements service aggregation and integration to reduce management cost and security risk.
- Provides optimized data protocol and communication protocol, enhancing the network communication quality and efficiency.

Application scenarios

The Mobile Gateway Service is generally applied in the following scenarios:

- Open mobile service capability

  With the rapid development of mobile Internet and inclusive financing, enterprises are increasingly eager to open their existing mature backend services. With Mobile Gateway Service, you can develop your mobile servicing capability without any additional configuration.
- Single service with multi-terminal output

  The mobile Internet era requires service to support various types of terminal devices, which greatly increases the system complexity. Using Mobile Gateway Service, you can adapt service to multiple terminals by defining your service in mobile gateway.
- Standard and unified APIs open for heterogeneous services

  In many enterprises, the backend services are in multiple languages and structures. To open standard and unified service APIs to others, you only need to access the Mobile Gateway Service by following certain standards.

3.Terminology

API Group

The group to which API belongs. It can be a specific system name, module name, or an abstract identifier.

appId

Mobile application ID, which is generated upon mPaaS application creation.

HRPC

An RPC solution implemented on the basis of HTTP.

Mobile Gateway Service (MGS)

A component that provides gateway API service.

MPC

The abbreviation of mpaaschannel, which is a set of RPC solution implemented by mPaaS.

OperationType

The unique identifier of API service. It is the **OperationType** you entered when creating an API.

workspaceId

The ID of workspace on mobile development platform, which is used to isolate different workspaces.

4.Quick start

4.1. HTTP API

Quick Start guides you to quickly register and release an HTTP API service for mobile clients to call. The overall process is divided into five steps:

1. Register an API group
2. Create an API
3. Configure an API service
4. Test the API service
5. Generate a client SDK

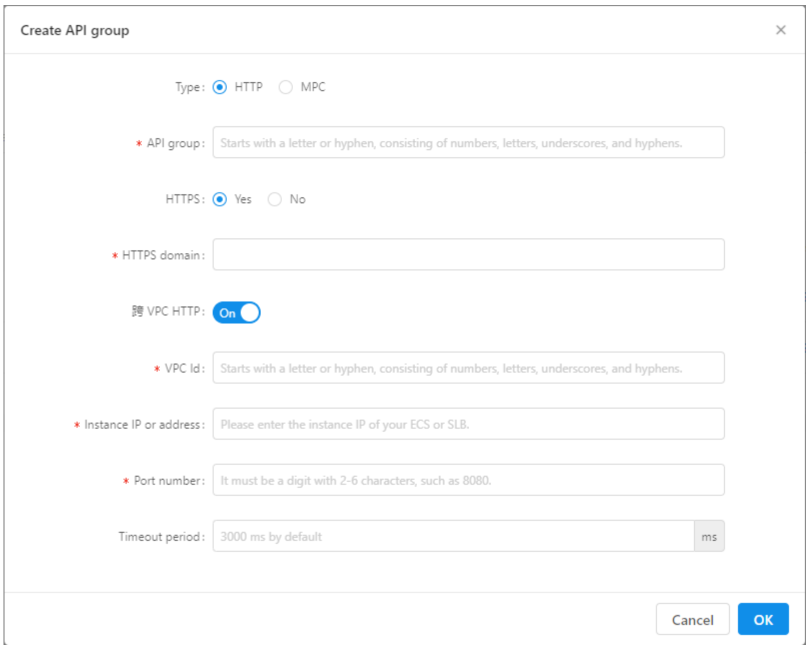
Prerequisites

1. Log on to the console and choose **Mobile Development Platform mPaaS** from the **Products and Services** drop-down list to go to the Mobile Development Platform homepage.
2. After you switch to the correct workspace, click the name of the app that needs to connect to the API service.
3. In the left-side navigation pane, click **Mobile Gateway Service**.

Register an API group

You can select an existing API group on the **Access System** page only after you have registered an API group. For more information, see the screenshot in Configure API Services.

1. Click the **API group** tab to go to the API Groups page.
2. Click **Create API group**. In the dialog box that appears, configure the API group information.



Create API group

Type: [✓ HTTP] MPC

* API group: Starts with a letter or hyphen, consisting of numbers, letters, underscores, and hyphens.

HTTPS: [✓ Yes] No

* HTTPS domain:

跨 VPC HTTP: On

* VPC Id: Starts with a letter or hyphen, consisting of numbers, letters, underscores, and hyphens.

* Instance IP or address: Please enter the instance IP of your ECS or SLB.

* Port number: It must be a digit with 2-6 characters, such as 8080.

Timeout period: 3000 ms by default ms

Cancel OK

   - **Type**: Select HTTP.
   - **API group**: Required. The name of the service system.
   - **Cross-VPC HTTP**: Specifies whether to allow cross-VPC service calls. If you turn on this switch, cross-VPC service calls are allowed. This feature is applicable to scenarios in which the MGS service is deployed in a different VPC from the VPC in which your own service is deployed. In network environment, MGS calls the backend service across VPCs.

     > **Note**
     > Currently, the cross-VPC HTTP feature is available only in the Shanghai Financial District environment.

     To enable the cross-VPC HTTP call feature, you must also complete the following configurations:

     - **Whether to support HTTPS** :MGS allows HTTPS listeners of SLB (SLB) to forward HTTPS requests to meet the requirements of encrypted data transmission. By default, HTTPS is supported. If you do not need to use HTTPS domain names, select **No**.
     - **HTTPS domain**: This parameter is required only if you select HTTPS. This parameter corresponds to the domain name of the certificate that is attached to the SLB instance.
     - **VPC Id**: the Virtual Private Cloud of the ECS instance or the network of the SLB instance where the corresponding service is deployed.
     - **Instance IP or address**: the primary private IP address of the ECS instance or the IP address of the SLB instance.

    - **Port number**: the port number of the service.

      > **Important**
      > - Currently, MGS does not support self-signed certificates. If you use cross-VPC HTTPS, you need to configure a CA to authentication certificate the domain name. Otherwise, the domain name resolution will fail.
      > - If you use an ECS instance to configure a backend service, you must enable the security group port of the ECS instance to prevent security policy interception. For relevant port IP information, please search group number 41708565 to join DingTalk group and contact technical helpdesk to obtain it.
      > - Only cross-VPC access in the same region is supported. Cross-VPC access in different regions is not supported. For example, services deployed in Hangzhou cannot call services in Shanghai. At the same time, the region is divided into financial and non-financial areas. For example, Hangzhou Financial District and Hangzhou Non-Financial District are considered as different regions.
      > - When you use both an SLB instance and an ECS instance to configure a backend service, make sure that the SLB instance and the ECS instance are deployed in the same VPC. That is, the information about the ECS Virtual Private Cloud where the service is deployed is consistent with the information about the network to which the SLB instance belongs.

  - **Service URL**: the HTTP or HTTPS URL of the business system. This item is required if the cross-VPC HTTP feature is not enabled.
  - **Timeout**: optional. The timeout period when a request is sent to the business system. Unit: milliseconds. Default value: 3000ms.
3. After you configure the API group, click **OK** to complete group creation.

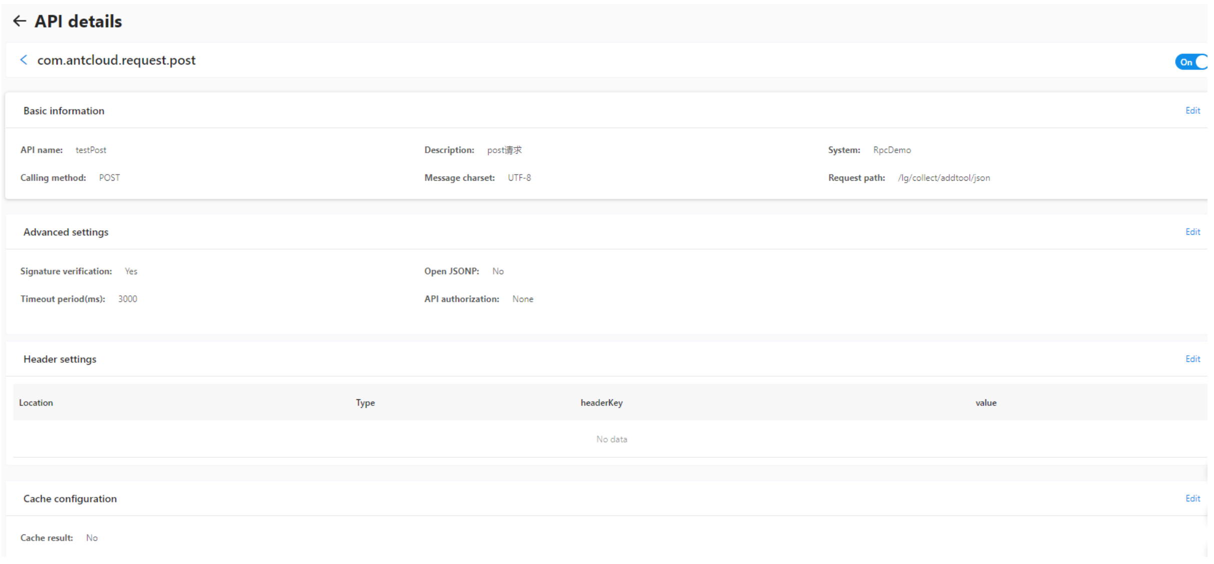
For more information about **API groups**, see API groups .

Create an API

1. Click the **Manage APIs** tab. On the APIs page, click **Create Native API**.
2. In the dialog box that appears, enter the API information.
   - **API Type**: The default value is HTTP.
   - **Add Method**: You can only manually register an HTTP API.
   - **operationType**: required. The unique identifier of the API service in the current environment and application. The naming rule is `Organization. Product domain. Products. Sub-products. Operation` .
3. Click **OK**.

Configure the API service

1. On the **Manage APIs** tab, click **Configure** in the Actions column corresponding to an API to go to the API configuration page.
2. In the API configuration section, click **Edit** to modify the parameters. After you modify the parameters, click **Save**.



API details

com.antcloud.request.post On

Basic information Edit

API name: testPost
Description: post请求
System: RpcDemo
Calling method: POST
Message charset: UTF-8
Request path: /lg/collect/addtool/json

Advanced settings Edit

Signature verification: Yes
Open JSONP: No
Timeout period(ms): 3000
API authorization: None

Header settings Edit

Location	Type	headerKey	value
No data			

Cache configuration Edit

Cache result: No

> **Important**
> - To get started, you can turn off **Signature verification** in the **Advanced settings** section.
> - For more information about signature verification, see Backend signature verification.
> - For more information about how to configure an API, see Configure an API.

3. Turn on the switch in the upper-right corner to make the API service **enabled**. Only API services in the activated state can be called.

Test the API service

For more information, see API testing.

Generate a client SDK

For more information, see Generate code.

Result

After you complete the preceding steps, the API service can be called by the client. For more information about client development, see the following **Client development guides**:

- Android
- iOS

- H5 JS

4.2. Dubbo API

The Dubbo service is only applicable to Private cloud. Quick Start guides you to register and release a Dubbo API service for mobile clients to call. The overall process is divided into six steps:

1. Server-side development
2. Register an API group
3. Create an API
4. Configure an API service
5. Test the API service
6. Generate a client SDK

Server development

Introduce the second-party package of the gateway

Introduce the following two-party package into the main `pom.xml` file of the project (if the original project already has dependencies, please ignore it). Use the latest version of `mobilegw-unify` series dependencies. The latest version is `1.0.5.20201010` .

```
<!-- mobilegw unify dependency-->
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-spi-dubbo</artifactId>
    <version>${the-lastest-version}</version>
</dependency>
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-spi-adapter</artifactId>
    <version>${the-lastest-version}</version>
</dependency>
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-log</artifactId>
    <version>${the-lastest-version}</version>
</dependency>
<dependency>
    <groupId>com.alipay.hybirdpb</groupId>
    <artifactId>classparser</artifactId>
    <version>1.2.2</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.5</version>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.72_noneautotype</version>
</dependency>

<! -- If pb is used, add the following dependency -->
<dependency>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java</artifactId>
    <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>io.protostuff</groupId>
    <artifactId>protostuff-core</artifactId>
    <version>1.3.8.20160722</version>
</dependency>
<dependency>
    <groupId>io.protostuff</groupId>
    <artifactId>protostuff-runtime</artifactId>
    <version>1.3.8.20160722</version>
</dependency>
<dependency>
    <groupId>io.protostuff</groupId>
    <artifactId>protostuff-api</artifactId>
    <version>1.3.8.20160722</version>
</dependency>
<dependency>
    <groupId>io.protostuff</groupId>
    <artifactId>protostuff-collectionschema</artifactId>
    <version>1.3.8.20160722</version>
</dependency>

<! -- Use the latest version of Apache for dubbo -->
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.7.8</version>
</dependency>
```

Define and implement service interfaces

1. Define the service interface `com.alipay.xxxx.MockRpc` based on your business requirements.

   We recommend that you define the input parameters in the method definition as VO. In this way, you can directly add parameters to VO later without changing the method declaration format. For more information about service interface definition specifications, see Business Interface Definition Specification.

2. Provides an implementation `com.alipay.xxxx.MockRpcImpl` for this interface.

Define an OperationType

Add a `@OperationType` annotation to the method of the service interface to define the interface name of the published service. The `@OperationType` has three parameter members. For ease of maintenance, please complete them:

- **value**: The unique identifier of the interface, which is globally unique in the gateway. Define rules: `Organization. Product domain. Products. Sub-products. Operation` . The value of this parameter should be defined as detailed as possible. Otherwise, if the value of this parameter is duplicated with that of another business party, the service cannot be registered.
- **name**: The name of the interface.
- **desc**: The description of the interface.

Example:

```
public interface MockRpc {

    @ OperationType(value="com.alipay.mock", name="DUBBO mock interface", desc="Complex mock interface")
    Resp mock(Req s);

    @OperationType(value="com.alipay.mock2",name="xxx", desc="xxx")
    String mock2(String s);
}

public static class Resp {
    private String msg;
    private int    code;

    // ignore getter & setter
}

public static class Req {
    private String name;
    private int age;

    // ignore getter & setter
}
```

Declare an API service

The purpose of this step is to declare the defined RPC service as an API that provides services through the `SPI` package provided by the gateway. The following two parameters are required:

- **registryUrl**: The address of the registry.
- **appName**: The name of the application on the business side.

You can declare an API service by `Spring` or `Spring Boot` .

Spring declaration mode

1. In the Spring configuration file of the corresponding bundle, declare the Spring Bean of the preceding service. Example:

   ```
   <bean id="mockRpc" class="com.alipay.gateway.spi.dubbo.test.MockRpcImpl"/>
   ```

2. In the Spring configuration file for the corresponding bundle, declare the `com.alipay.gateway.spi.dubbo.DubboServiceStarter` type of Spring bean. `DubboServiceStarter` will register all beans with `@OperationType` to the specified registry through the Dubbo protocol. Example:

   ```
    <bean id="dubboServiceStarter" class="com.alipay.gateway.spi.dubbo.DubboServiceStarter">
        <property name="registryUrl" value="${registry_url}"/>
        <property name="appName" value="${app_name}"/>
    </bean>
   ```

Spring Boot declaration mode

1. A Spring Bean that declares the above service as an annotation. Example:

   ```
   @Service
   public class MockRpcImpl implements MockRpc{
   }
   ```

2. `com.alipay.gateway.spi.dubbo.DubboServiceStarter` types of Spring beans are declared as annotations. `DubboServiceStarter` will register all beans with `@OperationType` to the specified registry through the Dubbo protocol. Example:

   ```
       @Configuration
       public class DubboDemo {
           @Bean(name="dubboServiceStarter")
           public DubboServiceStarter dubboServiceStarter(){
               DubboServiceStarter dubboServiceStarter = new DubboServiceStarter();
               dubboServiceStarter.setAppName("${app_name}");
               dubboServiceStarter.setRegistryUrl("${registry_url}");
               return dubboServiceStarter;
           }
       }
   ```

Register an API group

1. Go to the Mobile Gateway Management page.
   i. Log on to the console and choose **Mobile Development Platform mPaaS** from the **Products and Services** drop-down list to go to the Mobile Development Platform homepage.
   ii. After you switch to the correct workspace, click the name of the app for which you want to connect to the API service.
   iii. In the left-side navigation pane, choose **Background connection** > **Mobile Gateway Service**.
2. Click the **API group** tab. On the API Groups page, click **Create API group**.
3. In the dialog box that appears, enter the form information.
   - **Type**: Select DUBBO.
   - **API group**: Required. The name of the business system that provides the service. The name of the API group must be the same as that of the registered API application.
   - **Registry**: Required. The address of the registry.
   - **Timeout period**: Optional. The timeout period when a request is sent to the business system. Unit: milliseconds. Default value: 3000ms.
   - **Registry**: Enter the address of the registry. ZooKeeper clusters or direct connections are supported.
   - **Registry authentication**: allows you to manage permissions on the registry. Only authenticated users can access the registry. After you turn on the registry authentication switch, you need to set the corresponding username and password.
4. Click **OK**. For more information about API group configurations, see Configure API groups.

Create an API

1. Click the **APIs** tab. On the APIs page, click **Create API**.
2. In the dialog box that appears, set **API Type** to **DUBBO**, select **API Group**, select the required service from the `operationType` list, and then click **OK**.

Configure the API service

1. On the **API Management** tab, click **Configure** in the Actions column corresponding to an API to go to the API configuration page.
2. In the API configuration section, click **Modify** to edit the parameters. After you modify the parameters, click **Save**.

   Important
   - To get started, you can turn off **Signature Verification** in the **Advanced Settings** section. For more information about signature verification, see Backend signature verification.
   - For more information about how to configure an API, see Configure an API.
3. Check the switch in the upper-right corner to make sure that the API service is in the **Enabled** state. Only API services in the activated state can be called.

Test the API service

For more information, see API testing.

Generate a client SDK

For more information, see Generate code.

Result

After you complete the preceding steps, the API service can be called by the client. For more information about client development, see the following client development guides:

- Android
- iOS
- H5 JS

5.Client-side development guide

5.1. Android

5.1.1. Quick start

A gateway is a bridge between a client and a server. The client uses the gateway to access background service interfaces.

By using a gateway, you can:

- Encapsulates the communication between the client and the server by using a dynamic proxy.
- If the server and the client define the same interface, the server can automatically generate code and export it to the client.
- Unified exception handling for `RpcException`, pop-up dialog boxes, toast message boxes, etc.

Mobile gateways support two access methods: **Native AAR access** and **Component-based (Portal&Bundle) access**.

Prerequisites

- If you use native AAR, you must add mPaaS to the project Add mPaaS to your project.
- If you use component-based access, you must first complete the component-based access process General steps.

Add an SDK

Native AAR mode

Install the **Mobile Gateway Service** component in the project by using **AAR**. For more information, see Manage AAR components.

Component-based (Portal&Bundle)

Install the **Mobile Gateway Service** component on the **Components** page in the Portal and Bundle projects.

Initialize mPaaS

If you use **Native AAR** method, you must initialize mPaaS.

