

# **Ant Technology**

## **Mobile Gateway Service User Guide**

**Document Version: 20230209**

# Legal disclaimer

**Ant Group all rights reserved©2022.**

No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Ant Group.

## Trademark statement











and other trademarks related to Ant Group are owned by Ant Group. The third-party registered trademarks involved in this document are owned by the right holder according to law.

## Disclaimer

The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Ant Group reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through channels authorized by Ant Group. You must pay attention to the version changes of this document as they occur and download and obtain the latest version of this document from Ant Group's authorized channels. Ant Group does not assume any responsibility for direct or indirect losses caused by improper use of documents.

# Document conventions

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

# Table of Contents

1.Change history	05
2.About Mobile Gateway Service	06
3.Terminology	07
4.Client-side development guide	08
4.1. Android	08
4.1.1. Quick start	08
4.1.2. Advance guide	09
4.2. iOS	11
4.2.1. Add SDK	11
4.2.2. Use the SDK	12
4.3. H5 JS programming	15
5.Server-side development guide	17
5.1. Backend signature verification	17
5.2. Service definition and development	17
5.3. Instructions on gateway helper classes	23
6.Gateway exception troubleshooting	27
7.FAQ	29
8.Reference	30
8.1. Gateway result codes	30
8.2. Security guard result codes	32
8.3. Gateway log instructions	35
8.3.1. Gateway server logs	35
8.3.2. Gateway SPI logs	38
8.4. Service interface definition specifications	38
8.5. Key generation method	41
8.6. Gateway signature mechanism introduction	41

# 1.Change history

Document version	Revisions
V20211105	<ul style="list-style-type: none"><li>Added descriptions about the service circuit breaking and dynamic routing to <a href="#">Mobile Gateway Service overview</a>.</li><li>In the <a href="#">API groups</a> topic, updated the procedure for creating a Duboo API group, and added multi-IDC and registry authentication information.</li><li>In the <a href="#">Configure APIs</a> topic, added descriptions about configuring the circuit breaking mechanism and updated the parameter description.</li><li>In the <a href="#">Gateway management overview</a>, added descriptions about the circuit breaking mechanism.</li><li>Added the <a href="#">Routing rule</a> topic to describe how to configure and manage routing rules.</li></ul>
V20210630	<ul style="list-style-type: none"><li>Updated the Maven dependency version in <a href="#">Service definition and development</a>.</li><li>Added a code sample about using SM2/SM3 for signature signing in <a href="#">Verify the backend signature</a>.</li></ul>

## 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.

### appld

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.Client-side development guide

### 4.1. Android

#### 4.1.1. Quick start

Important: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use [10.1.68](#) or [10.1.60](#) instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see [mPaaS 10.1.68 upgrade guide](#) or [mPaaS 10.1.60 upgrade guide](#).

The gateway serves as a bridge between the client and the server. The client accesses the backend service API through the gateway. With the gateway, you can:

- Encapsulate the communication between the client and server through dynamic proxy.
- Export the codes automatically generated by the server for the use by the client if both the server and client have consistent APIs defined.
- Perform unified process on `RpcException`, and notify users of the exception through pop-up dialog box and toast box.

The mobile gateway supports three types of access: native AAR, mPaaS Inside and component-based (Portal & Bundle).

##### Prerequisites

- For native AAR access, you need to [Add mPaaS to project](#).
- For mPaaS Inside access, you need to first complete [mPaaS Inside access procedure](#).
- For component-based access, you need to first complete [Component-based access procedure](#).

##### Add the SDK

###### Native AAR

Referring to [AAR component management](#), use Component management (AAR) to install the Mobile Gateway component in your projects.

###### mPaaS Inside

Use Component management to install the Mobile Gateway component in your projects.

For more information, see [Manage component dependency](#).

###### Component-based (Portal & Bundle)

Use Component management to install the Mobile Gateway component in your Portal and Bundle projects.

For more information, see [Manage component dependency](#).

##### Initialize mPaaS

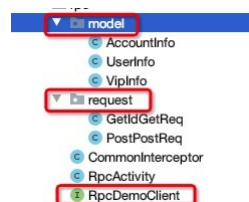
If you access MGS through native AAR or mPaaS Inside method, you need to initialize mPaaS.

```
public class MyApplication extends Application {  
  
    @Override  
    protected void attachBaseContext(Context base) {  
        super.attachBaseContext(base);  
        // Callback settings of initializing mPaaS  
        QuinoxlessFramework.setup(this, new IInitCallback() {  
            @Override  
            public void onPostInit() {  
                // This callback indicates that mPaaS has been initialized, and mPaaS related calls can be made in this callback.  
            }  
        });  
    }  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        // Initialize mPaaS  
        QuinoxlessFramework.init();  
    }  
}
```

##### Generate RPC code

When the App gains access to the backend service in the mobile gateway console, you can go to the mPaaS console and, from the left-side navigation pane, choose **Mobile Gateway Service > Manage APIs > Generate code**, then you can download the RPC code of the client. For more information, see [Mobile Gateway Service > Server Control](#).

The structure of the downloaded RPC code is as follows. It consists of the RPC configuration, request model and response model.





## Call RPC

The client initiates RPC request.

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

RPC exception is thrown through `RpcException`, and you can handle the exception with reference to the result code. For instructions on error codes, see [Gateway result codes](#).

## Related links

- [Code sample](#)
- [Gateway result codes](#)
- [Key generation method](#)

## 4.1.2. Advance guide

This article describes the settings of the mobile gateway RPC interceptor, RPC request header, RPC Cookie and RPC signature.

### Important

Added setting RPC signature content in 10.2.3 baseline.

## Intercept RPC requests

In some situations during business development, it is required to control the network requests from clients (for example, intercept network requests, forbid the access to some interfaces or limit traffic), and you can do that with RPC interceptor.

## Create a global interceptor

```
public class CommonInterceptor implements RpcInterceptor {
    /**
     * Pre-interception: Call back before sending RPC.
     * @param proxy: RPC proxy object.
     * @param clazz: Model class of rpcface, you can judge which RPC model class is called through clazz.
     * @param method: The method called by the current RPC.
     * @throws RpcException
     * @return true means proceeding; false means interrupting the current request and throwing RpcException, with 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-interception: Call back when RPC is successfully initiated.
     * @return true means proceeding; false means interrupting the current request and throwing RpcException, with 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: Call back when initiating RPC failed.
     * @param exception: Error occurs on the current RPC.
     * @return true means proceeding to throw the current exception; false means returning as normal without throwing any exception. If there is no special requirement,
     * don't 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 framework startup, register the interceptor when `RpcService` is being 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 header

Set the RPC request header in the `initRpcConfig` method of the `MainActivity` class. For details, see [Code sample](#).

```
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 cookie

### Set cookie

Set RPC cookie by calling the following interface. Among them, the rule of `Your domain` is all the contents between the first `.` of the gateway URL and the first `/` after the dot. For example, if the gateway URL is `http://test-cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm`, then `Your domain` is `.cloud.alipay.com`.

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

### Remove cookie

Call the following interface to remove the set cookies.

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

## Set SM3 signature verification

After the RPC is initialized, the global signature verification method can be specified as the sm3 type through the `setGlobalSignType` method of the `MPRpc` class.

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

## Set RPC signature

### Interface

```
class TransportConstants {
    public static final int SIGN_TYPE_DEFAULT = 0; // Default signature method, namely md5
    public static final int SIGN_TYPE_MD5 = 1; // md5
    public static final int SIGN_TYPE_HMACSHA256 = 3; // hmacsha256
    public static final int SIGN_TYPE_SHA256 = 4; // sha256
    public static final int SIGN_TYPE_SM3 = 5; // sm3
}

// Global rpc signature algorithm setting
MPRpc.setGlobalSignType(int signType);
```

### Example

Set the global RPC signType and take effect for all RPCs.

```
// Set the signature method to SM3
MPRpc.setGlobalSignType(TransportConstants.SIGN_TYPE_SM3);
```

## 4.2. iOS

### 4.2.1. Add SDK

This guide introduces how to integrate Mobile Gateway Service (MGS) to iOS client. You can integrate MGS to iOS client based on native project with CocoaPods.