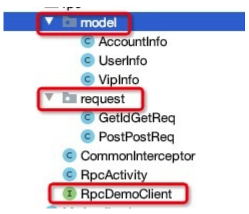
```
public class MyApplication extends Application {

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

For more details, see Initialize mPaaS.

Generate RPC code

After the app is connected to the background service in the Mobile Gateway console, log on to the mPaaS console. In the left-side navigation pane, choose **Mobile Gateway Service** > **Manage APIs** > **Generate code** to download the RPC code of the client. For more information, see API registration.

The downloaded RPC code structure is as follows, including RPC configuration, request model, and response model.



Call RPC

The client initiates the PRC call. Sample code:

```
// Obtain the client instance.
RpcDemoClient client = MPRpc.getRpcProxy(RpcDemoClient.class);
// Specify the request.
GetIdGetReq req = new GetIdGetReq();
req.id = "123";
req.age = 14;
req.isMale = true;
// Initiate an RPC request.
try {
    String response = client.getIdGet(req);
} catch (RpcException e) {
    // Handle RPC request exceptions.
    Log.i("RpcException", "code: " + e.getCode() + " msg: " + e.getMsg());
}
```

RPC call exceptions are thrown through the `RpcException`. You can handle the error based on the `RpcException` `code` result code. For more information about error codes, see Gateway result codes.

References

- Sample code
- Gateway result codes

- Key generation method

5.1.2. Advanced Guide

This topic describes how to configure the RPC interceptor, RPC request header, and RPC cookie.

RPC interception

In business development, you can use RPC interceptors to control network requests from clients, such as blocking network requests, prohibiting access to certain interfaces, or throttling.

Create a global interceptor

```
public class CommonInterceptor implements RpcInterceptor {

    /**
    * Preblock: The callback is invoked before the RPC is sent.
    * @param proxy RPC the proxy object.
    * @param clazz rpcface the model class. You can use the clzz parameter to determine which RPC model class is being called.
    * @param method The method of the current RPC call.
    * @throws RpcException
    * @return true indicates that the execution continues downward. false indicates that the current request is interrupted and RpcException i
s thrown. Error code: 9.
    */
    @Override
    public boolean preHandle(Object proxy,
                                ThreadLocal<Object> retValue,
                                byte[] retRawValue,
                                Class<?> clazz,
                                Method method,
                                Object[] args,
                                Annotation annotation,
                                ThreadLocal<Map<String, Object>> extParams)
                                throws RpcException {
            //Do something...
            return true;
     }

    /**Post-hold: The callback is invoked after the RPC is initiated.
    *@return true indicates that the execution continues downward. false indicates that the current request is interrupted and RpcException is
thrown. Error code: 9.
    */
    @Override
   public boolean postHandle(Object proxy,
                            ThreadLocal<Object> retValue,
                            byte[] retRawValue,
                            Class<?> clazz,
                            Method method,
                            Object[] args,
                            Annotation annotation) throws RpcException {
          //Do something...
          return true;
   }

   /**
   * Exception interception: callback after the RPC fails to be initiated.
   * @param exception Indicates the current RPC exception.
   * @return true indicates that the current exception continues to be thrown upward. false indicates that no exception is thrown and returns
normally without special requirements. Do not return false.
   */
   @Override
    public boolean exceptionHandle(Object proxy,
                                ThreadLocal<Object> retValue,
                                byte[] retRawValue,
                                Class<?> clazz,
                                Method method,
                                Object[] args,
                                RpcException exception,
                                Annotation annotation) throws RpcException {

        //Do something...
        return true;
    }
}
```

Register an interceptor

During the framework startup process, the interceptor is registered when the `RpcService` is initialized, for example:

```
public class MockLauncherApplicationAgent extends LauncherApplicationAgent {

    public MockLauncherApplicationAgent(Application context, Object bundleContext) {
        super(context, bundleContext);
    }

    @Override
    public void preInit() {
        super.preInit();
    }

    @Override
    public void postInit() {
        super.postInit();
        RpcService rpcService = getMicroApplicationContext().findServiceByInterface(RpcService.class.getName());
        rpcService.addRpcInterceptor(OperationType.class, new CommonInterceptor());
    }
}
```

Set RPC request headers

In the `initRpcConfig` method of the `MainActivity` class, set the RPC request headers. For more information, see Sample code.

```
private void initRpcConfig(RpcService rpcService) {
        // Set the request header.
        Map<String, String> headerMap = new HashMap<>();
        headerMap.put("key1", "val1");
        headerMap.put("key2", "val2");
        rpcInvokeContext.setRequestHeaders(headerMap);
    }
```

Set RPC cookies

Set a cookie

You can call the following API operations to set the RPC cookie. Where the `Your domain` rule is the first of the gateway URLs `.` and everything before the first `/` after it. For example, if the gateway URL is `http://test-cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm`, the `Your domain` is `.cloud.alipay.com`.

```
GwCookieCacheHelper.setCookies(Your domain, cookiesMap);
```

Remove cookies

You can call the following operation to remove a set cookie.

```
GwCookieCacheHelper.removeAllCookie();
```

Set SM3 Signature Verification

After RPC initialization, you can specify the global signature verification method as sm3 type by `MPRpc` the `setGlobalSignType` method of the class.

```
MPRpc.setGlobalSignType(TransportConstants.SIGN_TYPE_SM3);
```

5.2. iOS

5.2.1. Add a SDK

This topic describes how to connect a mobile analytics component to an iOS client. You can use CocoaPods to connect the mobile gateway SDK to the client based on an existing project.

Prerequisites

Before you add the mobile gateway SDK, make sure that you have completed the following operations:

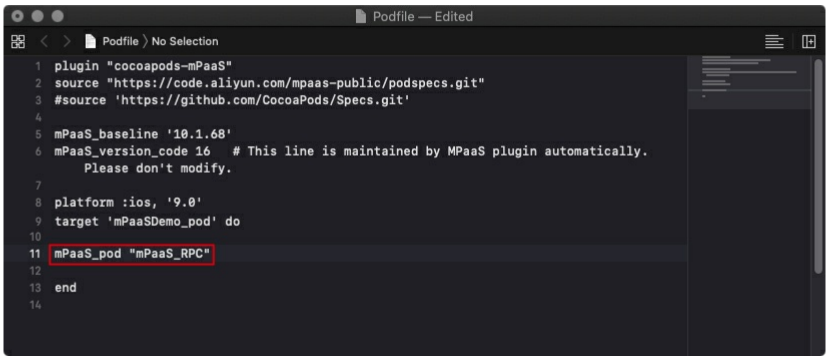
1. Connect the project to mPaaS. For more information, see Use CocoaPods.
2. Introduce a wireless bodyguard picture `yw_1222.jpg` for requesting signature: After the first step is completed, the wireless bodyguard picture will be automatically generated in the project, and you do not need any additional operation.

Add a SDK

Use the cocoapods-mPaaS plug-in to add the mobile gateway SDK.

you must perform the following operations:

1. In the Podfile file, use the `mPaaS_pod "mPaaS_RPC"` to add the mobile gateway component dependency.



```
plugin "cocoapods-mPaaS"
source "https://code.aliyun.com/mpaas-public/podspecs.git"
#source 'https://github.com/CocoaPods/Specs.git'

mPaaS_baseline '10.1.68'
mPaaS_version_code 16   # This line is maintained by MPaaS plugin automatically.
    Please don't modify.

platform :ios, '9.0'
target 'mPaaSDemo_pod' do

mPaaS_pod "mPaaS_RPC"

end
```

2. Run `pod install` on the command line to complete the connection.

What to do next

Use SDKs

5.2.2. Use SDKs

RPC-related modules are APMobileNetwork.framework and MPMgIsAdapter. We recommend that you use the APIs in MPMgIsAdapter.

This topic describes how to use the mobile gateway SDK by performing the following steps:

1. Initializing the gateway service
2. Generating RPC Code
3. Send reque
4. Request custom configurations
5. Custom RPC interceptors
6. Data encryption
7. Data signature

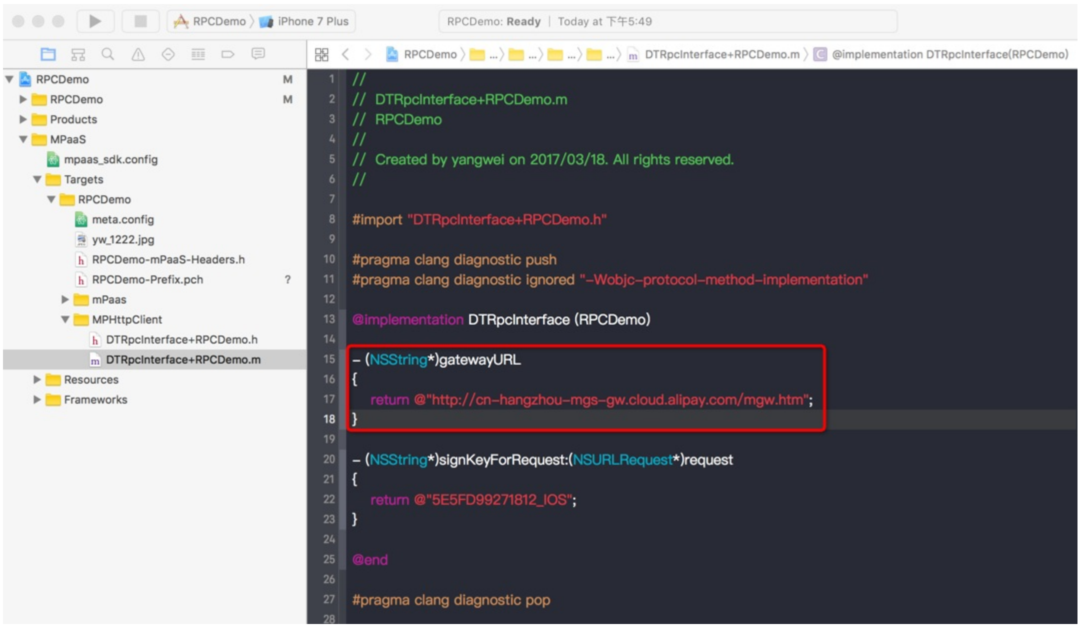
Initialize the gateway service

Call the following method to initialize the gateway service:

```
[MPRpcInterface initRpc];
```

Precautions for upgrading an earlier version

After the 10.1.32 version, you no longer need to add `Category` files of the `DTRpcInterface` class. The middle layer will implement packaging to read from the `meta.config`. After the version is upgraded, check whether the configuration of the old version exists in the project. If yes, remove it. Below is the `Category` file for the `DTRpcInterface` classes that should be removed for the new version.

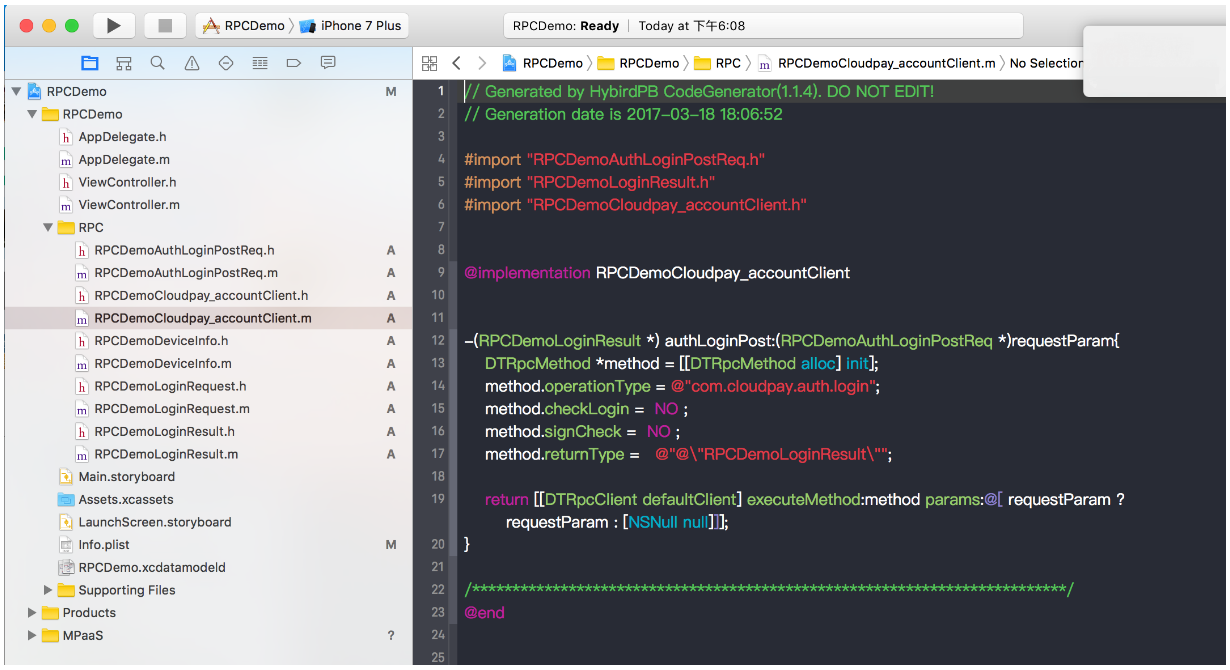


```
//
//  DTRpcInterface+RPCDemo.m
//  RPCDemo
//
//  Created by yangwei on 2017/03/18. All rights reserved.
//

#import "DTRpcInterface+RPCDemo.h"

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

@implementation DTRpcInterface (RPCDemo)

- (NSString*)gatewayURL
{
    return @"http://cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm";
}

- (NSString*)signKeyForRequest:(NSURLRequest*)request
{
    return @"5E5FD99271812_IOS";
}

@end

#pragma clang diagnostic pop
```

Generate RPC code

After the app is connected to the background service in the mobile gateway console, you can download the RPC code of the client. For more information, see Generate code.

The structure of the downloaded RPC code is as follows:



```
// Generated by HybirdPB CodeGenerator(1.1.4). DO NOT EDIT!
// Generation date is 2017-03-18 18:06:52

#import "RPCDemoAuthLoginPostReq.h"
#import "RPCDemoLoginResult.h"
#import "RPCDemoCloudpay_accountClient.h"


@implementation RPCDemoCloudpay_accountClient


-(RPCDemoLoginResult *) authLoginPost:(RPCDemoAuthLoginPostReq *)requestParam{
    DTRpcMethod *method = [[DTRpcMethod alloc] init];
    method.operationType = @"com.cloudpay.auth.login";
    method.checkLogin =  NO ;
    method.signCheck =  NO ;
    method.returnType =   @"@\"RPCDemoLoginResult\"";

    return [[DTRpcClient defaultClient] executeMethod:method params:@[ requestParam ?
        requestParam : [NSNull null]]];
}

/**************************************************************************/
@end

```

Where:

- `RPCDemoCloudpay_accountClient` is the RPC configuration.
- `RPCDemoAuthLoginPostReq` is the request model.
- `RPCDemoLoginResult` is the response model.

Send Request

RPC requests must be called in a sub-thread. You can use the sub-thread call interface encapsulated by the `MPRpcInterface` in the middle layer. The callback method is the main thread by default. The sample code is as follows:

```
- (void)sendRpc
{
    __block RPCDemoLoginResult *result = nil;
    [MPRpcInterface callAsyncBlock:^{
        @try
        {
            RPCDemoLoginRequest *req = [[RPCDemoLoginRequest alloc] init];
            req.loginId = @"alipayAdmin";
            req.loginPassword = @"123456";
            RPCDemoAuthLoginPostReq *loginPostReq = [[RPCDemoAuthLoginPostReq alloc] init];
            loginPostReq._requestBody = req;
            RPCDemoCloudpay_accountClient *service = [[RPCDemoCloudpay_accountClient alloc] init];
            result = [service authLoginPost:loginPostReq];
        }
        @catch (NSException *exception) {
            NSLog(@"%@", exception);
            NSError *error = [userInfo objectForKey:@"kDTRpcErrorCauseError"]; // Obtain the exception details.
            NSInteger code=error.code; // Obtain the error code of the exception details.
        }
    } completion:^{
        NSString *str = @"";
        if (result && result.success) {
            str = @"Logon succeeded";
        } else {
            str = @"Logon failed";
        }

        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:str message:nil delegate:nil
                                              cancelButtonTitle:nil otherButtonTitles:@"ok", nil];
        [alert show];
    }];
}
```

> **Note**
>
> To use the `try catch` to catch exceptions, when the gateway will be thrown, according to the Gateway result code description query reason.

Request custom configurations

The RPC request method description `DTRpcMethod` records the RPC request method name, parameters, and return type.

- If you do not need to sign the request, you can set the `DTRpcMethod` `signCheck` attribute to NO.

```
-(MPDemoUserInfo *) dataPostSetTimeout:(MPDemoPostPostReq *)requestParam
{
  DTRpcMethod *method = [[DTRpcMethod alloc] init];
  method.operationType = @"com.antcloud.request.post";
  method.checkLogin =  NO ;
  method.signCheck =  NO ;
  method.returnType =   @"@\"MPDemoUserInfo\"";

  return [[DTRpcClient defaultClient] executeMethod:method params:@[ ]];
}
```

- If you need to set the timeout period, you can configure the `DTRpcMethod` `timeoutInterval` properties.

```
-(MPDemoUserInfo *) dataPostSetTimeout:(MPDemoPostPostReq *)requestParam
{
  DTRpcMethod *method = [[DTRpcMethod alloc] init];
  method.operationType = @"com.antcloud.request.post";
  method.checkLogin =  NO ;
  method.signCheck =  YES ;
   method.timeoutInterval = 1; // This timeout period is the time when the client receives the response from the gateway. The timeout period
configured by the server is the time when the backend business system returns the response. The default value is 20s. If the setting is less
than 1, it is invalid.
  method.returnType =   @"@\"MPDemoUserInfo\"";

  return [[DTRpcClient defaultClient] executeMethod:method params:@[ ]];
}
```

- If you need to add Header to all APIs, you can use the following extension method. `DTRpcClient`

```
-(MPDemoUserInfo *) dataPostAddHeader:(MPDemoPostPostReq *)requestParam
{
  DTRpcMethod *method = [[DTRpcMethod alloc] init];
  method.operationType = @"com.antcloud.request.postAddHeader";
  method.checkLogin =  NO ;
  method.signCheck =  YES ;
  method.returnType =   @"@\"MPDemoUserInfo\"";

  // Add a header to the API.
  NSDictionary *customHeader = @{@"testKey": @"testValue"};
  return [[DTRpcClient defaultClient] executeMethod:method params:@[ ] requestHeaderField:customHeader responseHeaderFields:nil];
}
```

- If you want to add Header to all APIs, use interceptors. For more information, see Use interceptors. For more information, see Sample code.
- The `checkLogin` attribute is used for interface `session` verification and needs to be completed in conjunction with the gateway console. The default setting is NO.

Custom RPC Interceptor

Based on business requirements, you may need to perform logic processing before the RPC is sent or after the RPC is processed. The RPC module provides an interceptor mechanism to handle such requirements.

Custom interceptor

Create interceptors and implement protocol - `<DTRpcInterceptor>` methods to process RPC requests before and after operations.

```
@interface HXRpcInterceptor : NSObject<DTRpcInterceptor>

@end

@implementation HXRpcInterceptor

- (DTRpcOperation *)beforeRpcOperation:(DTRpcOperation *)operation{
    // TODO
    return operation;
}

- (DTRpcOperation *)afterRpcOperation:(DTRpcOperation *)operation{
   // TODO
   return operation;
}
@end
```

Register an interceptor

You can call the extension interface of the middle layer to register custom subinterceptors in the interceptor container.

```
HXRpcInterceptor *mpTestIntercaptor = [[HXRpcInterceptor alloc] init]; // Custom subinterceptor
    [MPRpcInterface addRpcInterceptor:mpTestIntercaptor];
```

Data encryption

RPC provides various data encryption configuration features. For more information, see Data encryption.

Data signature (supported in 10.2.3)

The 10.2.3 baseline RPC provides a variety of data signature configuration features. 10.2.3 The baseline upgraded the wireless bodyguard SDK to support the national secret signature. After the upgrade, the wireless bodyguard picture needs to be replaced with V6 version to use this baseline.

The 10.1.68 baseline defaults to the V5 version. Follow these steps to use the plug-in to generate a V6 image and replace the original `yw_1222.jpg` wireless bodyguard image.

1. Install the mPaaS command-line tool. The command-line tool is included in the plug-in installation. You can set N by removing the Xcode signature.
2. Use the following command line to generate a new wireless bodyguard image.

```
mpaas inst sgimage -c /path/to/Ant-mpaas-0D4F511111111-default-IOS.config -V 6 -t 1 -o /path/to/output --app-secret sssssdderrff --verbose
```

> **Note**
> Replace the config file directory, target file directory, and appsecret parameters with actual values.

3. If you want the wireless bodyguard to support the national secret function, please follow the following code to configure the category code to set the signature algorithm, the default configuration is not MPAASRPCSignTypeDefault, the signature algorithm is MD5.

   Optional values of the signature algorithm are as follows:

   - MD5: MPAASRPCSignTypeDefault (default)
   - SHA256: MPAASRPCSignTypeSHA256
   - HMACSHA256: MPAASRPCSignTypeHMACSHA256
   - SM3: MPAASRPCSignTypeSM3

   Sample code:

```
#import <APMobileNetwork/DTRpcInterface.h>

@interface DTRpcInterface (mPaaSDemo)

@end

@implementation DTRpcInterface (mPaaSDemo)

- (MPAASRPCSignType)customRPCSignType
{
    return MPAASRPCSignTypeSM3;
}

@end
```

Links

- Wireless bodyguard result code description
- Gateway Result Code Description

5.3. H5 JS programming

Currently, many mobile app frontend uses JavaScript (JS) language for coding. mPaaS also provides a mobile web solution- H5 container HTML5 Container overview. H5 is hosted on Android and iOS and requires client integration.

After the client is integrated to the H5 container, the frontend can easily use the gateway:

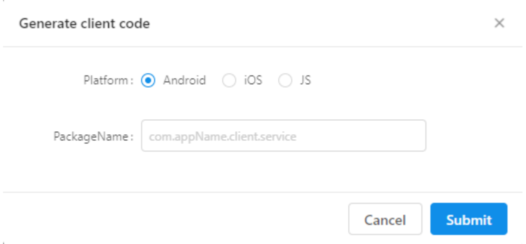
- Encapsulates the communication between the client and the server by using a dynamic proxy.
- If the server and the client define the same interface, the server can automatically generate code and export it to the client.
- Unified exception handling for `RpcException`, pop-up dialog boxes, toast message boxes, etc.

Prerequisites

Before performing H5 JS programming, make sure that the Android/iOS client has been connected to the H5 container. For more information, see Connect to Android and Connect to iOS.

Generate JS code

After you connect the mobile gateway console to the App background service, you can use the console to generate a JS SDK for RPC for the client to call. For more information, see Generate code.


Generate client code

Platform: [✓ Android] iOS JS

PackageName: com.appName.client.service

Currently, for each API, the following template code is generated based on the agreed interface parameters:

```
var params = [{
    "_requestBody":{"userName":"", "userId":0}
}]
var operationType = 'alipay.mobile.ic.dispatch'

AlipayJSBridge.call('rpc', {
  operationType: operationType,
  requestData: params,
  headers:{}
}, function (result) {
  console.log(result);
});
```

When the front end needs to use RPC, it will directly use the above template to fill in the call request parameters.

Call the RPC interface

JS calls RPC as follows:

```
AlipayJSBridge.call('rpc', {
  operationType: 'alipay.client.xxxx',
  requestData: [],
  headers:{}
}, function (result) {
  console.log(result);
});
```

Parameter Description

Parameter	Type	Required	Default value	Description
operationType	string	Y		RPC service name
requestData	array	Y		The parameters of the RPC request. You need to configure the parameters based on the specific RPC API.
headers	dictionary	Y	{}	The headers set by the RPC request.
gateway	string	Y	alipay gateway	Gateway address
compress	boolean	Y	true	Indicates whether request GZIP compression is supported.
disableLimitView	boolean	Y	false	Specifies whether to automatically pop up the unified traffic limit pop-up window when the RPC gateway is traffic limited.

Request Result

The returned results.	Type	Description
result	dictionary	The result of the RPC response. String values that are not in a dictionary structure are put into a dictionary structure with a key of `resData` .

Error code

Error code	Description
10	Network error.
11	The request took longer than the server was prepared to wait.
Others	Defined by the mobilegw gateway.

6.Server-side development guide

6.1. Backend signature verification description

Mobile Gateway provides the server-side HTTP service signature verification function to improve data security from the gateway to the server.

- After you enable signature verification for an API group in the Gateway console, Mobile Gateway creates a signature for each API request in the group. You can create a public /private key for the signature in the Gateway console.
- After the server reads the signature string, it calculates the local signature of the received request and compares it with the received signature to determine whether the request is valid.

Read signature

The signature calculated by the mobile gateway is stored in the header of the request, and the header key is `X-Mgs-Proxy-Signature` .

The key key configured in the API group can be used to distinguish and obtain keys corresponding to different key values. Header keys are `X-Mgs-Proxy-Signature-Secret-Key` .

Signature verification method

Organization signature data

```
String stringToSign =
HTTPMethod + "\n" +
Content-MD5 + "\n" +
Url
```

- `HTTPMethod` : All uppercase HTTPMethod, such as `PUT` or `POST` .
- `Content-MD5` : The MD5 hash of the request body. The calculation method is as follows:
  i. If the `HTTPMethod` is not one of PUT or POST, MD5 is an empty string `""` ; otherwise, the second step is executed.
  ii. If the request contains a body and the body is a form, the MD5 value is an empty string `""` . Otherwise, perform step 3.
  iii. Use the following method to calculate the MD5. If the request does not contain a body, the `bodyStream` is a string `"null"` .

```
String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes("UTF-8")));
```

> **Important**
> Even if the `content-MD5` is an empty string `""` , the newline character "\n" after the `content-MD5` in the signing method cannot be omitted, i.e. there will be two consecutive "\n" in the signature at this time.

- `Url` : The path, query, and form parameters in the body are assembled. Assume that the request format is `http://ip:port/test/testSign?c=3&a=1` and the parameters in the Form are `b=2&d=4` . The assembly steps are as follows:
  i. Obtain the path: `ip:port` is the path after, `?` The previous part. In this case, the `/test/testSign` .
  ii. If both the Query and Form parameters are empty, the `Url` is Path. Otherwise, the next step is performed.
  iii. Concatenate the required parameters. Sort the parameters in the query and form by key and lexicographic order, and then concatenate them into `Key1=Value1&Key2=Value2&...&KeyN=ValueN` . In this case, the `a=1&b=2&c=3&d=4` .

> **Note**
> You can specify multiple values for a query or form parameter. You can specify only the first `Value` .

  iv. The concatenated URL. The URL is `Path?Key1=Value1&Key2=Value2&...&KeyN=ValueN` . In this case, the `/test/testSign?a=1&b=2&c=3&d=4` .

Verify the signature

- Use the MD5 algorithm to verify signatures

```
String sign = "xxxxxxx";// The signature passed by the mobile gateway.
  String salt ="xxx";     //MD5 Salt

  MessageDigest digest = MessageDigest.getInstance("MD5");
  String toSignedContent = stringToSign + salt;
  byte[] content = digest.digest(toSignedContent.getBytes("UTF-8"));
  String computedSign = new String(Hex.encodeHexString(content));

  boolean isSignLegal = sign.equals(computedSign) ? true : false;
```

- Use the RSA algorithm to verify signatures

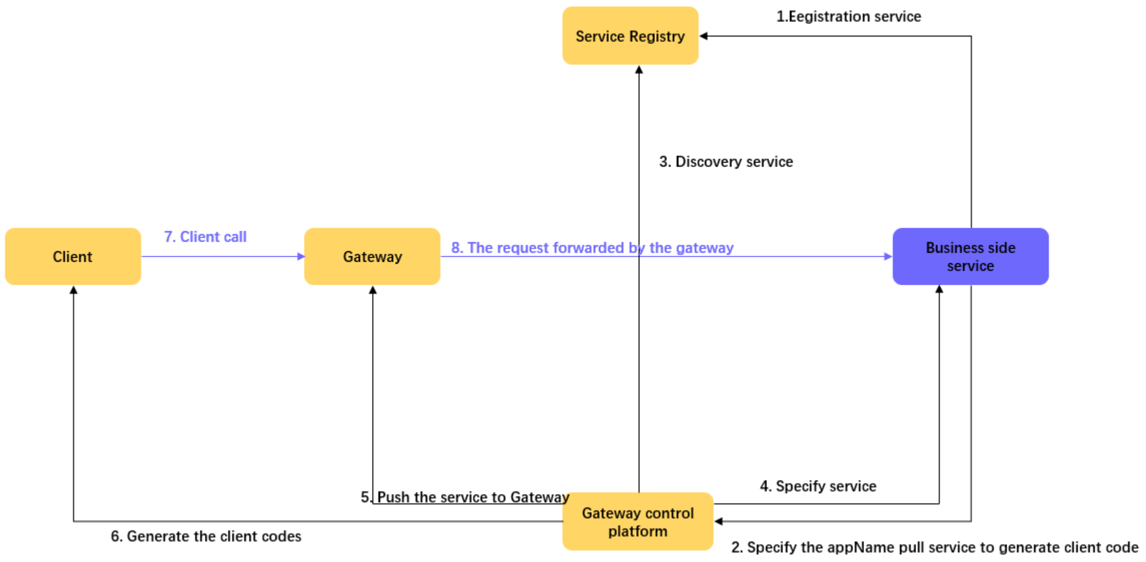
```
String sign = "xxxxxxx"; // The signature passed by the mobile gateway.
  String publicKey ="xxx"; // The RSA public key of the mobile gateway.

  PublicKey pubKey = KeyReader.getPublicKeyFromX509("RSA", new ByteArrayInputStream(publicKey.getBytes()));
  java.security.Signature signature = java.security.Signature.getInstance("SHA1WithRSA");
  signature.initVerify(pubKey);
  signature.update(stringToSign.getBytes("UTF-8"));

  boolean isSignLegal = signature.verify(Base64.decodeBase64(sign.getBytes(""UTF-8"")));
```

Examples

For more information, see HttpSignUtil.java.

6.2. Service definition and development

This document is only applicable to systems that are integrated with gateway SPI, such as business systems that expose mpaaschannel or Dubbo API services. For business systems that use HTTP APIs, you do not need to view this document.


Service Registry
1.Eegistration service
3. Discovery service
Client
7. Client call
Gateway
8. The request forwarded by the gateway
Business side service
4. Specify service
5. Push the service to Gateway
Gateway control platform
6. Generate the client codes
2. Specify the appName pull service to generate client code

Introducing the Gateway Second-Party Package

Introduce the following two-party package into the main `pom.xml` file of the project (if the original project already has dependencies, please ignore it).

You must reference all the basic dependencies. You can use the type of the API that you want to integrate.

Basic Dependency

```
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-spi-adapter</artifactId>
    <version>1.0.5.20201010</version>
</dependency>
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-log</artifactId>
    <version>1.0.5.20201010</version>
</dependency>
<dependency>
    <groupId>com.alipay.hybirdpb</groupId>
    <artifactId>classparser</artifactId>
    <version>1.2.2</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.5</version>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.72_noneautotype</version>
</dependency>
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-common</artifactId>
    <version>1.0.5.20201010</version>
</dependency>
```

MPC dependency

```
<dependency>
    <groupId>com.alipay.gateway</groupId>
    <artifactId>mobilegw-unify-spi-mpc</artifactId>
    <version>1.0.5.20201010</version>
</dependency>
<dependency>
    <groupId>com.alipay.mpaaschannel</groupId>
    <artifactId>common</artifactId>
    <version>2.4.2019040801</version>
</dependency>
<dependency>
    <groupId>com.alipay.mpaaschannel</groupId>
    <artifactId>tenant-client</artifactId>
    <version>2.4.2019040801</version>
</dependency>
```

Define and implement the service API

According to business requirements, define the service API: `com.alipay.xxxx.MockRpc` , and provide the implementation `com.alipay.xxxx.MockRpcImpl` of the API.

Note

- The input parameters in the method definition are defined as VO as much as possible. If you add parameters later, you can add parameters in VO without changing the declaration format of the method.
- For more information about service API definition specifications, see Service API definition specifications.

Define operationType

Add a `@OperationType` annotation to the method of the service API to define the API name of the published service. `@OperationType` have three parameter members:

- `value` : The unique identifier of the RPC service. The definition rule is `the organization. Product domain. Products. Sub-products. Operation` .
- `name` : the Chinese name of the API.
- `desc` : the description of the API.

Note

- The `value` is globally unique in the gateway. Try to define it in detail. Otherwise, it may be the same as the value of other business parties, resulting in failure to register the service.
- For ease of maintenance, be sure to fill in the three fields of the full `@OperationType` .

Example:

```
public API MockRpc {

    @OperationType("com.alipay.mock")
    Resp mock(Req s);

    @OperationType("com.alipay.mock2")
    String mock2(String s);
}

public static class Resp {
    private String msg;
    private int    code;

    // ignore getter & setter
}

public static class Req {
    private String name;
    private int age;

    // ignore getter & setter
}
```

Then, the defined API service is registered to the specified registry by using the SPI package provided by the gateway.

Register the MPC API service

The following parameters are required to register the MPC API service:

- `registryUrl` : The value is the address of the registry. The address of the shared Fintech registry is the `mpcpub.mpaas.cn-hangzhou.aliyuncs.com` .
- `appName` : The value is the application name of the business party and is the same as the API group name.
- `workspaceId` : The workspace ID of the application environment.
- `projectName` : The projectName of the tenant to which the application belongs, which is the same as the project name in the API group.
- `privateKeyPath` : The ClassPath that stores the RSA private key, which is used to verify validity when a connection is established with the mpaaschannel. We recommend that you place it in a `/META-INF/config/rsa-mpc-pri-key-{env}.der` , `{env}` different environments, such as dev, sit, and prod.

Configure a public key

Log on to the Apsara Uni-manager Management Console. In the left-side navigation pane, choose **Code Management** > **Interface Keys** > **Configuration**. On the page that appears, configure the RSA public key.

The method for generating an RSA public-private key is as follows, where the public key is configured on the console and the private key file is configured in the `${privateKeyPath}` of the backend application:

```
* the way to generate key pair:
* ### Generate a 2048-bit RSA private key
*
* $ openssl genrsa -out private_key.pem 2048
*
* ### Convert private Key to PKCS#8 format (so Java can read it)
*
* $ openssl pkcs8 -topk8 -inform PEM -outform DER -in private_key.pem -out private_key.der -nocrypt
*
* ### Output public key portion in DER format (so Java can read it)
*
* $ openssl rsa -in private_key.pem -pubout -outform DER -out public_key.der
*
* ### change to base64:
*
* ## The generated private key, which is configured in the backend application
* $ openssl base64 -in private_key.der -out private_key_base64.der
*
* ## The generated public key, which is configured in the console port key
* $ openssl base64 -in public_key.der -out public_key_base64.der
*
* ### remember to clear the whitespace chars and line breaks before submit!!!
```

Spring mode

1. In the spring configuration file of the corresponding bundle, declare the spring bean of the defined service.

```
 <bean id="mockRpc" class="com.alipay.gateway.spi.mpc.test.MockRpcImpl"/>
```

2. In the spring configuration file for the corresponding bundle, declare the starter bean that exposes the service.
   The API `MpcServiceStarter` registers all beans with `OperationType` to the specified registry through the mpaaschannel protocol.

```
<bean id="mpcServiceStarter" class="com.alipay.gateway.spi.mpc.MpcServiceStarter">
 <property name="registryUrl" value="${registy_url}"/>
 <property name="appName" value="${app_name}"/>
 <property name="workspaceId" value="${workspace_id}"/>
 <property name="projectName" value="${project_name}"/>
 <property name="privateKeyPath" value="${privatekey_path}"/>
</bean>
```

Spring-boot mode

Spring-boot is essentially the same as spring, except that the registration method is changed to annotation instead of configuring xml files.

1. Register the defined service as a bean by way of annotations:

```
@Service
public class MockRpcImpl implements MockRpc{
}
```

2. Define the starter of the exposed service as an annotation:

```
@Configuration
public class MpaaschannelDemo {
 @Bean(name="mpcServiceStarter")
 public MpcServiceStarter mpcServiceStarter(){
      MpcServiceStarter mpcServiceStarter = new MpcServiceStarter();
      mpcServiceStarter.setWorkspaceId("${workspace_id}");
      mpcServiceStarter.setAppName("${app_name}");
      mpcServiceStarter.setRegistryUrl("${registy_url}");
      mpcServiceStarter.setProjectName("${project_name}");
      mpcServiceStarter.setPrivateKeyPath("${privatekey_path}");
      return mpcServiceStarter;
 }
}
```

Configure MPC logs

To facilitate troubleshooting, you can configure MPC-related logs as appropriate. The following example uses `log4j` configuration:

```
!" -- [MPC Logger] tenant link: records the link information and settings information. -->
    <appender name="MPC-TENANT-LINK-APPENDER" class="org.apache.log4j.DailyRollingFileAppender">
        <param name="file" value="${log_root}/mpaaschannel/tenant-link.log"/>
        <param name="append" value="true"/>
        <param name="encoding" value="${file.encoding}"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d [%X{remoteAddr}][%X{uniqueId}] %-5p %c{2} - %m%n"/>
        </layout>
    </appender>

    <! -- [MPC Logger] Records data related to a stream (including a pair of tenant streams <-> component streams) -->
    <appender name="MPC-STREAM-DATA-APPENDER" class="org.apache.log4j.DailyRollingFileAppender">
        <param name="file" value="${log_root}/mpaaschannel/stream-data.log"/>
        <param name="append" value="true"/>
        <param name="encoding" value="${file.encoding}"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d [%X{remoteAddr}][%X{uniqueId}] %-5p %c{2} - %m%n"/>
        </layout>
    </appender>


<! -- [MPC Logger] tenant logs -->
    <logger name="TENANT-LINK-DIGEST" additivity="false">
        <level value="INFO" />
        <appender-ref ref="MPC-TENANT-LINK-APPENDER" />
        <appender-ref ref="ERROR-APPENDER" />
    </logger>

    <! -- [MPC Logger] component log -->
    <logger name="STREAM-DATA-DIGEST" additivity="false">
        <level value="INFO" />
        <appender-ref ref="MPC-STREAM-DATA-APPENDER" />
        <appender-ref ref="ERROR-APPENDER" />
    </logger>
```

The returned results.

After completing the preceding steps, you can perform a series of operations on the gateway to expose the defined API service to the client. For more information, see Register an API operation.

6.3. Gateway auxiliary class usage instructions

This article describes the use of relevant auxiliary classes used in gateways, including interceptor classes, MobileRpcHolder, and gateway error codes.

Implement the interceptor function

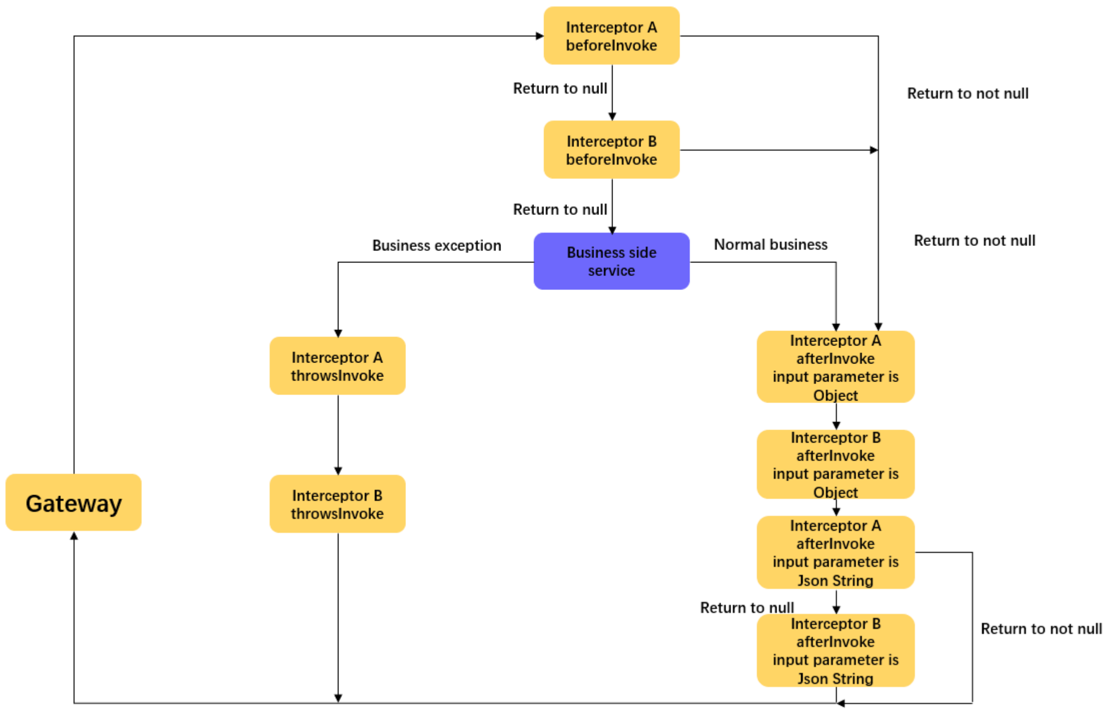
Interceptors are only applicable to non-HTTP services.

`mobilegw-unify-spi-adapter.jar` actually call the business method through Java reflection, that is, the method specified by the `OperatioinType` . In the process of method invocation, the business side can implement the interceptor defined in the SPI package to implement the extension.

The gateway's SPI package defines two interceptors: the `AbstractMobileServiceInterceptor` abstract class and the `MobileServiceInterceptor` interface.

AbstractMobileServiceInterceptor

The `MobileServiceInterceptor` mainly provides four methods: `beforeInvoke` , `afterInvoke` (divided into two types: one type of input parameter is the object returned by the business, and the other type of input parameter is the JSON string converted from the object), `throwsInvoke` , and `getOrder` .


Interceptor A beforeInvoke
Return to null
Interceptor B beforeInvoke
Return to null
Return to not null
Return to not null
Business exception
Business side service
Normal business
Interceptor A throwsInvoke
Interceptor B throwsInvoke
Interceptor A afterInvoke input parameter is Object
Interceptor B afterInvoke input parameter is Object
Interceptor A afterInvoke input parameter is Json String
Return to null
Interceptor B afterInvoke input parameter is Json String
Return to not null
Gateway

As shown in the preceding figure, the interceptor mainly intercepts in the following three situations:

- Before the method is called: that is, the `beforeInvoke` method, which has a return value. Once the return value of the method is not empty, the gateway determines that the interception is successful and will skip the `beforeInvoke` methods of the remaining interceptors, while skipping the method of the calling business side and going directly to the `afterInvoke` method of the interceptor.
- After the method is called: the method is `afterInvoke` . There are two types of `afterInvoke` . One type of input parameter is Object, which is the object returned by the business side. This method has no return value and will be executed by all interceptors. The other type of input parameter is a JSON string converted from Object. This method can change the incoming JSON-formatted data and return it. If the return value is not empty, the gateway determines that the interception is successful and subsequent interceptors are ignored.
- The method has an exception: that is, the `throwsInvoke` method. This method has no return value, and all interceptors of this method will be executed. It is called when an exception occurs on the business side.

MobileServiceInterceptor

`MobileServiceInterceptor` inherits the `Ordered` API of the framework, the interceptor implemented by the business side can also specify the execution order by implementing the `getOrder` method, the smaller the value set, the higher the priority of execution.

Example

1. Code your own interceptor classes, and inherit from `AbstractMobileServiceInterceptor` classes, or implement `MobileServiceInterceptor` interfaces.