#### Prerequisites

You have connected your project to mPaaS. For more information, see [Access mPaaS based on native framework and using Cocoapods](#).

#### Add the SDK

Use CocoaPods plugin to add the MGS SDK. Complete the following steps:

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

```

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

```

2. In the terminal, run `pod install` to complete access.

## Follow-up steps

[Use the SDK](#)

## 4.2.2. Use the SDK

Important: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use [10.1.68](#) or [10.1.60](#) instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see [mPaaS 10.1.68 upgrade guide](#) or [mPaaS 10.1.60 upgrade guide](#).

The RPC related modules include `APMobileNetwork.framework` and `MPMgsAdapter`. APIs in the `MPMgsAdapter` module are recommended.

This topic describes how to use the mobile gateway SDK.

1. [Initialize gateway services](#)
2. [Generate the RPC code](#)
3. [Send a request](#)
4. [Customize a request](#)
5. [Customize an RPC interceptor](#)
6. [Encrypt data](#)
7. [Data signature](#)

## Initialize gateway services

Call the following method to initialize gateway services:

```
[MPRpcInterface initRpc];
```

## Upgrade precautions

The `Category` file of the `DTRpcInterface` class does not need to be added since version 10.1.32. The middle tier implements package reading from `meta.conf`

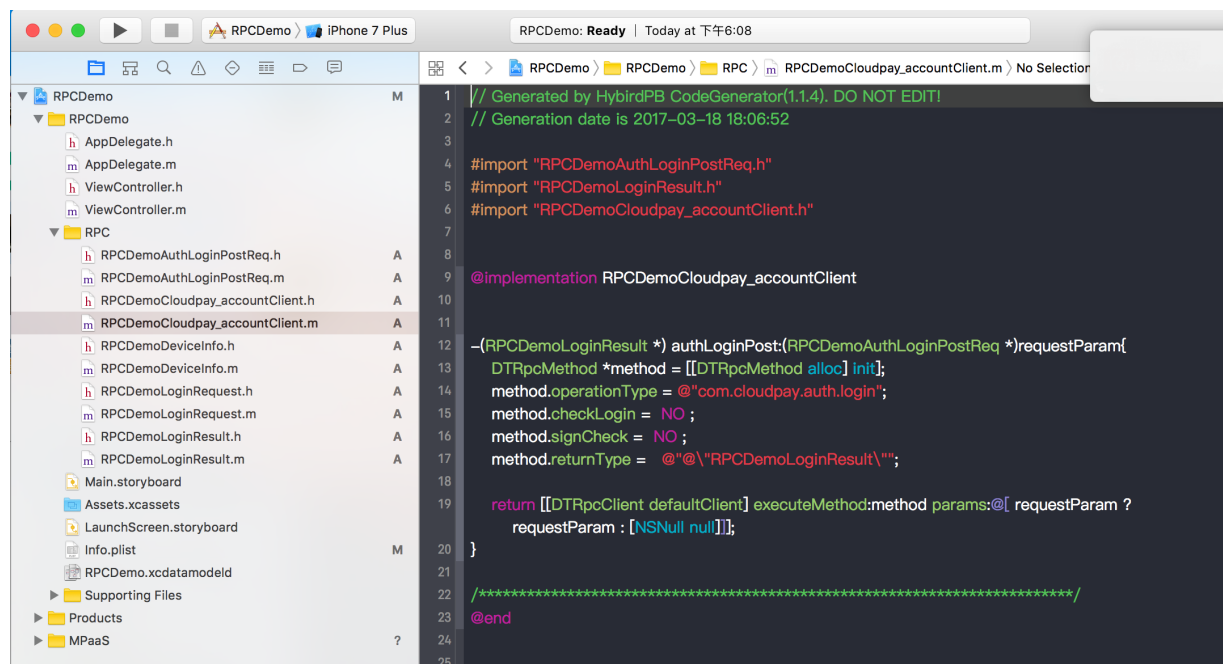
```

1 //
2 // DTRpcInterface+RPCDemo.m
3 // RPCDemo
4 //
5 // Created by yangwei on 2017/03/18. All rights reserved.
6 //
7
8 #import "DTRpcInterface+RPCDemo.h"
9
10 #pragma clang diagnostic push
11 #pragma clang diagnostic ignored "-Wobjc-protocol-method-implementation"
12
13 @implementation DTRpcInterface (RPCDemo)
14
15 - (NSString*)gatewayURL
16 {
17     return @"http://cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm";
18 }
19
20 - (NSString*)signKeyForRequest:(NSURLRequest*)request
21 {
22     return @"5E5FD99271812_IOS";
23 }
24
25 @end
26
27 #pragma clang diagnostic pop
28