```
public class MyInterceptor implements MobileServiceInterceptor {

 /*
 Description
     method: the method of the business party (the method defined by @ OperatioinType)
     args: an object array, that is, the input parameters of the business-side method. The number of input parameters is equal to the array
size.
          When used, the business party performs type conversion as needed.
     bean: the interface instance of the business side.
 Response parameters:
     Object: Data can be returned in the interceptor. Once the return value is not empty, the gateway considers that it has been intercepted
and does not call the business method again.
         At the same time, the beforeInvoke method of other interceptors is directly skipped, and the afterInvoke method in the interceptor
is executed.
 */

 @Override
 public Object beforeInvoke(Method method, Object[] args, Object target) {
     //Do Something
     return null;
 }

 /*
 * Parameter description
 * returnValue: the object returned by the business method.
 * Other parameters as above
 */
 @Override
 public void afterInvoke(Object returnValue, Method method, Object[] args, Object target) {
     // Note: The input parameter is the object returned by the business party.
 }

 @Override
 public String afterInvoke(String returnJsonValue, Method method, Object[] args, Object target) {
     // Note: The input parameter is a JSON-formatted string converted from the object returned by the business party.
     // The new JSON-formatted data can be returned.
     return null;
 }

 @Override
 public void throwsInvoke(Throwable t, Method method, Object[] args, Object target) {
 }

 @Override
 public int getOrder() {
     // The highest level (the smallest value) and the lowest level (the largest value).
     return 0;
 }
}
```

2. Releases the implemented class `MyInterceptor` as Bean.
   - Spring Boot: Add the annotation `@service` directly to the class.

     ```
     @service
     public class MyInterceptor implements MobileServiceInterceptor{}
     ```

   - Spring: Declaration in the `xml` file of the configuration.

     ```
     <bean id="myInterceptor" class="com.xxx.xxx.MyInterceptor"/>
     ```

MobileRpcHolder auxiliary class

`MobileRpcHolder` is a static auxiliary class provided in the `mobilegw-unify-spi-adapter.jar`, which defines the relevant information in a request process, the most important definition is as follows:

```
Map<String, String> session Save the requested session
The Map<String, String> header holds information about the request's header
Map<String, String> context saves the context information of a gateway call
String operationType to save the operationType of this request
```

Before the service (i. e. `OperationType`) of the service side is called, the SPI service sets the `MobileRpcHolder` information according to the request `MobileRpcRequest` forwarded by the gateway. This information is cleared after the service is called.

The lifecycle of a `MobileRpcHolder` is the entire service invocation process, which is cleared after the invocation.

The business side can also set this information as needed. This information will always exist in the process of calling the business service, and the business service can obtain this information during the calling process. Specific settings can be used by interceptors to dynamically modify the information stored in the `MobileRpcHolder` before and after method calls.

The following examples illustrate how the `MobileRpcHolder` can be used.

Example

Here, we take modifying and obtaining a session as an example.

1. Modify the session. Create an interceptor. For more information, see the preceding interceptor example. The following example uses interception before a method call:

```
@Override
public Object beforeInvoke(Method method, Object[] args, Object target) {
    Map<String, String> session = MobileRpcHolder.getSession();
    session.put("key_test", "value_test");
    MobileRpcHolder.setSession(session);
}
```

This allows you to modify the session information in the `MobileRpcHolder` .

2. Gets the session. The business party can obtain the session information in the service defined by itself.

```
@OperationType("com.alipay.account.query")
public String mock2(String s) {
    Map<String, String> session = MobileRpcHolder.getSession();
}
```

Other information such as the header and context can be modified and obtained in the same way.

```
// Obtain all information about the header.
 Map<String,String> headers = MobileRpcHolder.getHeaders();
 // The context information refers to the context information in the request.
 Map<String,String> context = MobileRpcHolder.getRequestCtx();
 // Obtain the OperationType.
 String opt = MobileRpcHolder.getOperationType();
```

Error codes for using the gateway

Mobile gateways have a set of error code specifications. For more information, see Gateway result codes.

Note that `BizException 6666` , this error is thrown by the gateway after a service exception occurs.

If you want to return other error codes when specific errors occur, you can control RPC layer errors by throwing `RpcException(ResultEnum resultCode)` , such as `resultCode=1001` , which will return "no permission to access" to the client.

Sample code

```
@Override
public String mock2(String s) throws RpcException {
    try{
        test();
    }catch (Exception e){
        throw new RpcException(IllegalArgument);
    }
    return "11111111";
}
```

Custom error codes

If you want to use a custom error code, you cannot throw an exception when calling a business method.

The business method returns status code 6666 whenever an exception occurs. At the same time, the client receives the status code, that is, it considers that the service is incorrect and does not parse the data returned by the service. The client parses the returned data only when it receives the 1000 status code.

Specifically, the server and the client agree on specific error codes, and then catch all exceptions when calling business methods, and put the custom error codes in the returned data. In this way, the service is abnormal and the gateway returns 1000 success. At the same time, the client parses the returned data and extracts custom error codes.

7.Use the Tablestore console

7.1. API groups

API group is the group to which the API belongs. It can be a specific system name, module name, or abstract identifier.

Create an API group

Complete the following steps to create an HTTP API group:

1. Select **API group** tab to go to the API group list page.
2. click **Create API group**, and then fill out the form in the pop-up dialog box.
   - **Type**: It is HTTP by default.
   - **API group**: Required, it is the name of the business system which provides services.
   - **Host**: Required for HTTP service, it is the business system's HTTP/HTTPS URL.
   - **Timeout period**: Optional, it is the timeout period (in ms) for sending requests to the business system. It defaults to 3,000 ms.
3. After you fill out the form, click **OK** to submit.

Configure the API group

Complete the following steps to configure the API group:

1. In the API group list, find the **HTTP** group, and click **Details** in the **Operations** column of the API group to go to the API group details page.
2. On the detail page, click **Edit** at the upper-right corner to configure the group. The configuration items of HTTP group are as follows:
   - **Host**: The URL address of HTTP services.
   - **Timeout period**: In milliseconds, 3,000 ms by default.
   - **Verification signature**: Enable it if the business system needs to verify the caller's identity. See instructions on backend signature verification for how to verify. Once you turn the switch on, you must complete the following configuration.
     - **Encryption algorithm**: The algorithm for generating signature. Public cloud supports MD5 and RSA algorithms, while private cloud supports MD5, RSA, and MOBILEGW.
     - **Key**: The key used in backend signature, customizable.
     - **Key content**: The value used in backend signature.
       - When the signing algorithm is MD5, the content is customizable.
       - When the signing algorithm is RSA, the content is the public key of Mobile Gateway Service.
       - When the signing algorithm is SM2 or SM3, the content is customizable.

     For how to generate keys, see Key generation method.

7.2. API management

7.2.1. API registration

Mobile Gateway supports many different types of API services. This topic describes how to register an API by using the console.

The operations that you need to perform vary based on different types of API services.

Before adding an MPC API, make sure that the corresponding MPC API group has been created.

- HTTP API
- MPC API

Add HTTP API

Log on to the mPaaS console and perform the following steps to register an HTTP API:

1. In the left-side navigation pane, click **Mobile Gateway Service**.
2. On the **API management** tab, click **Create API**.
3. In the dialog box that appears, select the **HTTP** API type.
4. In the **operationType** bar, input value and click **OK** to complete the registration.
   - `operationType` is the unique identifier of the API service in the current environment and app.
   - `operationType` definition rules: `Organization. Product domain. Products. Sub-products. Operation` .

Add MPC API

Complete the following steps to automatically pull the MPC API:

1. In the left-side navigation pane, click **Mobile Gateway Service**.
2. On the **API management** tab, click **Create API**.
3. In the dialog box that appears, select the **MPC** API type.
4. Select the corresponding API group. In the API group, select the API service to be registered from the obtained API list.
5. Click **OK** to complete the registration.

7.2.2. Configure the API

After you register an API service, you must configure the relevant configurations to use the API service, especially the HTTP API service. The API can be called only when the service is in the **Activate** state. You must manually activate the HTTP API.

About this task

The API includes the following types of parameter configurations:

- Basic information :API name, API description, and access system. Different types of API services have different attributes.

- Advanced settings: Signature verification, ETag caching, and timeout period.
- Header Settings: You can add or remove headers for all requests.
- Thilling Configuration: Configure throttling for API calls.
- Cache Configuration: Caches API responses to reduce the pressure on the business system.
- Parameter settings: request parameters settings and response settings. This parameter is unique to HTTP APIs.

For detailed parameter introduction and configuration rules, click the name of the configuration type above.

Procedure

To configure the API, complete the following steps:

1. Log on to the mPaaS console. In the left-side navigation pane, click **Mobile Gateway Service**.
2. In the API list, find the API that you want to configure and click **Configure** in the Actions column.
3. Toggle the API switch in the upper-right corner of the details page to enable or disable the current API.
4. Click Edit to modify the parameters. For more information, see the following section.
5. Click **Save** to complete the configuration.

The configurations that you want to modify.

Basic Information

Edit the corresponding parameter values based on different types of API services:

- HTTP API
  - **API Name**: required. The name of the API to facilitate subsequent maintenance.
  - **Description**: Optional. The description of the API.
  - **Access System**: required. The business system to which the API belongs.
  - **Request Path**: required. The URL path. You can use ${} to include the path parameter, for example, `/pets/${id}` .
  - **Request Mode**: required. Valves GET, POST, PUT, DELETE, and HEAD.
  - **Packet Encoding**: required. The value must be in UTF-8 or GBK format.
- MPC API
  - **API Name**: required. The name of the API to facilitate subsequent maintenance.
  - **Description**: Optional. The description of the API.
  - **Access System**: the business system to which the API belongs.
  - **Interface Method**: the API server method.
  - **Interface Name**: the API server interface.

Advanced Settings

- **Signature Verification**: Select whether to enable signature verification. If this parameter is enabled, the signature of the client request is verified.
- **Timeout**: specifies the timeout period of the service. Unit: milliseconds. Timeout Priority: Interface Timeout Settings> System Timeout Settings> Default 3000 ms.
- **Open JSONP**: specifies whether to support cross-domain HTTP requests. JSONP allows quick cross-domain to use APIs.

Header Settings

The gateway supports adding or removing headers for all requests. In the **Header Settings** section, click Edit to open the edit mode. Then, click **Add** to add a rule. Each rule contains five attributes: **location**, **type**, **headerKey**, **value**, and **action**.

- **Location**: Select the request header or response header.
  - If you add a request header, the header is automatically added to the request. The business can obtain the header by MobileRpcHolder.
  - If you add a response header, the header is automatically added to the response. The client can obtain the header from the response.
- **Type**: You can select add or delete.
  - Add: adds a new header. If an existing header exists in the original request, the new header is overwritten.
  - delete: deletes a header. You can configure the value or not to delete the header. If you specify a value, the field is deleted only when the value matches the specified value.
- **headerKey** :headerKey can be any string that complies with RFC, except for HTTP-specific headers, such as host and content-type. HeadKey cannot be mpaasgw-specific headers, such as operation-type.
- **value**: can be any string.
- **Actions**: Delete the current header rule.

> **Note**
> Do not use the underscores "_" when defining HTTP headers.

Configure throttling

The throttling configuration includes the throttling mode, throttling value, and throttling response:

- **Throttling Mode**
  - **Disable**: does not limit API calls.
  - **Block**: If the number of requests exceeds the throttling threshold, requests are blocked.
- **Threshold Value** Set a proper throttling threshold based on your business requirements. Unit: seconds. If the throttling mode is set to Block and this value is exceeded, requests are throttling.
- **Threshold Response** The default response is `{"resultStatus":1002,"tips":"Too many customers, please wait"}` If you want to customize a throttling response, use the following format:

```
{
    "result": "==This is the custom response content. Enter==",
    "tips": "ok",
    "resultStatus": 1000
}
```

In the preceding formulation:

- `result` for customized response data, `JSON` format. The client uses this field for processing only when the `resultStatus` is 1000.
- `tips` is a custom throttling prompt. If the `resultStatus` is 1002, this field will be used to prompt the user.
- `resultStatus` the result code returned for throttling. For more information, see Gateway result codes.

Configure the cache settings

Caches API responses to reduce the pressure on the business system. For more information, see API cache.

Parameters Setting

The parameter settings apply to API services of the HTTP type. You do not need to configure parameters for APIs that are created in automatic import mode.

- **Request parameters**: Specify dynamic parameters and URL query parameters in Path. The parameter names must be unique.
  - **Parameter name**: required. The name of the parameter.

    > **Note**
    > If you set this parameter to Path, make sure that the name is the same as that in the request path. For example, if the Request Path parameter is set to `/pets/${id}`, the parameter name must be `id`.

  - **Parameter location**: Required. The parameter is in the path or query string.
  - **Type**: Required. Valid values: String, Int, Long, Float, Double, and Boolean.
  - **Default value**: Optional. The default value of the parameter.
  - **Description**: Optional. The description of the parameter.
- **Request Body**: the data model and Content-Type of the request body.

  > **Note**
  > The request body configuration is displayed only when the API is called in the POST mode.

  - **Type of request body**: The following types of request bodies are supported: String, Int, Long, Float, Double, Boolean, List, Map, and Object.
  - **Message type**: Supported message types include application/json, application/x-www-form-urlencoded, and application/protobuf.
- **Response result**: the type of the response result.
  - **Response result type**: Indicates a basic data type or a custom data model.

7.2.2.1. Procedure

After you register an API service, you must configure the relevant configurations to use the API service, especially the HTTP API service. The API can be called only when the service is in the **Enabled** state. You must manually activate the HTTP API.

About this task

The API includes the following types of parameter configurations:

- Basic information: The API name, API description, and integration system. Different types of API services have different attributes.
- Advanced settings: Signature verification, ETag cache, and timeout period.
- Header settings: You can add or remove headers for all requests.
- Throttling configuration: Configure throttling for API calls.
- Cache configuration: Cache API responses to reduce pressure on business systems.
- Parameter settings: Request parameters settings and response settings. This parameter is unique to HTTP APIs.

For detailed parameter introduction and configuration rules, click the name of the configuration type above.

To configure the API, complete the following steps:

1. Log on to the mPaaS console. In the left-side navigation pane, click **Mobile Gateway Service**.
2. In the API list, find the API that you want to configure and click **Configure** in the actions column.
3. Toggle the API switch in the upper-right corner of the details page to enable or disable the current API.
4. Click **Modify** to modify the parameters. For more information, see detailed configuration information.
5. Click **Save** to complete the configuration.

7.2.2.2. Basic information configuration

Edit the corresponding parameter values based on different types of API services:

- HTTP API
  - **API Name**: Required. The name of the API to facilitate subsequent maintenance.
  - **Description**: Optional. The detailed description of the API.
  - **Integration System**: Required. The business system to which the API belongs.
  - **Request Path**: Required. The URL path. You can use ${} to include the path parameter, for example, `/pets/${id}`.
  - **Request Mode**: Required. Valves GET, POST, PUT, DELETE, and HEAD.
  - **Packet Encoding**: Required. The value must be in UTF-8 or GBK format.

- MPC API
  - **API Name**: Required. The name of the API to facilitate subsequent maintenance.
  - **Description**: Optional. The detailed description of the API.
  - **Integration System**: The business system to which the API belongs.
  - **Interface Method**: The API server method.
  - **Interface Name**: The API server interface.

7.2.2.3. Advanced configurations

- **Signature verification**: Select whether to enable signature verification. If this switch is enabled, the signature of the client request is verified.
- **Timeout period**: the timeout period of the service. Unit: milliseconds.

  Timeout priority: Interface Timeout Settings> System Timeout Settings> Default 3000 ms.
- **Open JSONP**: indicates whether cross-domain HTTP requests are supported. JSONP allows quick cross-domain to use APIs.

7.2.2.4. Header settings

The gateway supports adding or removing headers for all requests. In the **Header Settings** section, click **Modify** to open the edit mode. Then, click the **Add** to add a rule. Each rule contains five attributes: **location**, **type**, **headerKey**, **value**, and **action**.

- **Location**: Select the request header or response header.
  - If you add a request header, the header is automatically added to the request. The business can obtain the header by MobileRpcHolder.
  - If you add a response header, the header is automatically added to the response. The client can obtain the header from the response.
- **Type**: You can select add or delete.
  - Add: adds a new header. If an existing header exists in the original request, the new header is overwritten.
  - delete: deletes a header. You can configure the value or not to delete the header. If you specify a value, the field is deleted only when the value matches the specified value.
- **headerKey**: headerKey can be any string that complies with RFC, except for HTTP-specific headers, such as host and content-type. headKey cannot be mpaasgw-specific headers, such as operation-type.
- **value**: Can be any string.
- **Action**: Delete the current header rule.

> **Note**
> Do not use the underscores "_" when defining HTTP headers.

7.2.2.5. Throttling configuration

The throttling configuration includes the throttling mode, throttling value, and throttling response:

- **Throttling Mode**
  - **Disable**: Does not limit API calls.
  - **Block**: If the number of requests exceeds the throttling threshold, requests are blocked.
- **Throttling Value** Set a proper throttling threshold based on your business requirements. Unit: seconds. If the throttling mode is Block and the value of this parameter is exceeded, requests are throttling.
- **Throttling Response** The default response to throttling is `{"resultStatus":1002,"tips":"Too many customers, please wait"}` If you want to customize a response to throttling, use the following format:

```
{
    "result": "==This is the custom response content. Enter==",
    "tips": "ok",
    "resultStatus": 1000
}
```

  In the preceding code:
  - `result` for customized response data, `JSON` format. The client uses this field for processing only when the `resultStatus` is 1000.
  - `tips` is a custom throttling prompt. If the `resultStatus` is 1002, this field will be used to prompt the user.
  - `resultStatus` the result code returned for throttling. For more information, see Gateway result codes.

7.2.2.6. Cache configuration

Cache API responses to reduce pressure on business systems. For more information, see API cache.

7.2.2.7. Parameter settings

The parameter settings apply to API services of the HTTP type. You do not need to configure parameters for APIs that are created in automatic import mode.

- **Request Parameters**: Specify dynamic parameters and URL query parameters in Path. The parameter names must be unique.
  - **Parameter Name**: Required. The name of the parameter.

    > **Note**
    > If you set this parameter to Path, make sure that the name is the same as that in the request path. For example, if the Request Path parameter is set to `/pets/${id}` , the parameter name must be `id` .

  - **Parameter Location**: Required. The parameter is in the path or query string.

- **Type**: Required. You can select String, Int, Long, Float, Double, or Boolean.
- **Default Value**: Optional. The default value of the parameter.
- **Description**: Optional. The description of the parameter.

- **Request Body**: The data model and Content-Type of the request body.

> **Note**
> The request body configuration is displayed only when the API is called in the POST mode.

- **Request Body Type**: The following types of request bodies are supported: String, Int, Long, Float, Double, Boolean, List, Map, and Object.
- **Message Type**: Supported message types include application/json, application/x-www-form-urlencoded, and application/protobuf.

- **Response Result**: Specifies the type of response result.
  - **Response Result Type**: Indicates a basic data type or a customized data model.

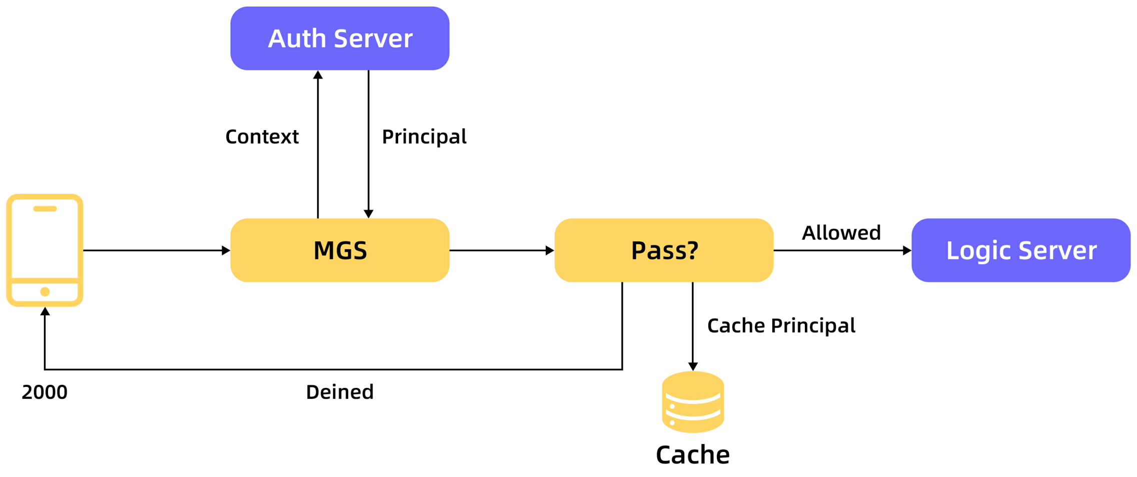
7.2.3. API authorization

Understand the usage scenarios of API authorization. Enable API authorization, configure authorization rules, define authorizer interfaces, and apply authorization rules to APIs based on your business requirements.

Features

The API authorization feature allows businesses to define common API access authorization rules on MGS:

1. Create an authorization API A and configure it in the gateway management, and then associate it with the service API B configuration.
2. When the client initiates a request to the backend service API B, MGS extracts the authorization parameters from the request header or cookie according to the API authorization configuration and puts them in the context and then calls the authorization API A associated with the service API B. The authorization API A server needs to perform service permission verification based on the parameters in the context.
3. If the verification is valid, MGS adds the verification result principal to the request header and passes it to the backend service API B. If caching is required, MGS caches the verification result principal to improve the performance of authorization.



Usage scenarios

Scenario 1

If a customer has a distributed session, a session ID is generated after login. The authorization process is as follows:

1. User A requests to log in to the interface. After successful login, user A generates a session ID and session information, saves them to the distributed cache, `sessionId: {username:A, age:18, ...}` them, and delivers the sessionId to the client.
2. User A requests an interface that requires logon authorization. The gateway obtains the sessionId from the request header and sends it to the authorization system. The authorization system obtains the user information from the distributed cache based on the sessionId and returns the `{username:A, age:18,...}` to the gateway.
3. The gateway determines that the logon is successful, adds the `{username:A, age:18,...}` to the header, and forwards the request to the backend service server.

Scenario 2

The client uses an HMAC-based authorization scheme. The authorization process is as follows:

1. After user A logs in successfully, a token is issued to the client and `token=hmac(username+password)` .
2. User A requests an interface that requires login authorization. The gateway obtains token from Header and sends it to the authorization system. The authorization system calculates HMAC again according to HMAC. If it matches, it returns the user information to `{username:A, age:18,...}` to the gateway.
3. The gateway determines that the logon is successful, adds the `{username:A, age:18,...}` to the request header, and forwards the request to the backend service server.

Procedure

Configure authorization rules

1. Log on to the mPaaS console. In the left-side navigation pane, choose **Background connection** > **Mobile Gateway Service**.
2. Click the **Manage gateway** tab. In the **API authorization** section, click **Create authorization API** or click **Details** in the Actions column of an existing authorization rule.

  - **Authorization API name**: Required. The name of the authorization rule.
  - **Authorization API**: Required. The API is used to verify the authorization of the request.
  - **Cache authorization result**: Indicates whether to cache the verification result of authorization.
  - **Cache TTL**: the cache lifetime of the verification result.
  - **Identity source**: If you click **Add source field**, enter the request parameters that is used for authorization and the request identity, which consists of the following fields:
    - **Location**: the location, `header` or `cookie`, of the parameter.
    - **Field**: the name of the parameter.

> **Note**
> If the identity source field in the API request is missing, the authorization verification fails.

Define the authorizer interface

> **Note**
> If the authorization interface provided by the backend system is HTTP, you need to configure the authorization API as the POST method.

Before adding an authorization relationship, the business system needs to develop a `Auth API` in advance. When the API needs to verify the authorization relationship, the `Auth API` is called for authorization verification. The definition of `Auth API` (request and response) follows the following criteria:

AuthRequest

```
public class AuthRequest {
  private Map<String,String> context;
}
```

AuthResponse

```
public class AuthResponse {
  private boolean success;
  private Map<String,String> principal;
}
```

Interface example

```
@PostMapping("/testAuth")
public AuthResponse testAuth(@RequestBody AuthRequest authRequest) {
    String sid = authRequest.getContext().get("sid");
    Map<String, String> principal = new HashMap<>();
    principal.put("uid", sid + "_uid");
    AuthResponse authResponse = new AuthResponse();
    authResponse.setSuccess(true);
    authResponse.setPrincipal(principal);
    return authResponse;
}
```

- If the value of the `success` field in the response is `true`, the gateway caches the `principal` information based on the cache policy, and then puts the `principal` information into the `header` of the request and transparently transmits it to the backend business system. If you do not have a principal, you must pass an empty Map.
- If the value of the `success` field in the response for verifying authorization is `false`, the gateway returns a 2000 error code. The client needs to perform corresponding operations as 2000, for example, a logon box appears.

Use authorization rules

After an authorization rule is configured, you can choose **Advanced Settings** > **API Authorization** on the API Configuration page to enable the authorization feature for the API.

To use API authorization, make sure that the **API authorization** feature is enabled on the **Manage gateway** page. Perform the following steps to enable the feature:

1. Log on to the mPaaS console. In the left-side navigation pane, click **Mobile Gateway Service**.
2. Click the **Manage gateway** tab and make sure that **API Authorization** is enabled.

This API performs authorization verification before requesting the backend system. If you pass, the request is accepted and the gateway routes the request to the backend system. Otherwise, the request will be rejected and the caller will receive an error response of authorization failure.

7.2.4. API traffic limit

API traffic limit allows you to configure traffic limit for a single API. You can also set the default traffic limit value for an API and set the total traffic limit value for an application to prevent the backend server from being overwhelmed during peak hours. If you specify both the default value of API traffic limit and the total value of app traffic limit, API traffic limit takes effect based on the traffic limit values.

This topic describes how to configure the default value of API traffic limit and the total value of application traffic limit. To configure traffic limit for a single API, you can configure traffic limit in the traffic limit configuration section on the API details page. For more information, see Configure an API.

Prerequisites

To use the traffic limit configuration, ensure that the traffic limit feature is enabled. Log on to the mPaaS console. In the left-side navigation pane, choose **Mobile Gateway Service** > **Manage gateway**. Turn on **Limit API traffic**.

Default value of API traffic limit

Set the default traffic limit value for an API to apply to all APIs in the current application. The default traffic limit configuration takes effect based on the following rules:

- If a traffic limit value has been configured for a single API, the traffic limit value of the API is subject to the previously configured traffic limit value.
- The traffic limit configuration of a single API overwrites the default traffic limit configuration of the API.
- The modified default traffic limit configuration takes effect for APIs that have previously used the default traffic limit configuration.

To configure a universal filter, perform the following steps:

1. Turn on **API default traffic limit** switch.
2. In the **Default traffic limit** dialog box, click **Edit** to configure the traffic limit settings.
   - **Default traffic limit**: Set a proper traffic limit threshold based on your business requirements. If this value is exceeded, the request is throttled.

     > **Note**
     > The traffic limit threshold refers to the maximum number of requests within one second.

   - **Traffic limit response**: The default response to traffic limit is `{"resultStatus":1002,"tips":"Too many customers, please wait"}`. To customize a response to traffic limit, use the following format:

     ```
     {
         "result": "==This is the custom response content. Enter==",
         "tips": "ok",
         "resultStatus": 1000,
     }
     ```

     In the preceding code:
     - `result` is customized response data, `JSON` format. The client uses this field for processing only when the `resultStatus` is 1000.
     - `tips` is a custom traffic limit prompt. If the `resultStatus` is 1002, this field will be used to prompt the user.
     - `resultStatus` is the result code returned for traffic limit. For more information, see Gateway result codes.

Total traffic limit of the App

Sets the sum value of traffic limit for all APIs under the current application. If the total traffic limit value of the application is exceeded, all API requests under the current application will be limited.

To configure a universal filter, perform the following steps:

1. Turn on **Total traffic limit of the App** switch.
2. In the Total App Threshold configuration box, click **Edit** to configure the traffic limit information.
   - **Total traffic limit**: Set a proper threshold based on your business requirements. Unit: seconds. If this value is exceeded, the request is throttled.
   - **Traffic limit response**: The default response to traffic limit is `{"resultStatus":1002,"tips":"Too many customers, please wait"}`. To customize a response to traffic limit, use the following format:

     ```
     {
         "result": "==This is the custom response content. Enter==",
         "tips": "ok",
         "resultStatus": 1000,
     }
     ```

     In the preceding code:
     - `result` for customized response data, `JSON` format. The client uses this field for processing only when the `resultStatus` is 1000.
     - `resultStatus` the result code returned for traffic limit. For more information, see Gateway result codes.
     - `tips` is a custom traffic limit prompt. If the `resultStatus` is 1002, this field will be used to prompt the user.

7.2.5. API Cache

Configure API cache information to cache API responses to reduce the pressure on the business system.

About this task

The API cache contains the response of the entire backend request, including the response header and response body. Therefore, the status class information, such as user data in the `cookie`, must be excluded from the response header of the cached API. This caching feature is only suitable for caching stateless data.

The backend service system can add a `Pragma: no-cache` to the response header to notify the gateway not to cache the response.

Procedure

1. Log on to the mPaaS console. In the left-side navigation pane, click **Mobile Gateway Service**.
2. In the API list, find the API that you want to configure and click **Configure** in the Actions column.
3. Click **Modify** in the **Cache Configuration** section and configure the following rules:
   - **Cache result**: Specifies whether to enable caching.
   - **Cache time**: The lifetime of the cache. Unit: seconds.
   - **Cache key**: The key-value expression used for caching. Click **Edit** to modify the cache key. In the displayed modal box, enter the primary key required for the cache. You can drag and drop them to sort. For information about key-value syntax, see Key-value syntax below.

Key-Value Syntax

Supported Syntax

When an API request arrives at the gateway, the gateway obtains the corresponding data as the cached key value according to the key-value configuration. The syntax is as follows:

Statement	Description
$	The root object, for example:$.bar

[num]	Array access, where num is a number. Example:$[0].bar.foos[1].name
.	Property access, for example:$.bar
['key']	Property access, for example:$['bar']
$.header	The API request header object, which is used to obtain the fields in the request header, for example:$.header.remote_addr
$.cookie	API request cookie object to get the value in the cookie, for example:$.cookie.session_id
$.http_body	The backend is an HTTP request body object, which is used to obtain the fields in the request body, for example, $.http_body.name
$.http_qs	The backend is an HTTP request parameters object that is used to obtain request parameters, for example, $.http_qs.name

Sample code

Take the following request data as an example and obtain the object from the request packet:

```
URL:/json.htm?tenantId=boo

Header:
Content-Type:application/json
opt:com.mobile.info.get
workspaceId:default
appId:B2D553102
cookie:JSESSIONID=abcd;traceId=trace1000

Body:
[
    {
        "key": "1234",
        "locations": [
            "beijing",
            "shanghai"
        ],
        "language": "zh-Hans",
        "unit": "c"
    },
    {
        "demo": {
            "name": "nick"
        }
    }
]
```

An example of the expression is shown below:

```
$.header.appId = B2D553102
$.cookie.traceId = trace1000
$.http_qs.tenantId = boo
$[0].key = 1234
$[0].locations[1] = shanghai
$[1].demo.name = nick
```

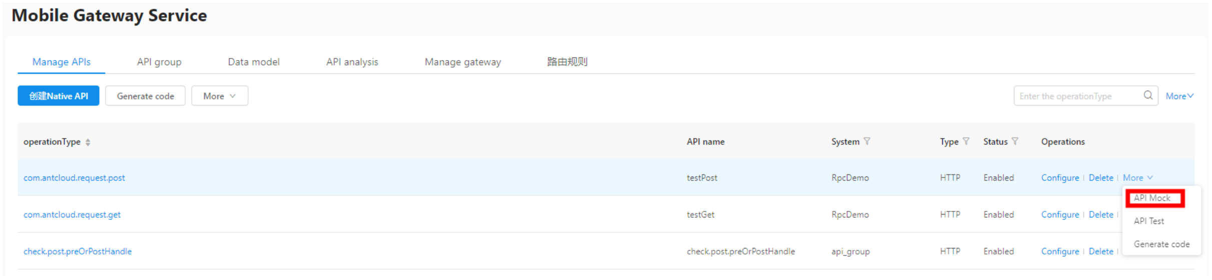
7.2.6. API Mock

API Mock is to simulate the return value of an API (mock) to provide a specific response result. To use the API mock feature, go to the **Manage gateway > Function Switch** page and turn on the **API Mock** switch.

Procedure

Complete these steps to configure API Mock:

1. Choose the **Manage APIs** tab> **More** > **API Mock** in the Operation column of the API list.



operationType	API name	System	Type	Status	Operations
com.antcloud.request.post	testPost	RpcDemo	HTTP	Enabled	Configure | Delete | More
com.antcloud.request.get	testGet	RpcDemo	HTTP	Enabled	Configure | Delete
check.post.preOrPostHandle	check.post.preOrPostHandle	api_group	HTTP	Enabled	Configure | Delete

2. On the **Mock Configuration** page, set the following parameters:
   - **API Mock**: Turn on API Mock.
   - **Hit Rule**: Percentage rule is supported.
   - **Rule Configuration**: The percentage value. Valid values: 0 to 100.
   - **Mock Data**: The response data of the Mock API. The Mock data format is as follows:

     ```
     {
     "resultStatus": 1000,
     "tips": "ok",
     "result": "==This is the business data of Mock. Enter=="
     }
     ```

     In the code:
     - The `resultStatus` is the response result code. For more information, see Gateway result codes.
     - `tips` is a response prompt.
     - The `result` is the customized response data in JSON format.
3. Click **Submit**.

7.2.7. Synchronize API

You can use this function to synchronize the APIs in the current workspace to other workspaces.

Note
- To ensure the system stability, only the API configuration that doesn't exist in the target workspace is synchronized.
- After synchronization, the system will automatically load the configuration and bring it into force.

Procedure

1. Log in to the mPaaS console, and from the navigation bar on the left, click **Mobile Gateway Service**.
2. On the **Manage APIs** tab, click **More** > **Sync API**.
3. In the **Basic information** area, select to synchronize the API configuration of the current workspace to the **Target workspace**.
4. Select the **Status after sync**:
   - **Unchanged**: Keep the same API configuration as that in the current workspace.
   - **All closed**: After synchronization, the API configuration is set as closed.
5. In the **Select API** area, select the API to be synchronized.
6. Click **OK** to start API synchronization.

Result

When the configuration is successfully synchronized, an Alert pops up on the current page to show the synchronization result.

7.2.8. Export and import API

To facilitate the application of the current API configuration to other environments or other applications, the Mobile Gateway Service supports exporting the API of the current application in the form of a .txt file. At the same time, you can also import the API configuration and apply it to the current environment. The following is a detailed introduction to the API export and import operations.

Export API

Select the API to be exported as needed, the API group and Data Model associated with API will be exported together.

Note
Only HTTP APIs can be exported.

The operation steps are as follows:

1. Log in to the mPaaS console, and from the navigation bar on the left, click **Mobile Gateway Service**.
2. On the **Manage APIs** tab, click **More** > **Export API**.
3. In the **Select API** area, select the API to be exported.
4. Click **OK** to start API exporting. The exported APIs are in a `.txt` file.

Import API

To ensure the stability of system, it is suggested that you select **Remain** as the import strategy.

The operation steps are as follows:

1. Log in to the mPaaS console, and from the navigation bar on the left, click **Mobile Gateway Service**.
2. On the **Manage APIs** tab, click **More** > **Import API**.
3. Confirm that the AppID and the current workspace are correct.
4. Select **Import strategy**: The strategy used when the imported configuration has conflict with the existing configuration.
   - **Keep existing configuration**: Keep the existing configuration and discard the imported configuration when there is a conflict.
   - **Replace with new configuration**: Replace the existing configuration with the imported configuration when there is a conflict.
5. Click **Upload file** and select the API file to upload.
6. Click **OK** to start API importing. When the configuration is successfully imported, the importing result is displayed on the page.

7.3. Call API

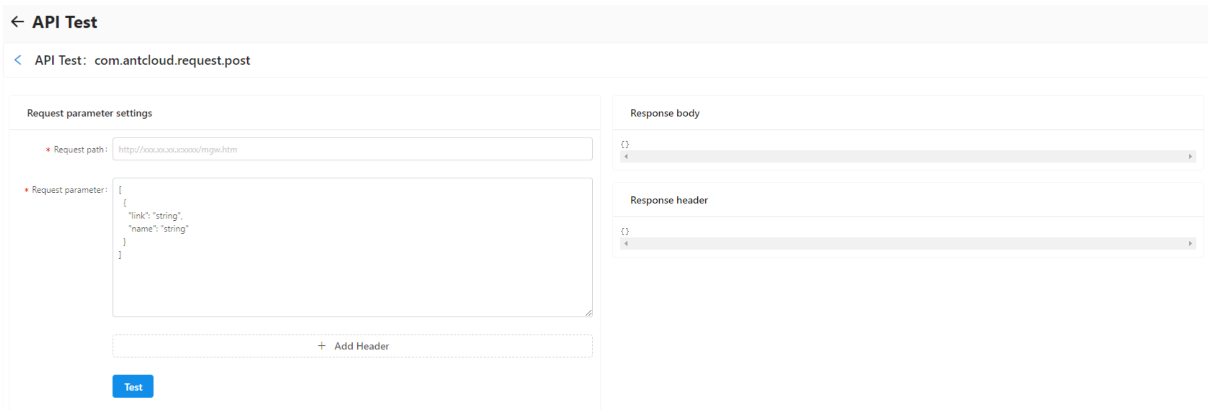
7.3.1. API test

In API testing, you can test whether the functionality of the current API is intact. To perform an API test, complete the configurations on the API Test page.

> **Important**
>
> The operationType of com.antcloud.session.validate is a dedicated interface of the MSS component. If the operationType of com.antcloud.session.validate is configured and gateway encryption is enabled, a 6004 error will be reported when testing this operationType in APITest.

Procedure

1. Choose **Manage APIs** > **More** > **API Test** in the Operation column of the API list to open the following page:


API Test
API Test: com.antcloud.request.post
Request parameter settings
* Request path: http://xxx.xx.xx.x:xxxx/mgw.htm
* Request parameter:
+ Add Header
Test
Response body
Response header