```

## Generate the RPC code

When the app accesses the backend service in the mobile gateway console, you can download the RPC code for the client. For more information, see [Generate RPC code](#). The following figure shows the structure of the downloaded RPC code.



In the code:

- `RPCDemoCloudpay_accountClient` : RPC configuration.
- `RPCDemoAuthLoginPostReq` : request model.
- `RPCDemoLoginResult` : response model.

## Send a request

RPC requests must be called in subthreads. The subthread encapsulated with `MPPrcInterface` at the middle tier can be used to call the API. The main thread

```

- (void)sendRpc
{
    __block RPCDemoLoginResult *result = nil;
    [MPPrcInterface 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 (NSEException *exception) {
            NSLog(@"%@", exception);
            NSError *error = [userInfo objectForKey:@"kDTRpcErrorCauseError"]; // Obtain the detailed information of the exception
            NSInteger code = error.code; // Obtain the error code of the exception
        }
    ] completion:^(
        NSString *str = @"";
        if (result && result.success) {
            str = @"Successful login";
        } else {
            str = @"Login failure";
        }
    ) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:str message:nil delegate:nil
            cancelButtonTitle:nil otherButtonTitles:@"ok", nil];
        [alert show];
    }
};
}

```

Note: Use `try catch` to catch exceptions. When the gateway is abnormal, an exception is generated. In this case, check the cause of the exception based on

## Customize a request

`DTRpcMethod` describes RPC request methods. It records the method name, arguments, and return types of RPC requests.

- If no signature is required for request sending, set the `signCheck` attribute of `DTRpcMethod` 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:@[]];
}
```

- To set a timeout interval, set the `timeoutInterval` attribute of `DTRpcMethod`.

```
-(MPDemoUserInfo *) dataPostSetTimeout:(MPDemoPostPostReq *)requestParam
{
    DTRpcMethod *method = [[DTRpcMethod alloc] init];
    method.operationType = @"com.antcloud.request.post";
    method.checkLogin = NO ;
    method.signCheck = YES ;
    method.timeoutInterval = 1; //The timeout interval defines the period in which the client receives a response from the gateway. The timeout interval configured on the server defines the period in which the backend business system delivers a response. The default value is 20, in seconds. A value smaller than 1 is invalid, in which case the default value is used.
    method.returnType = @"@"@"MPDemoUserInfo\"";

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

- To add a header to an API, use the following extension method of `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];
}
```

- To add a header to all APIs, you can use the [interceptor](#). For the specific implementation method, please refer to the mobile gateway [code example](#).
- `checkLogin` attribute is used to verify the API `session` and must work with the gateway console. The default value NO is recommended.

## Customize an RPC interceptor

Based on your business requirements, logical processing may be required before an RPC request is sent or after RPC processing. The RPC module provides an i

### Customize an interceptor

Create an interceptor and implement the `<DTRpcInterceptor>` protocol method to handle related operations before RPC requests are sent or after they are p

```
@interface HXRpcInterceptor : NSObject<DTRpcInterceptor>

@end

@implementation HXRpcInterceptor

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

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

@end
```

## Register the interceptor

Call the extension API at the middle tier to register the customized child interceptor in the interceptor container.

```
HXRpcInterceptor *mpTestInteraptor = [[HXRpcInterceptor alloc] init]; // Customized child interceptor
[MPRpcInterface addRpcInterceptor:mpTestInteraptor];
```

## Encrypt data

The RPC provides a variety of data encryption configurations. For more information, see [Encrypt data](#).

### Data signature(10.2.3 support)

10.2.3 Baseline RPC provides a variety of data signature configuration functions. The 10.2.3 baseline has upgraded the wireless bodyguard SDK to support the

The 10.1.68 baseline defaults to V5 version. Please follow the steps below to use the plugin to generate the V6 image and replace the original `yw_1222.jpg`

1. [Install the mPaaS command line tool](#) (the command line tool package is included in the plugin installation, remove the Xcode signature to set N).
2. Use the following command line to generate a new wireless bodyguard image.

```
mpaas inst sgimage -c /path/to/Ant-mpaas-0D4F51111111-default-IOS.config -V 6 -t 1 -o /path/to/output --app-secret sssssdderrff --verbose
```

#### Note

The config file directory, target file directory, and appsecret parameter descriptions should be replaced with actual values.

3. If the wireless bodyguard has to support the national secret function, please configure the category code according to the following code to set the signature. The optional values of the signature algorithm are as follows:

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

Code example:

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

@interface DTRpcInterface (mPaaSDemo)

@end

@implementation DTRpcInterface (mPaaSDemo)

- (MPAASRPCSignType)customRPCSignType
{
    return MPAASRPCSignTypeSM3;
}

@end
```

## Related topics

- [Security Guard result codes](#)
- [Gateway result codes](#)

## 4.3. H5 JS programming

### Overview

At present, JS is widely used for mobile frontend coding. mPaaS provides another mobile Web solution: HTML5 Container. HTML5 is based on Android and iOS, see [mPaaS overview](#) for more information about how access HTML5.

After the client accesses HTML5, the frontend can conveniently use the gateway:

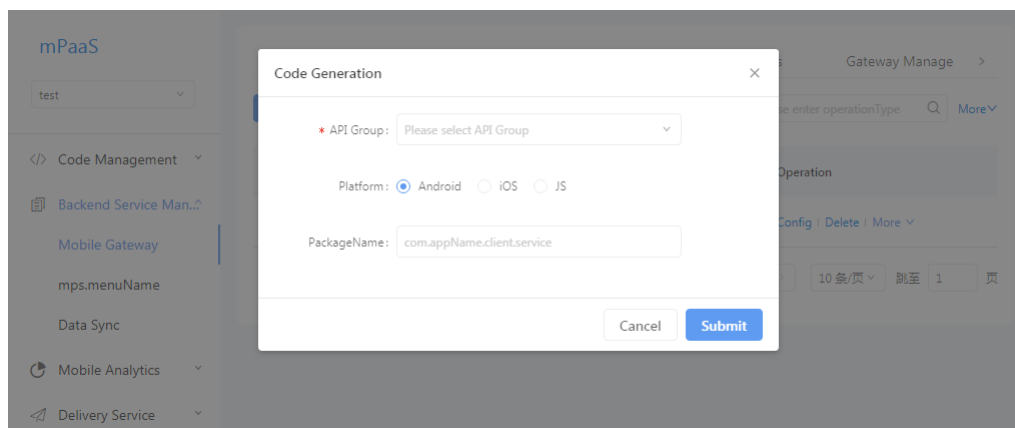
- Encapsulate the communication between client and server via dynamic agent.
- If the server and client have defined the same interfaces, then the codes are automatically generated by server and exported to the client for use.
- Uniformly process `RpcException` by using pop-up dialog, toast, and so on.