```
[
 {
  "link": "string",
  "name": "string"
 }
]
```

2. On the **API Test** page, configure the following:
   - **Request path**: Enter the gateway address of the application. For example, the public cloud address of the mPaaS mobile gateway is `https://cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm` .

     > **Important**
     >
     > The address must contain a `/mgw.htm` . Otherwise, the request fails.

   - **Request parameter**: By default, data in the request parameters format is simulated. Adjust the value based on the business meaning.
   - **Add Header**: You can add Request headers as needed.
3. Click **Test**. The following result appears:
   - **Response body**: the response data returned by the service.
   - **Response header**: includes the gateway conventions and response headers returned by the service.
     - Result-Status: For more information, see Gateway result codes.
     - Mgw-TraceId: the TraceId of the request, which can be used for link analysis.

7.3.2. Generate code

Mobile Gateway Service supports generating the client SDK for API.

About this task

Only the HTTP API supports code generation.

Procedure

1. Open the **Generate client code** window by doing any one of the following operations:
   - On the **API group** tab, click **Generate code** on the operation column.
   - On the **Manage API** tab, click **Generate code** on top of the list.
   - On the **Manage API** tab, click **More** > **Generate code** on the operation column. This method is used to generate single API codes (not API group), for HTTP APIs only.
2. In the **Generate client code** window, configure the following information:
   - **API group**: Select the API group which needs to generate SDK.
   - **Platform**: Select Android, iOS or JS, and configure relevant information accordingly.
     - If you select **Android**, you must enter the package name of the App in **PackageName**; if not filled, it defaults to `com.client.service` .
     - If you select **iOS**, you must enter the unique prefix in **Prefix**; if not filled, no prefix is attached by default.
3. Click **Submit** to generate API SDK for the invocation by client.

7.3.3. HTTP API request format

The MRPC protocol for connecting mPaaS client and MGS is a HTTP-based RPC protocol. When you call APIs on the API test page in MGS console or call the APIs by using non-mPaaS clients such as Postman, you can construct the RPC requests with reference to the following instruction.

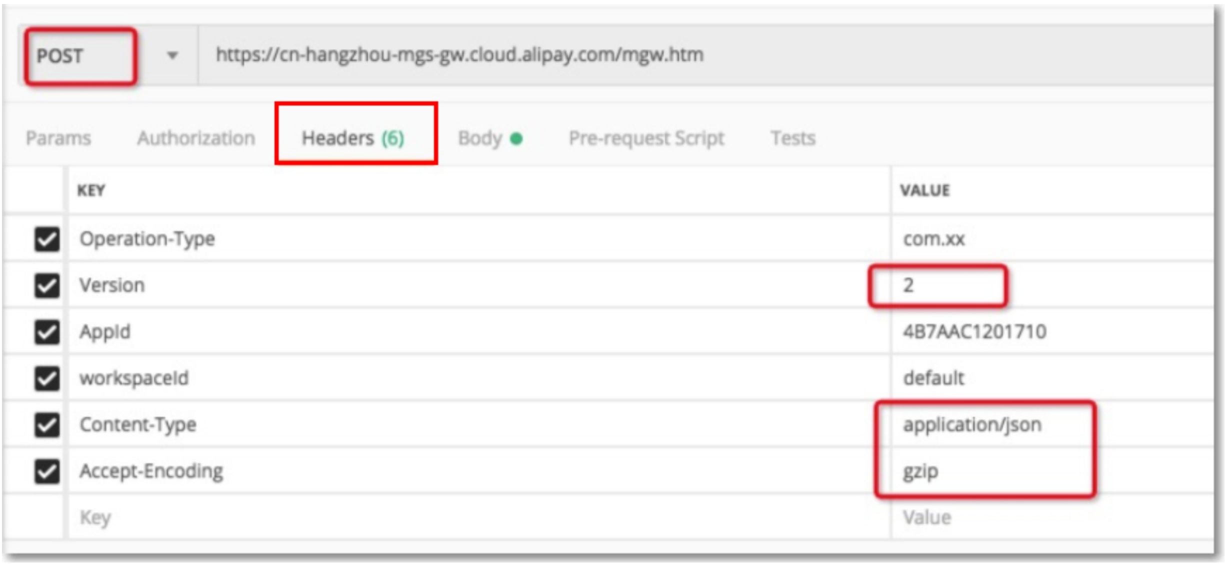
API test page

In case of calling an API on the API test page, the request parameters should be in `[{}]` format. If the API back-end service is HTTP service, the query parameters to be delivered to the HTTP `get` request should be input in `[{}]`, and the body parameters to be delivered to the HTTP `post` request should be put in the value of the key `_requestBody`.

Non-mPaaS client such as Postman

> **Note**
> When you directly call APIs through Postman and other non-mPaaS clients, you must turn the **Signature verification** and **Data encryption** function switch off in the MGS console in advance, otherwise a request failure may be prompted.

In the figure below, the parameters marked in the red box are fixed, the rest can be replaced with specific API parameters. Note that if the back-end HTTP service is of `get` type, the `post` here is still fixed and cannot be changed. You just need to change the method to `get` on the **API details** page > **Basic information** area in the MGS console.



KEY	VALUE
Operation-Type	com.xx
Version	2
AppId	4B7AAC1201710
workspaceId	default
Content-Type	application/json
Accept-Encoding	gzip
Key	Value

The Body parameters in the request should be in `[{}]` format. If the API back-end service is HTTP service, the query parameters to be delivered to the HTTP `get` request should be input in `[{}]`, and the body parameters to be delivered to the HTTP `post` request should be put in the value of the key `_requestBody`.

7.4. Manage gateway

7.4.1. Introduction to gateway management

Gateway management includes the following operations:

Gateway management includes the following operations:

Function switch

The function switch works globally. You can temporarily enable or disable all API related functions on demand.

Signature verification

Implement signature verification on the requests from client to mobile gateway to verify the callers' identity to ensure safety. It is **On** by default.

Limit API traffic

Limit the access volume of a certain API to prevent the background server from crash in peak hours. It is **Off** by default.

API Mock

Mock the returned value of a certain API to provide specific response. It is **Off** by default.

API authorization

Verify the legality of the client request before MGS routing the request to the backend business system, if the verification passes, the request is approved to proceed. This function defaults to OFF.

See API authorization for more information.

Data encryption

Encrypt the requests from client to mobile gateway to ensure the data security during transmission. It is **Off** by default. Currently, the supported encryption algorithms are ECC and RSA. This function must be used in combination with the client. If the data encryption method you set here differs from the client's, the gateway might fail to parse the requests from the client.

For specific configuration, see Encrypt data.

CORS

CORS (Cross-Origin Resource Sharing) controls the cross-origin access as per the rules. If cross-origin request is required, please configure this rule.

See Cross-Origin Resource Sharing for more information.

Customize result code

Gateway result codes have default prompt text. You can also customize the result code prompts based on actual requirement.

On the **Manage gateway** tab page, click **Customize result code** to go to the customization page.

Operation records

Record and display configuration staff's operations on the gateway, thus making it convenient for the customers to trace back.

Common tools

Trace analysis can analyze `TraceId` and parse the corresponding time and gateway server.

7.4.2. Data encryption

To encrypt data, on the server side, you need to perform relevant configurations to generate keys; on the client side, complete corresponding configurations according to different operating platforms.

Server

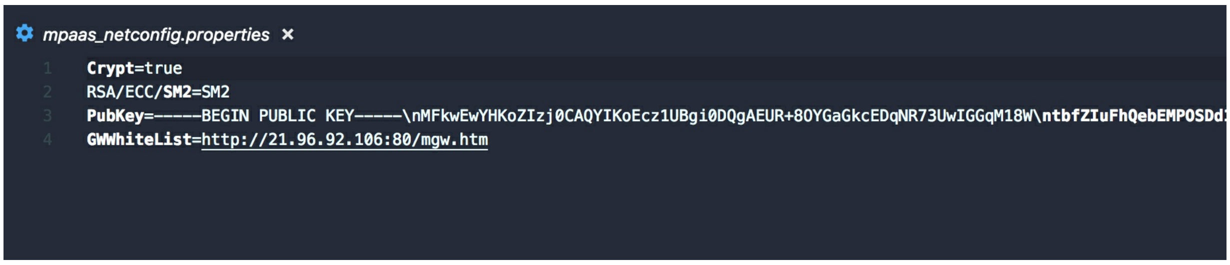
1. Log on to the mPaaS console. In the left-side navigation pane, click **Mobile Gateway Service**.
2. Click the **Manage gateway** tab. On the **Manage gateway** tab, click the **Function switch** tab.
3. Switch the status of **Data encryption** to **On**.
4. In the **Configure encryption algorithm** dialog box that appears, configure the following settings:
   - Encryption algorithm: ECC, RSA, and SM2 are supported.
   - Key pair:
     - If the encryption algorithm is set to ECC or SSM, enter the private key content.
     - If the encryption algorithm is RSA, enter the public and private keys respectively.

   For more information about how to generate a key for an encryption algorithm, see How to generate a key.

Client configuration

Android configuration

Create a `mpaas_netconfig.properties` file in the `assets` directory to store network-related global configurations.



mpaas_netconfig.properties

```
Crypt=true
RSA/ECC/SM2=SM2
PubKey=-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEUR+8OYGaGkcEDqNR73UwIGGqM18W\ntbfZIuFhQebEMPOSDd
GWWhiteList=http://21.96.92.106:80/mgw.htm
```

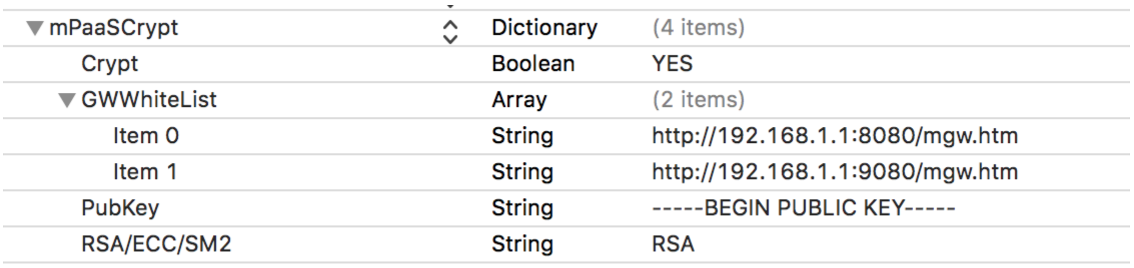
- `Crypt` : Indicates whether to use self-encryption, `true` indicates to use, `false` indicates to disable self-encryption function.
- `RSA/ECC/SM2` : Indicates the asymmetric encryption algorithm to use, whose value can only be filled with `RSA` or `ECC` or `SM2` .
- `PubKey` : Indicates the public key of the selected asymmetric encryption algorithm.

> **Note**
>
> Since the `value` values of the `properties` files in Android need to be on the same line, you need to be aware of using line breaks `\n` convert the `Pubkey` to one line when populating the public key.

- `GWWhiteList` : The gateway that needs to be encrypted is the gateway address of the current environment (the `rpcGW` field in the configuration file obtained from the mPaaS console). Without this key, all requests will not be encrypted.

iOS configuration

The iOS encryption configuration is read from the `info.plist` , as shown in the following figure:



▼ mPaaSCrypt	Dictionary	(4 items)
Crypt	Boolean	YES
▼ GWWhiteList	Array	(2 items)
Item 0	String	http://192.168.1.1:8080/mgw.htm
Item 1	String	http://192.168.1.1:9080/mgw.htm
PubKey	String	-----BEGIN PUBLIC KEY-----
RSA/ECC/SM2	String	RSA

- `mPaaSCrypt` : The main key and value of the encryption configuration are `Dictionary` types, which contain relevant information required for client encryption.
- `Crypt` : specifies whether to encrypt data. The value is `Boolean` type. `YES` indicates that data is encrypted. `NO` indicates that data is not encrypted.
  - When Crypt is set to `NO` , RPC does not encrypt and `RSA/ECC/SM2` and `PubKey` settings are ignored.
  - When Crypt is set to `YES` , `RSA/ECC/SM2` and `PubKey` must be set and cannot be empty string, otherwise it will be asserted in Debug and the program will exit directly.
- `GWWhiteList` : The gateway that needs to be encrypted is the gateway address of the current environment (the `rpcGW` field in the configuration file obtained from the mPaaS console). Without this key, all requests will not be encrypted.
- `RSA/ECC/SM2` : asymmetric encryption algorithm selection. The value is of the `String` type and can only be `RSA` or `ECC` or `SM2` . The `RSA/ECC/SM2` and `PubKey` settings must correspond to each other.

  - Select a `RSA` algorithm and enter `RSA` public key in PubKey.
  - Select a `ECC` algorithm and enter `ECC` public key in PubKey.
  - Select a `SM2` algorithm and enter `SM2` public key in PubKey.
- `PubKey` : Asymmetric encryption the public key. The value is `String` type, consistent with the asymmetric encryption algorithm chosen.

  The PubKey format must include the `-----BEGIN PUBLIC KEY-----` and `-----END PUBLIC KEY-----` . The format is as follows:

```
-----BEGIN PUBLIC KEY-----
 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0YTfXiICxPUaDHg7Wlxc
 bzN1UsGfDBHOyn4JYqZq8ySIBa+F9Uuyk0w+Ft/8sQE8MXSnJEqOAcUtG7Y0Js8L
 lDsDi0Dd+e9Zpq+WHp4+cM8GAujTy/hSHjuZPYbovtjTXp9iFo9Mxz3SbllvQ0d3
 VOpbks986gET/rchAlu9L+6oLf+HsiyYSAXQfYD4GI7sjtqYoRiSA6bWw1m+uFDc
 j1iHwW3HA11LsHDkQlLoNgXhvKoy+H7yM6t94ZhvXdgFK2yd5wq6FKIuZmgqiEg9
 A8S3/aUMKRIlVRvfkfcM+sBxiVgr80s6VTojfq/b2I3xKqnJ4KZMStpJHvsxWfw7
 2wIDAQAB
 -----END PUBLIC KEY-----
```

7.4.3. Cross-origin resource sharing (CORS)

Cross-domain access refers to a request for a resource with a different source (different domain name, protocol, or port) from its own resource. The different sources can be different domain names, protocols, or ports.

CORS

Cross-domain access refers to a request for a resource with a different source (different domain name, protocol, or port) from its own resource. The different sources can be different domain names, protocols, or ports.

For security reasons, the browser sets a same-origin policy to restrict cross-domain requests from within the script. However, in practical applications, cross-domain access often occurs. To this end , the W3C provides a standard cross-domain solution, cross-domain Resource Sharing (Cross-Origin Resource Sharing,CORS), to support secure cross-domain requests and Data Transmission Service.

Browsers divide CORS requests into the following two categories:

- Simple requests
- Precheck request: a protection mechanism that prevents resources from being modified by requests that are not originally authorized. The browser sends a preflight request using the `OPTIONS` method before sending the actual request to know whether the server allows the cross-domain request. The actual HTTP request is initiated only after the server confirms the permission.

Simple requests

A request is a simple request if all of the following conditions are met:

- The request method is one of the following:
  - `HEAD`
  - `GET`
  - `POST`
- The HTTP header information cannot exceed the following fields:
  - `Cache-Control`
  - `Content-Language`
  - `Content-Type`
  - `Expires`
  - `Last-Modified`
  - `Pragma`
  - `DPR`
  - `Downlink`
  - `Save-Data`
  - `Viewport-Width`
  - `Width`
- The `Content-Type` values are limited to the following:
  - `text/plain`
  - `multipart/form-data`
  - `application/x-www-form-urlencoded`

Precheck Request

If a request does not meet the simple request conditions, a `OPTIONS` request is triggered for precheck before formal communication. This type of request is a preflight request.

A precheck request includes the following information in the request header:

- `Origin` : Request source information.
- `Access-Control-Request-Method` : the type of the next request, such as POST or GET.
- `Access-Control-Request-Headers` : the list of headers explicitly set by the user to be included in the next request.

After the server receives the precheck request, it determines whether to allow the cross-domain based on the preceding information and returns the corresponding information through the response header:

- `Access-Control-Allow-Origin` : List of origins allowed for cross-domain.
- `Access-Control-Allow-Methods` : List of methods allowed to be cross-domain.

- `Access-Control-Allow-Headers` : List of headers that can be cross-domain.
- `Access-Control-Expose-Headers` : List of headers that can be exposed.
- `Access-Control-Max-Age` : Maximum browser cache time. Unit: seconds.
- `Access-Control-Allow-Credentials` : Whether sending cookies is allowed.

The browser determines whether to continue sending the real request based on the returned CORS information. The preceding actions are automatically performed by the browser. You only need to configure specific CORS rules on the server.

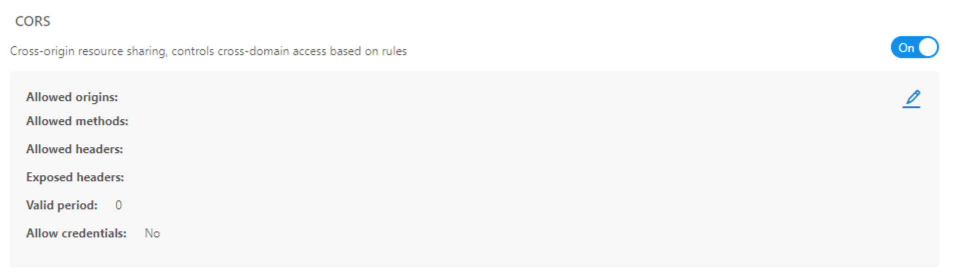
Gateway support for CORS

The gateway provides the function of configuring CORS rules so that the business party can decide whether to allow specific cross-domain requests. This rule is configured in the appId + workspaceId dimension.

Configure CORS

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

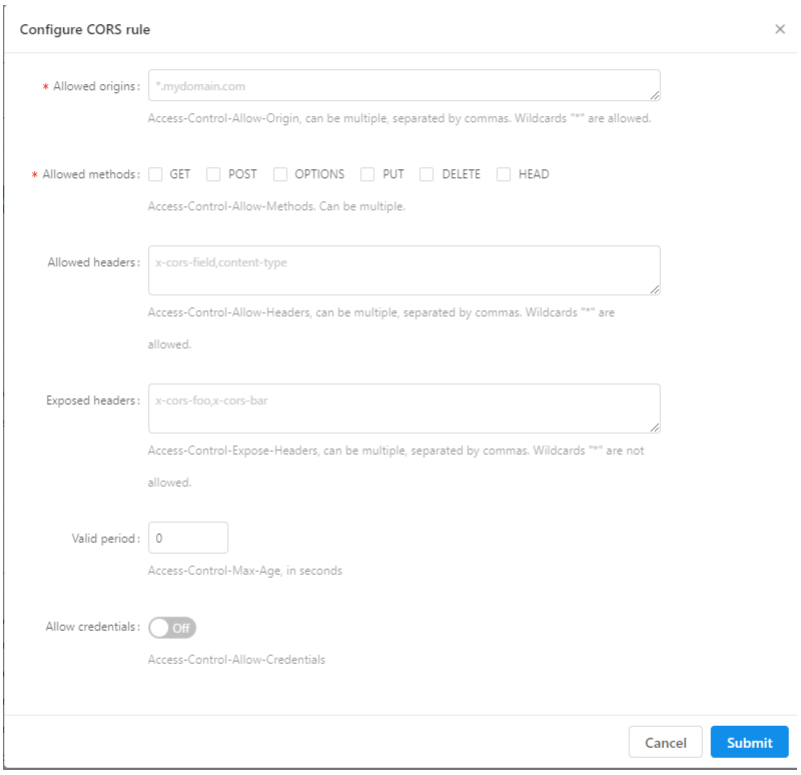
1. In the left-side navigation pane, click **Mobile Gateway Service**.
2. Click the **Manage gateway** tab. On the Gateways tab, click the **Function switch** tab to configure CORS.


CORS

Cross-origin resource sharing, controls cross-domain access based on rules

On

Allowed origins:
Allowed methods:
Allowed headers:
Exposed headers:
Valid period: 0
Allow credentials: No


Configure CORS rule

* Allowed origins: *.mydomain.com
Access-Control-Allow-Origin, can be multiple, separated by commas. Wildcards "*" are allowed.

* Allowed methods: GET POST OPTIONS PUT DELETE HEAD
Access-Control-Allow-Methods. Can be multiple.

Allowed headers: x-cors-field,content-type
Access-Control-Allow-Headers, can be multiple, separated by commas. Wildcards "*" are allowed.

Exposed headers: x-cors-foo,x-cors-bar
Access-Control-Expose-Headers, can be multiple, separated by commas. Wildcards "*" are not allowed.

Valid period: 0
Access-Control-Max-Age, in seconds

Allow credentials: Off
Access-Control-Allow-Credentials

Cancel Submit

After CORS is enabled, all API services of the app in the workspace will support cross-domain requests that meet the following configurations:

- Allowed origins: `Access-Control-Allow-Origin` . Multiple sources can be specified. Separate multiple sources with commas (,). Wildcards are allowed.
- Allowed methods: `Access-Control-Allow-Methods` . You can select multiple methods.
- Allow headers: `Access-Control-Allow-Headers` , you can set multiple, comma separated, allowing '*' wildcard.
- Exposed headers: `Access-Control-Expose-Headers` : Multiple can be set, comma separated, '*' wildcard is not allowed.
- Valid period: `Access-Control-Max-Age` . The maximum browser cache time. Unit: seconds.
- Allow credentials: `Access-Control-Allow-Credentials` , whether the Cookie can be sent.

Cross-domain Request

Add the `X-CORS-${appId}-${workspaceId}` request header to the cross-domain API request. After the precheck request arrives at the gateway, the gateway parses the `X-CORS-${appId}-${workspaceId}` in the `Access-Control-Request-Headers` to obtain the appId and workspaceId, and then obtains the corresponding CORS configuration. The request header of a gateway cross-domain request must contain the following content:

- X-CORS-\${AppId}-\${WorskapceId}: Be sure to include this request header and replace the placeholder content with the actual AppId and WorkspaceId;
- Operation-Type
- WorkspaceId
- AppId
- Content-Type
- Version