### Prerequisites

The Android/iOS client has accessed HTML5 Container.

### Generate JS codes

When application accesses backend services from mobile gateway console, the JS calling codes of RPC can be automatically generated on the console. See [Mobile Gateway Service > Generate codes](#) for more information.



According to the agreed interface parameter, the following template codes will be generated for each API:

```
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 frontend needs to use RPC, it uses the template mentioned above and fills the called request parameters into the template.

### Usage instruction

The codes that JS uses to call RPC are as follows:

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

### Parameter description

Name	Type	Description	Optional	Default value
operationType	string	RPC service name	N	
requestData	array	Parameter of RPC request, developers need to build it on their own according to the actual RPC interface	N	
headers	dictionary	The headers of RPC request	Y	{}
gateway	string	Gateway address	Y	alipay gateway
compress	boolean	Whether or not the request gzip compressing is supported	Y	true
disableLimitView	boolean	Whether or not to disable the auto pop-up of traffic limit window when the traffic of RPC gateway is limited	Y	false

### Result

Result	Type	Description
result	dictionary	The result of RPC response (The string values of non-dictionary structure will be put into a dictionary structure, the key is resData)

### Errors

Error	Description
10	Network error
11	Request timeout
Others	Defined by mobilegw



## 5. Server-side development guide

### 5.1. Backend signature verification

Mobile Gateway Service provides the function of verifying server HTTP service signature to improve the security of data from gateway to server.

Mobile Gateway Service provides the function of verifying server HTTP service signature to improve the security of data from gateway to server.

- After initiating the signature verification on a certain API group on the console, Mobile Gateway Service creates signature information for each API request in the group. The public and private keys used in the signature can be created in the Mobile Gateway Service console.
- After the server reads signature string, it calculates the local signature of the received request, and compares if the signature is consistent with the received signature, thus judging if the request is legal.

#### Read signature

The signature calculated by mobile gateway is saved in the Header of Request, and the Header Key is X-Mgs-Proxy-Signature.

The secret key configured in API group is used to distinguish and obtain the Keys corresponding to different secret key values, and the Header Key is X-Mgs-Proxy-Signature-Secret-Key.

#### Verify signature

##### Organize signature data

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

- HTTPMethod**: HTTPMethod with all characters in upper case, for example: PUT or POST.
- Content-MD5**: It indicates the MD5 value of a request Body. The MD5 value is calculated in the following methods:
  - If **HttpMethod** is not PUT or POST, then the MD5 value is an empty string "", otherwise the step 2 applies.
  - If the request has a Body, and the Body is Form, then the MD5 value is an empty string "", otherwise the step 3 applies.
  - Use the following method to calculate MD5. If the request has no Body, the bodyStream is string "null".

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

**Important:** Even if content-MD5 is an empty string "", the newline "\n" after content-MD5 in the signing method cannot be omitted. Namely, there will be two consecutive "\n" in the signature.

- Url**: It is assembled by Path, Query, and the Form parameter in Body. Suppose the request is in the format of http://ip:port/test/testSign?c=3&a=1 and the Form parameter is b=2&d=4, then the assembling steps are as follows:
  - Obtain Path. Path is the part between ip:port and ?, for example: /test/testSign.
  - If both the Query and Form parameter are empty, the Url is the Path. Otherwise, you need to continue the next step.
  - Concatenate parameters. Sort the Query and Form parameters by key in lexicographic order, and then concatenate them in the format of Key1=Value1&Key2=Value2&...&KeyN=ValueN, for example: a=1&b=2&c=3&d=4.

**Important:** The Query or Form parameters may have multiple values, but only the first value is used.

- Concatenate Url. Url is Path?Key1=Value1&Key2=Value2&...&KeyN=ValueN, for example: /test/testSign?a=1&b=2&c=3&d=4.

#### Verify signature

- Use MD5 algorithm:

```
String sign = "xxxxxxx"; // The signature passed from 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 RSA algorithm:

```
String sign = "xxxxxxx"; // The signature passed from mobile gateway
String publicKey = "xxx"; // The RSA public key of 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"));

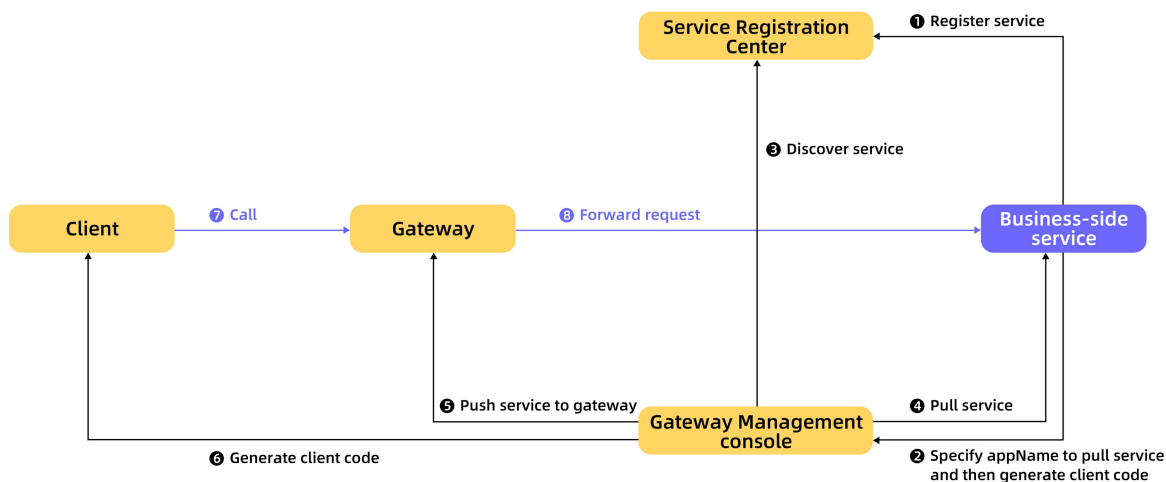
boolean isSignLegal = signature.verify(Base64.decodeBase64(sign.getBytes("UTF-8")));
```

#### Code example

For more details, please refer to [HttpSignUtil.java](#).

### 5.2. Service definition and development

This document is specific for the business systems integrating gateway SPI, such as the business systems which provide mpaaschannel or dubbo API services externally. For the business systems integrating HTTP API, it is unnecessary to refer to this document.



## Introduce the second-party package of gateway

Introduce the following second-party package (ignore if the original project has dependency already) in the `pom.xml` file of the project.

All basic dependencies must be introduced. You can select `MPC`, `Dubbo`, `HRPC`, or `TR` dependency according to the API type that is to be integrated.

### Note

Before introducing the dependencies, make sure that you have completed Maven configuration.

## Basic dependencies

```

<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-spi-adapter</artifactId>
  <version>1.0.5.20200930</version>
</dependency>
<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-log</artifactId>
  <version>1.0.5.20200930</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>1.2.69_noneautotype</version>
</dependency>

```

## MPC dependencies

```

<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-spi-mpc</artifactId>
  <version>1.0.5.20200930</version>
</dependency>
<dependency>
  <groupId>com.alipay.mpaaschannel</groupId>
  <artifactId>common</artifactId>
  <version>2.3.2018071001</version>
</dependency>
<dependency>
  <groupId>com.alipay.mpaaschannel</groupId>
  <artifactId>tenant-client</artifactId>
  <version>2.3.2018071001</version>
</dependency>

```

## Dubbo dependencies

```
<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-spi-dubbo</artifactId>
  <version>1.0.5.20200930</version>
</dependency>
<dependency>
  <groupId>org.apache.dubbo</groupId>
  <artifactId>dubbo</artifactId>
  <version>2.7.8</version>
</dependency>
```