```
$.ajax({
    url: 'http://${mpaasgw_host}/mgw.htm',// Enter the gateway address.
    headers: {
      'X-CORS-${appId}-${workspaceId}':'1' // Be sure to set this request header.
      'Operation-Type':${operationType}, // Specify operationType.
      'AppId':${appId}, // Enter an appId.
      'WorkspaceId':${worksapceId}, // Enter worksapceId.
      'Content-Type':'application/json',
      'Version':'2.0',
    },
    type: 'POST',
    dataType: 'json',
    data: JSON.stringify(reqData),
    success: function(data){}
  });
```

> **Note**
> The **Allowed headers** parameter in the CORS configuration. You can add or set an asterisk (*) based on your business needs.

7.5. Data model

As a business staff, you can define API service's requests and response as **data models**, and reuse the models to reduce fussy parameter settings. This function is used to define the parameters of HTTP API service. For other types of API services, you don't have to define manually.

About this task

Currently, Mobile Gateway Service supports the following data model definition methods:

- Visual edition: Add model parameters item by item.
- Sample data edition: (Recommended) Parse the data model from the sample data.

Procedure

Complete the following steps to configure data model:

1. On the Mobile Gateway Service homepage, select **Data model** tab to go to the data model list page.
2. Click **Create data model** to add a new data model, or click **Details** right to a specific data model in the list to edit the data model:
    - **Model name**: The name of data model, which comprises letters, underscores, and numbers; and starts with a letter or an underscore.
    - **Model description**: Description of the data model.
    - **Model parameters**:
        - **Parameter name**: Required, the name of the parameter in a data model.
        - **Type**: Required, selectable types include `String` , `Int` , `Long` , `Float` , `Double` , `Boolean` , `List` and the data model that is already defined.
        - **Default value**: Optional, the default value of the parameter.
        - **Description**: Optional, the description of the parameter.

    > **Note**
    > `Map` type is not supported currently.

3. Click **Submit** to save the changes.

8.Gateway exception troubleshooting

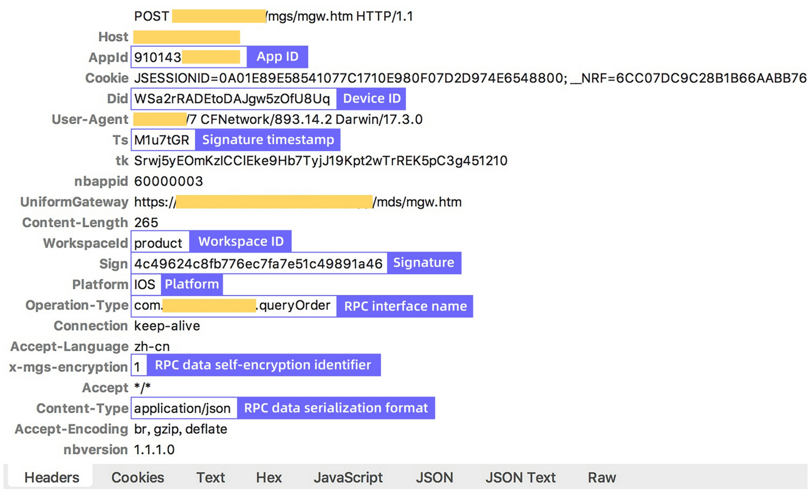
Single request troubleshooting

1. Capture client-side request packets

Generally, Charles (recommended) or Fiddler tool is used to capture client-side packets. With the packet capture tool, you can find some critical data of the RPC requests.

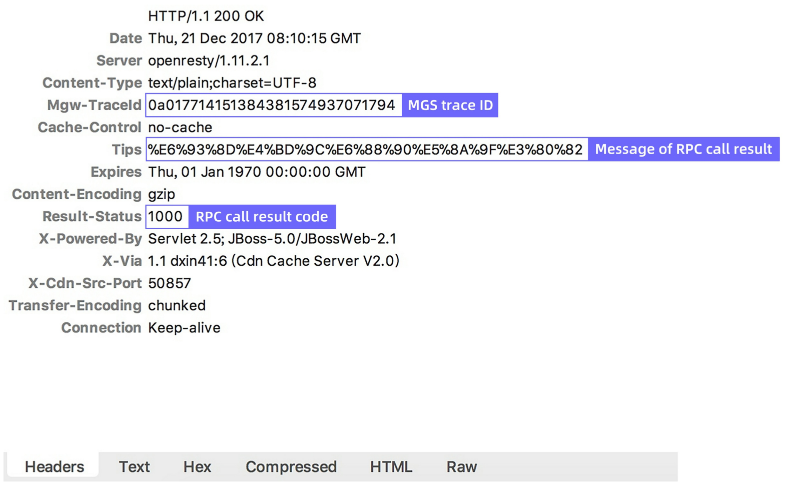
Here is an example of packet capture:

- Example of request header:



```
                POST [redacted]/mgs/mgw.htm HTTP/1.1
                Host [redacted]
               AppId 910143[redacted]
              Cookie JSESSIONID=0A01E89E58541077C1710E980F07D2D974E6548800; _NRF=6CC07DC9C28B1B66AABB76
                 Did WSa2rRADEtoDAJgw5zOfU8Uq
          User-Agent [redacted]/7 CFNetwork/893.14.2 Darwin/17.3.0
                  Ts M1u7tGR
                  tk Srwj5yEOmKzlCClEke9Hb7TyjJ19Kpt2wTrREK5pC3g451210
             nbappid 60000003
      UniformGateway https://[redacted]/mds/mgw.htm
      Content-Length 265
         WorkspaceId product
                Sign 4c49624c8fb776ec7fa7e51c49891a46
            Platform IOS
      Operation-Type com.[redacted].queryOrder
          Connection keep-alive
     Accept-Language zh-cn
    x-mgs-encryption 1
              Accept */*
        Content-Type application/json
     Accept-Encoding br, gzip, deflate
           nbversion 1.1.1.0
```

- Example of response header:



```
                     HTTP/1.1 200 OK
                Date Thu, 21 Dec 2017 08:10:15 GMT
              Server openresty/1.11.2.1
        Content-Type text/plain;charset=UTF-8
        Mgw-TraceId 0a017714151384381574937071794
       Cache-Control no-cache
                Tips %E6%93%8D%E4%BD%9C%E6%88%90%E5%8A%9F%E3%80%82
             Expires Thu, 01 Jan 1970 00:00:00 GMT
    Content-Encoding gzip
       Result-Status 1000
        X-Powered-By Servlet 2.5; JBoss-5.0/JBossWeb-2.1
               X-Via 1.1 dxin41:6 (Cdn Cache Server V2.0)
       X-Cdn-Src-Port 50857
    Transfer-Encoding chunked
          Connection Keep-alive
```

2. Query MGS log by TraceId (for private cloud only)

1. Obtain Mgw-TraceId from the response header.
2. In mPaaS console, select the target App, go to the **Mobile Gateway** > **Gateway management**> **Tools** > **Trace analysis** page, and enter the TraceId to parse the corresponding MGS server IP and processing time of the request.
3. Connect MGS server through SSH, and then query the request-related logs by TraceId.

```
ssh -p2022 account@IP account/password
cd /home/admin/logs/gateway
grep #traceid# *.log
```

4. Analyze logs according to the Gateway logs and Gateway result codes.

Cluster GREP troubleshooting (for private cloud only)

Sometimes, you may need to search a certain log in MGS cluster. At this time, you can use the open-source PSSH tool.

1. Download PSSH tool.
2. Export the server IP list of MGS from Gamma platform to `mgs_host.txt` file, for example:

```
log@10.2.216.33:2022
log@10.2.216.26:2022
log@10.2.216.25:2022
```

3. Run the following command:

```
pssh -i -h mgs_host.txt -A -P 'grep "xxxx" /home/admin/logs/gateway/xxx.log'
```

9.FAQ

Is there a limit on the size of gateway request data packets?

The client will limit the request/response size to a maximum of 200k; it is recommended not to exceed 2k. The gateway is primarily used for transmitting data via RPC, not for transmitting images or videos. Requests and responses exceeding 2k will impact performance and stability. If file uploading and downloading are involved, it is recommended to use services that support big data channels, such as OSS.

How to troubleshoot in case of a call failure?

See Gateway exception troubleshooting.

What are the meanings of the result codes returned by APIs?

See Gateway result codes.

When OkHttp is referrenced, how to deal with the conflict between OKio and mPaaS?

To solve the conflict, perform the following steps:

1. Comment out the wire component of mPaaS.

```
mpaascomponents{
   excludeDependencies=['com.alipay.android.phone.thirdparty:wire-build']
}
```

2. Use the wire component provided on the Internet.

```
implementation  'com.squareup.wire:wire-lite-runtime:1.5.3.4@jar'
```

How to put parameters in POST body when sending a POST request by calling the MGS RPC interface through JSAPI?

First, you must correctly configure the POST body and corresponding data model for MGS. When sending a request through JSAPI, you need to take the POST body as the value of `_requestBody` and put it in the `requestData` parameter, as shown in the following sample:

```
window.onload = function() {
     ready(function() {
       window.AlipayJSBridge.call('rpc', {
         operationType: 'MYAPI',
         requestData: [
         {"_requestBody":"{\"key1\":\"value1\",\"key2\":\"value2\"}"}],
         headers:{},
         getResponse: true
       }, function(data) {
         alert(JSON.stringify(data));
       });
     });
   }
```

10.Reference

10.1. Gateway result code description

This topic describes the result codes that appear when you use Mobile Gateway Service.

Result code on the gateway side

- 1000 indicates that the API call is successful, while others indicate failure.
- 1001-5999 and 7XXX indicate gateway errors.
  - 7XXX indicates that the security guard reports an error during signature verification or decryption. For more information, see Wireless bodyguard result code description.
  - In addition to the result code, you can view the `Memo` and `tips` fields in the response header for more error information.
  - Public cloud users can also view detailed error information through the `~/logs/gateway/gateway-error.log` logs on the gateway server.
- If an exception occurs, you can try to troubleshoot the error by using the gateway exception troubleshooting. For more information, see Gateway exception troubleshooting.

Response code	Description	Explanation
1000	Processing succeeded	The gateway API call is successfully processed.
1001	Access is denied.	The Mock format is wrong, the `resultStatus` is missing, WAF verification fails, or the user of the authentication interface is not authorized to access.
1002	The number of calls exceeds the limit.	This exception occurs when traffic limit is triggered after you turn on Traffic limit configuration.
1005	No permissions	After you enable API authorization, the authorization verification fails when you call the API.
2000	Your logon session has timed out.	If the authorization verification feature is enabled, this exception is triggered if the logon status is not enabled.
3000	RPC interface does not exist or is down	In the environment corresponding to the current workspaceId, the mobile application corresponding to the appId does not have an API service configured with the operationType, or the API service is not in the **Open** state.
3001	The request data is empty.	The `requestData` in the client request data is empty. Check whether the client RPC service is normal. Make sure that the gateway service is initialized on the iOS client.
3002	Wrong data format	There is a problem with the RPC request format. Public cloud users can view details in the server log `gateway-error.log`.
3003	Data decryption failed	Data decryption failed.
4001	Service request timeout	MGS calls the business system service timeout. If the backend business system is overloaded, you need to check the running status of the backend system. If the timeout setting is unreasonable, you can adjust it appropriately. Note: The default timeout period is 3s.
4002	Remote call service system exception	An exception occurred when MGS called the business system service. Public cloud users can view details in the server log `gateway-error.log`.
4003	API group HOST exception	An `UnknownHostException` exception occurred when MGS called the HTTP business system service. Check whether the domain name configured for the API group exists.
5000	An unknown error occurred.	Other serious errors. Public cloud users can view details in the server log `gateway-error.log`.
7000	No public key set	The security guard in the mobile APP does not have the key corresponding to the appId or the gateway cannot obtain the signature key corresponding to the appId.
7001	Insufficient parameters for signature verification	The gateway server fails to verify the signature.
7002	Signature verification failed	The gateway server fails to verify the signature.

7003	Signature verification-timeliness failed	The ts timestamp of the API request input parameter exceeds the time validity set by the system. You need to check whether the client time is the system time.
7007	Check signature-the ts parameter is missing	The API request is missing the signature verification ts parameter.
7014	Sign verification-the sign parameter is missing	The API request does not have the sign parameter. In most cases, the client fails to sign data. As a result, the sign parameter is missing. Please check the client security guard picture is correct.
8002	Cross-domain precheck requests (CORS preflight)	Cross-domain precheck requests.

Business-side result code

The following result code allows you to view the error message inside the business system server.

You can view the `~/logs/mobileservice/monitor.log` logs on each business system to determine the exception details.

Response code	Applicable Agreement	Description	Explanation
6000	MPC, DUBBO	RPC-Target service not found	The published service cannot be found, the server cannot be accessed, or the service has been migrated.
6001	MPC, DUBBO	RPC-Target method not found	The method in the published service cannot be found.
6002	MPC, DUBBO	RPC-Incorrect number of parameters	The number of input parameters, which is not equal to the number of declared parameters.
6003	MPC, DUBBO	RPC-Target method not accessible	The target method cannot be called.
6004	HTTP, MPC, and DUBBO	RPC-JSON parsing exception	HTTP: An exception occurred while converting the RPC parameter to a backend HTTP request parameters. MPC/DUBBO: Failed to deserialize the RPC JSON-formatted data into a business parameters object.
6005	MPC, DUBBO	RPC-invalid parameters when calling target method	Parameters are invalid when you call a reflection operation.
6007	MPC, DUBBO	RPC-Authentication logon service unavailable	The authentication login port is not implemented in the SPI package or the authentication login port configuration is incorrect.
6666	HTTP, MPC, and DUBBO	RPC-Business throws exception	HTTP: The HTTP status code returned by the backend system is not equal to 200. MPC/DUBBO: exceptions thrown by the business side. RPC cannot be handled, unified as a business exception.

Android client result code

Response code	Description	Prompt copy
0	Unknown error.	Unknown error. Please try again later.
1	Client cannot find communication object, Transport not set	A network error occurred. Please try again later.
2	The client has no network, such as the user has disabled the network or disabled the network permissions of the application	Network cannot connect
3	SSL-related errors, including SSL handshake errors and SSL certificate errors	The client certificate is incorrect. Check whether the time setting of the mobile phone is accurate.
4	Client network connection timeout, TCP connection timeout, the current timeout period is 10s	Poor network
5	Scenarios where the network speed of the client is too slow, data read and write times out, and socketTimeout	Poor network

6	The client requests no response from the server, NoHttpResponseException	A network error occurred. Please try again later.
7	Client network IO error, corresponding to IOException	A network error occurred. Please try again later.
8	Client network request scheduling error, execution thread interruption exception	A network error occurred. Please try again later.
9	Client processing errors, including serialization errors, annotation processing errors, and thread execution errors	A network error occurred. Please try again later.
10	Client data deserialization error, server data format error	A network error occurred. Please try again later.
13	Request interruption error, such as network request will be interrupted when thread is interrupted	A network error occurred. Please try again later.
15	Client network authorization error, HttpHostConnectException,Connection to xxx refused, no network or connection refused by corresponding server	Network cannot connect
16	DNS resolution error	The network cannot be connected. Please try again later.
18	Network traffic limit, client traffic limit, network requests are throttled when the client request traffic exceeds the threshold	Network traffic limit. Please try again later.
Code ≧ 400 and code < 500	HTTP response code is 4xx	Network cannot connect
400 > code≧ 100 and 500 < code < 600	HTTP non-successful response codes	The network cannot be connected. Please try again later.

iOS RPC request returns error code

Value	Status code	Meaning
0	kDTRpcNetworkError	The network cannot be connected. **Important** All network failures will be classified here, and the specific failure error will be transmitted through userinfo. The key value of the corresponding error in the userinfo dictionary is kDTRpcErrorCauseError.
1	kDTRpcEmptyResponse	The data returned by the server is empty.
2	kDTRpcInvalidJSONString	The JSON string returned by the server is not in the correct format and cannot be successfully converted into a JSON object.
3	kDTRpcDecodeObjectError	Error of deserializing JSON object.
4	kDTRpcNetworkCancelled	The network has been cancelled.
5	kDTRpcEncodeObjectError	Error of serializing JSON object.
6	kDTRpcProtocolBuffersDecodeError	Error of deserializing PB object.
9	KDTRpcSizeControlError	If the RPC is too large, an exception will be thrown directly (this error is delegated to the caller) `[DTRpcInterface rpcSizeControl:size:isReq:NO]` .
24	KDTRpcAbandonError	When switching accounts, the returned RPC is discarded after the login RPC, and an exception is thrown directly.
3003	/	Error of decryption.

10.2. Wireless bodyguard result code description

The wireless bodyguard error codes listed in this article apply to both Android and iOS operating systems.

Error codes are classified into the following three types based on different error types:

- Common error codes
- Error codes for static data encryption and decryption

- Error codes of secure signature operations

If an error occurs, an error code in the 'SG ERROR: xxxx' format is displayed in the console of Xcode.

General error codes

Error code	Description
101	The error code returned because the parameters are invalid. Check the parameters.
102	Initialization of the main plug-in failed.
103	Plug-ins with dependencies are not introduced. When this error code is printed, you will be prompted with the name of the missing plug-in. Please perform the operation as prompted.
104	Plugin was introduced but failed to load. Generally, it is because there is nc `-all_load` or `-ObjC` added to the other linker flags, which can be solved after addition.
105	The corresponding plug-in is introduced, but the dependent plug-ins of the plug-in are not introduced. When this error code is printed, you will be prompted with the name of the missing plug-in. Please perform the operation as prompted.
106	The corresponding plug-in is introduced, but the version of the dependent plug-in of the plug-in does not meet the requirements. When this error code is printed, the version number of the dependency is displayed. Follow the instructions.
107	The corresponding plug-in is introduced, but the version of the plug-in does not meet the requirements.
108	The corresponding plug-in is introduced, but the dependent resources of the plug-in are not introduced.
109	A corresponding plug-in is introduced, but the version of the dependent resource of the plug-in does not meet the requirements.
121	The error code returned because the image file is invalid. Generally, the bundle id used to generate the image file is inconsistent with the bundle id of the current application.
122	No image file found. Make sure that the image file is in the project directory.
123	There is a problem with the image file format. Please regenerate the image file.
124	The version of the current image is too low.
125	init with authcode initialization error.
199	The error code returned because an unknown error occurred. Try again later.
201	The error code returned because the parameters are invalid. Check the parameters.
202	The error code returned because the image file is invalid. Generally, the bundle id used to generate the image file is inconsistent with the bundle id of the current application.
203	No image file found. Make sure that the image file is in the project directory.
204	There is a problem with the image file format. Please regenerate the image file.
205	The content of the image file is incorrect. Please regenerate the image file.
206	The key in the parameter cannot be found in the image file. Make sure that the image file contains this key.
207	The entered key is invalid.
208	Insufficient memory, please try again.
209	The key of the specified index does not exist.
212	Please upgrade the new version of the image. The version of the current image is too low.
299	The error code returned because an unknown error occurred. Try again later.

Static data encryption and decryption error codes

Error code	Description
301	The error code returned because the parameters are invalid. Check the parameters.
302	The error code returned because the image file is invalid. Generally, the apk signature of the image file obtained is inconsistent with the apk signature of the current program. Please use the apk of the current program to regenerate the image.
303	No image file found. Make sure that the image file is in the `res\drawable` directory.
304	If the format of the image file is incorrect, generate another image file. For more information, seeGenerate a wireless bodyguard image. A common scenario is the mixing of two-party and three-party images. The two-party and three-party images are incompatible and need to be generated separately.
305	The content in the image file is incorrect. Please regenerate the image file.
306	The key in the parameter cannot be found in the image file. Make sure that the image file contains this key.
307	The entered key is invalid.
308	Insufficient memory, please try again.
309	The key of the specified index does not exist.
310	The data to be decrypted is not decryptable data.
311	The data to be decrypted does not match the key.
312	The current image version is too low. Upgrade the image to a new version. For more information about how to generate an image, see Generate a wireless bodyguard image.
399	The error code returned because an unknown error occurred. Try again later.
401	The error code returned because the parameters are invalid. Check the parameters.
402	Insufficient memory, please try again.
403	Failed to obtain system properties. Check whether there is any software to intercept and obtain system parameters.
404	Failed to obtain the key of the image file. Check whether the format and content of the image file are correct.
405	Failed to obtain the dynamic encryption key. Try again.
406	The data format to be decrypted does not meet the decryption requirements.
407	The data to be decrypted does not meet the decryption requirements. Please confirm that the data is dynamically encrypted by the bodyguard on this device.
499	The error code returned because an unknown error occurred. Try again later.
501	The error code returned because the parameters are invalid. Check the parameters.
502	Insufficient memory, please try again.
503	Failed to obtain system properties. Check whether there is any software to intercept and obtain system parameters.
504	Failed to obtain the key of the image file. Check whether the format and content of the image file are correct.
505	Failed to obtain the dynamic encryption key. Try again.
506	The data to be decrypted is not decryptable data.

507	The error message returned because the data to be decrypted does not match the key. Try again.
508	The error message returned because the value corresponding to the specified key does not exist.
599	The error code returned because an unknown error occurred. Try again later.

Security signature interface error codes

Error code	Description
601	The error code returned because the parameters are invalid. Check the parameters.
602	Insufficient memory, please try again.
606	When you use the top signature with the seedkey, the seedsecret corresponding to the seedkey is not found.
607	There is a problem with the `yw_1222.jpg` picture file. Generally, the bundle id of the image generated does not match the bundle id of the application.
608	No `yw_1222.jpg` image file was found. Make sure that the image file is in the project directory. If the image already exists in the project, make sure that the `base64Code` field in the project `meta.config` file is not empty. If this parameter is not specified, manually generate the `yw_1222.jpg` image again.
609	`yw_1222.jpg` the image file format is incorrect, please regenerate the image file. For the generation method, see mPaaS plug-in > Generate wireless bodyguard image. A common scenario is the mixing of two-party and three-party images. The two-party and three-party images are incompatible and need to be generated separately.
610	`yw_1222.jpg` the content within the picture file is incorrect, please regenerate the picture file.
611	The key in the parameter cannot be found in the image file. Make sure that the image file contains this key.
615	The current image version is too low. Upgrade the image to a new version. For more information about how to generate an image, see Generate a wireless bodyguard image.
699	The error code returned because an unknown error occurred. Try again later.

10.3. Gateway log instructions

10.3.1. Gateway server logs

Only private cloud users have the permission to check gateway logs in the server.

Only private cloud users have the permission to check gateway logs in the server.

API summary log

Log path: `~/logs/gateway/gateway-page-digest.log`

- Log printing time
- Request address
- Response
- Result (Y/N)
- Time cost (ms)
- operationType
- System name
- appId
- workspaceId
- Result code
- Client productId
- Client productVersion
- Channel
- User ID
- Device ID
- UUID
- Client trackId
- Client IP
- Network protocol: HTTP or HTTP2
- Data protocol: JSON or PB
- Request size (byte)
- Response size (byte)

- Pressure test identifier
- TraceId: The unique identifier of request. It can link up the summary log, detail log and exception log.
- cpt identifier
- Client system type
- Back-end system time cost
- clientIp type: 4 or 6
- RPC protocol version: 1.0 or 2.0

Format:

```
 Time - (request address,response,result (Y/N),time cost,operationType,system name,appId,workspaceId,result code,client productId,client
productVersion,channel,user ID,device ID,UUID,client trackId,client IP,network protocol,data protocol,request size,response size,whether pressure
test has been done,TraceId,whether it is a component API,Client system type,Back-end system time cost,IP protocol version,RPC protocol version)
```

Sample:

```
 2020-06-03 14:14:08,001 - (/mgw.htm,response,Y,61ms,alipay.mcdp.space.initSpaceInfo,-,84EFA9A281942,default,1000,-,-,-,-
,Wz4Zak5peDgDAGRNW5rFFGhT,Wz4Zak5peDgDAGRNW5rFFGhTN9uqCLa,Wz4Zak5peDgDAGRNW5rFFGhTN9uqCLa,223.104.210.136,HTTP,JSON,2,2406,F,0a1d766715911648479408
```

API detailed log

Log path: `~/logs/gateway/gateway-page-detail.log`

The detailed log is divided into two categories:

- Request log: `[request]`
- Response log: `[ response]`

Request log

- Log printing time
- Client IP
- TraceId
- Log level
- Log type: request
- operationType
- appId
- workspaceId
- requestData
- sessionId
- did: Device ID
- contentType
- mmtp: T or F, indicating whether to use MMTP protocol or not
- async: T or F, indicating whether to implement asynchronous call

Response log

- Log printing time
- Client IP
- TraceId
- Log level
- Log type: response
- operationType
- appId
- workspaceId
- responseData
- resultStatus: Result code
- contentType
- sessionId
- did: Device ID
- mmtp: T or F, indicating whether to use MMTP protocol or not
- async: T or F, indicating whether to implement asynchronous call

Sample:

```
2017-12-21 15:37:10,208 [100.97.90.113][79c731d51513841830208829314258] INFO  -
[request]operationType=com.alipay.gateway.test,appId=2A9ADA1045,workspaceId=antcloud,requestData=***,sessionId=-
,did=WjtkmWe1uHsDADl7BEleyK2L,contentType=JSON,mmtp=F,async=T

2017-12-21 15:37:10,229 [][79c731d51513841830208829314258] INFO  -
[response]operationType=com.alipay.gateway.test,appId=2A9ADA1045,workspaceId=antcloud,responseData=***,resultStatus=1000,contentType=JSON,sessio
,did=WjtkmWe1uHsDADl7BEleyK2L,mmtp=F,async=T
```

API statistical log

Log path: `~/logs/gateway/gateway-page-stat-s.log`

- Log printing time
- operationType
- appId
- workspaceId
- Result: Y/N
- Result code
- Pressure test identifier
- Total requests

- Total time cost of requests (ms)

Format:

```
Time - operationType,appId,workspaceId,result (Y/N),result code,Pressure test identifier (T/F),total requests,total time cost of requests (ms)
```

Sample:

```
2017-12-21 15:34:58,419 - com.alipay.gateway.test,2A9ADA1045,antcloud,Y,1000,F,1,3
```

Gateway thread statistical log

Log path: `~/logs/gateway/gateway-threadpool.log`

- Log printing time
- Thread name
- Number of active threads
- Number of threads in the current thread pool
- Historical maximum number of created threads
- Number of core threads
- Maximum number of threads
- Task queue size
- Remaining queue capacity

Format:

```
Time [thread name,ActiveCount,PoolSize,LargestPoolSize,CorePoolSize,MaximumPoolSize,QueueSize,QueueRemainingCapacity]
```

Sample:

```
2017-12-21 16:33:32,617 [gateway-executor,0,80,80,80,400,0,1000]
```

Gateway configuration log

Log path: `~/logs/gateway/gateway-config.log`

The log records the notifications related with gateway configuration change.

Gateway default log

Log path: `~/logs/gateway/gateway-default.log`

The events that haven't been assigned to any specific log will be printed in this log.

Gateway error log

Log path: `~/logs/gateway/gateway-error.log`

The log records errors and exception stacks.

10.3.2. Gateway SPI logs

This part of the log description is only for business systems that are integrated with mpaasgw-spi-mpc or mpaasgw-spi-dubbo.

API summary logs

Log path: `~/logs/mobileservice/page-digest.log`

- Log print time
- operationType
- Client productId
- Client productVersion
- Duration: Unit: ms
- Result (Y/N)
- Response code
- uniqueId

Format:

```
Time- (operationType,productId,productVersion, duration, result (Y/N), result code, uniqueId)
```

Example:

```
2017-09-12 11:15:57,700 - (com.alipay.gateway.test,ANT_CLOUD_APP,3.0.0.20171214,36ms,Y,1000,79c731d5150518615768657974443)
```

SPI Startup Log

Log path: `~/logs/mobileservice/boot.log`

The registration and startup of the log entry business system mobileservice are divided into the following six phases:

- Start-To-Register-Service: Start parsing the API service interface.
- Start-To-Analyze-Method: Start parsing methods in the API service interface.
- Analyze-Method-Parameter: Parse method parameters.
- Method-Info: The method information.
- Registered-OperationType: A single API operationType is registered.
- Register-Service-Success: Register all API operationTypes in this operation.

This log can help you check whether the operationType is registered successfully.

Example:

```
2017-12-20 11:25:59,746 [Start-To-Register-Service] target: com.alibaba.mpaasgw.biz.shared.rpctest.MockRpcImpl@5b490d5e, interface: interface com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc

2017-12-20 11:25:59,771 [Start-To-Analyze-Method] method=mock

2017-12-20 11:25:59,780 [Analyze-Method-Parameter] parameters=["s"]

2017-12-20 11:25:59,839 [Method-Info] MethodInfo[paramCount=1,paramType={class com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc$Req},paramNames={s},returnType=class
com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc$Resp,target=com.alibaba.mpaasgw.biz.shared.rpctest.MockRpcImpl@5b490d5e,method=public abstract
com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc$Resp
com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc.mock(com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc$Req),interfaceClass=interface
com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc]

2017-12-20 11:25:59,839 [Registered-OperationType] operationType=com.alipay.sofa.mock

2017-12-20 11:25:59,840 [Register-Service-Success] target=com.alibaba.mpaasgw.biz.shared.rpctest.MockRpcImpl@5b490d5e, interface=interface com.alibaba.mpaasgw.biz.shared.rpctest.MockRpc
```

API monitoring logs

Log Path: `~/logs/mobileservice/monitor.log`

Record detailed logs of API requests, including debug logs of API requests and exception stacks in case of errors.

SPI default Log

Log Path: `~/logs/mobileservice/common-default.log`

The default SPI log. If no tracking point is specified for a specific log, this log is hit.

SPI error log

Log Path: `~/logs/mobileservice/common-error.log`

Logs the error and exception stack.

10.4. Business Interface Definition Specification

Considering the limitations of the mobile development environment (especially the iOS system) and keeping the interface definition simple, the server cannot use the full set of Java syntax when defining mobile service interfaces.

The interface definition specification involves three types of definitions:

- Internally supported data class specifications: Java native classes and wrapper classes that are supported.
- User interface class specification: A user-defined interface that contains the method declaration for API calls.
- User-defined entity class specifications: The user-defined entity class (including field declarations). The method parameter or return value in the interface class and other user-defined entity classes are referenced.

Internal Support Data Class Specification

Unsupported data type

- Container types cannot be nested in multiple layers.
- List or Map must contain generic information.
- Generic information for List or Map cannot be of type array
- Single byte is not supported (byte data byte [] is supported)
- Object arrays are not supported. Use list instead.
- Property names cannot be data and description, which will conflict with iOS properties.
- The key of the Map type must be of the String type.
- Type cannot be an abstract class.
- Type cannot be an interface class.

Wrong writing method:

```
public class Req {
    private Map<String,List<Person>> map; // The container type cannot be nested in multiple layers.
    private List<Map<Person>> list; // Container types cannot be nested in multiple layers.
    private List list1; //List or Map must have generic information.
    private Map map1; //List or Map must have generic information.
    private List<Person[]> listArray; // The generic information of a List or Map cannot be of the Array type.
    private byte b; // The value cannot be a single byte.
    private Person[] personArray; // Object arrays are not supported. Use List instead.
    private String description; // The property name cannot be description
}
```

Supported data type

```
boolean, char, double, float, int, long, short
java.lang.Boolean
java.lang.Character
java.lang.Double
java.lang.Float
java.lang.Integer
java.lang.Long
java.lang.Short
java.lang.String
java.util.List, but: must use the type parameter; cannot use its concrete subclass hereinafter referred to as List
java.util.Map, but: the type parameter must be used; its concrete subclass cannot be used; the key type must be String hereinafter referred to
as Map
Enum
byte[]
```

Correct writing method:

```
public class Req {
    private String s = "ss";
    private int i;
    private double d;
    private Long l;
    private long l1;
    private boolean b;
    private List<String> stringList;
    private List<Person> personList;
    private Map<String,Person> map;
    private byte[] bytes;
    private EnumType type;
}

public class Person {
    private String name;
    private int age;
```

User Interface Class Specification

Parameter of method

You cannot quote:

- Enumerated type
- Generics other than Map, List, Set mentioned above
- Abstract class
- Interface class
- Array of native type

You can quote:

- For a specific entity class, the reference type must be consistent with the actual object type; you cannot use the parent class reference type to point to a subclass object.
- Data classes are supported internally, but the array, map, list, and set collection types cannot be nested.

The following is an error example:

```
Map<String,String[]>
Map<String,List<Person>>(Person is a concrete entity class)
List<Map<String,Persion>>
List<Persion[]>
```

Return value of method

You cannot quote:

- Enumerated type
- Generics other than Map, List, and Set mentioned above
- Abstract class
- Interface class
- Array of native type

You can quote:

- For a specific data class, the reference type must be consistent with the actual object type; you cannot use the parent class reference type to point to a subclass object. For example, you cannot use the Object reference to point to other objects.

> **Important**
> If the parent class is a concrete class, the build tool will not detect such an error.

- See the definition of internal support data classes at the beginning of the article. The array, map, list, or set collection types cannot be nested. For more information, see the preceding examples.

Definition of method

- If you use the @ OperationType annotation, methods that are not annotated will be ignored by the build tool.
- The method cannot be overloading.

Limits on code generation tools

- Allows inheritance relationships defined by interface classes, but merges hierarchical relationships.
- Allows but ignores variables defined in interface classes.
- Allows but ignores method declarations in interface classes to throw exceptions.
- A source file can only contain the definition of one interface class, and cannot contain the definition of other classes (internal classes, anonymous classes, etc.).
- The interface class definition itself and the type to which it refers must be an internal supporting data class or can be defined from the source code.

User-Defined Entity Class Specification

Field definition

You cannot quote:

- Enumerated type
- Generics other than Map, List, and Set mentioned above
- Abstract class
- Interface class
- Array of native type

You can quote:

- For a specific entity class, the reference type must be consistent with the actual object type; you cannot use the parent class reference type to point to a subclass object.
- Data classes are supported internally. The array, map, list, or set collection types cannot be nested. For more information, see the preceding examples.
- Properties whose modifiers include transient are ignored.
- final static int-defined constants (other constants or static variables that do not meet this requirement will be ignored).

> **Note**
> Member variable definitions that begin with `is` are not recommended.

Definition of classes

- Can inherit from other entity classes.
- Ignoring its method declarations, the generation tool will automatically generate setter/getter methods based on the entity class fields.

Limits on code generation tools

- The declaration of the property must be one on a single line.
- Interfaces implemented by user-defined entity classes are allowed but ignored.
- A source file can contain only one definition of a user-defined entity class and cannot contain definitions of other classes (inner classes, anonymous classes, etc.).
- The interface class definition itself and the type to which it refers must be an internal supporting data class or can be defined from the source code.

10.5. Key generation method

You can check the key generation methods based on your business requirement. The keys include RSA key, ECC key, and SM2 key.

Prerequisites

You have downloaded and installed OpenSSL tool (V1.1.1 or later version) from OpenSSL official website.

Generate RSA key

1. Open the OpenSSL tool, and run the following command line to generate a RSA private key. You can select to generate a 1024-bit or 2048-bit private key:

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
```

2. Generate RSA public key based on the RSA private key:

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Generate ECC key

1. Open the OpenSSL tool, and run the following command line to generate an ECC key pair. You must select secp256k1 curve.

```
openssl ecparam -name secp256k1 -genkey -noout -out secp256k1-key.pem
```

2. Generate ECC public key based on `secp256k1-key.pem` key pair:

```
openssl ec -in secp256k1-key.pem -pubout -out ecpubkey.pem
```

Generate SM2 key

1. Open OpenSSL, and run the following command line to generate SM2 private key `sm2-key.pem` .

```
openssl ecparam -name SM2 -genkey -noout -out sm2-key.pem
```

2. Generate the SM2 public key `sm2pubkey.pem` based on the private key `sm2-key.pem` .

```
openssl ec -in sm2-key.pem -pubout -out sm2pubkey.pem
```

10.6. Gateway signature mechanism introduction

To prevent client requests from being tampered or forged, a signature mechanism is used for RPC requests. The RPC module automatically implements the signing functions.

To prevent client requests from being tampered or forged, a signature mechanism is used for RPC requests. The RPC module automatically implements the signing functions.

The basic signing and signature verification process is as follows:

1. Convert the `requestBody` content to a character string.
2. Use the Security Guard module to sign the character string with the encryption key stored in the encryption image (Security Guard image).
3. Send the encrypted signature in the request to the gateway.
4. The gateway signs with the same method. The system then checks whether the two signatures are consistent.