**Note:** For dubbo, use the native version. Don't use dubbox, because it is incompatible.

### HRPC dependencies

```
<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-spi-hrpc</artifactId>
  <version>1.0.5.20200930</version>
</dependency>
<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-registry</artifactId>
  <version>1.0.5.20200930</version>
</dependency>
<dependency>
  <groupId>hessian</groupId>
  <artifactId>hessian</artifactId>
  <version>3.3.6</version>
</dependency>
```

### TR dependencies

```
<dependency>
  <groupId>com.alipay.gateway</groupId>
  <artifactId>mobilegw-unify-spi-sofa</artifactId>
  <version>1.0.5.20200930</version>
</dependency>
```

### Define and implement service interface

Define the service interface `com.alipay.xxxx.MockRpc` based on business requirement, and provide the implementation `com.alipay.xxxx.MockRpcImpl` for this interface.

#### Notes:

- Try to define the incoming parameter in the method as VO. In this way, if you want to add any parameter later, you can add the parameters in the VO without changing the method declaration format.
- For relevant specifications about service interface definition, see [Service interface definition specifications](#).

### Define operationType

Add `@OperationType` annotation on the service interface's method to define the interface that is to publish services.

`@OperationType` contains three parameters:

- `value`: The unique identifier of RPC service; definition rule is `organization.product domain.product.sub-product.operation`.
- `name`: Interface name.
- `desc`: Interface description.

#### Notes:

- `value` must be globally unique in the gateway, you would better to define it as detailed as possible. otherwise, it might be the same as the value of other business party, which might lead service registration failure.
- For convenient maintenance, ensure that the three fields of `@OperationType` are completely entered.

Sample:

```
public interface 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
}
```

## Register API service

Register the pre-defined API service to registration center through the SPI package provided by the gateway.

## Register MPC API service

The following parameters are required for registering MPC API service:

- **registryUrl** : It refers to the address of the registration center. For the shared Ant Financial Cloud registration center, the address is `116.62.81.246`.
- **appName** : It refers to the business-side application name.
- **workspaceId** : It refers to the workspaceId of the workspace where the application is.
- **projectName** : It refers to the projectName of the tenant where the application is.
- **privateKeyPath** : The ClassPath used to store RSA key, which is used to verify the legality when the gateway establishes connection with mpaaschannel. Recommended: Put the RSA key in `/META-INF/config/rsa-mpc-pri-key-{env}.der` where `{env}` indicates the workspace, such as dev, sit, and prod.

## Configure public key

Go to the mAppCenter of the corresponding workspace, and click **Interface Keys > Configure** from the navigation bar on the left to configure the corresponding RSA public key of the private key.

The screenshot shows the mPaaS console interface. On the left, there's a navigation bar with options like 'Manage codes', 'Component services', 'Code configurations', 'Interface keys' (highlighted), 'Manage backend serv.', 'Mobile Analytics', and 'Delivery Service'. The main area is titled 'Configure interface key' and contains a text box with an RSA 2048 key. Below the key, there's a script for generating a key pair using openssl.

```

RSA 2048 key: MIIBJANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuKMLBKFUrqRZmJ0evQ
WEWNMPw0mwEkoYyAhS/3shaNdObHpFDrgVtGnCeMM2oukZfdN1hC+mlkUVI
m9W0AefhR8mO1tp7TYqWQ64qzf+ulviEJuvpM72iw2SHH0X/V9+vlAeZKXx
/E37cWZxpDYzgUimRVgh5bYMNiwzJo5ozKmcCQcCx+5MRU/gPURBhK/VH+*4Q
Z3+IEKs9HakOGRFxbPdvB+clW3Mj2XqKNah7TUzyFuuuuO9DqQhEZz8V5g
cEP/2NVTQCvBbVdRPH45ctaJSSZuTXQ/R4EWu/EhZCwHbPsCdhZVH+tfFwmNY
-----

* 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 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:
*
* $ openssl base64 -in private_key.der -out private_key_base64.der
*
* $ openssl base64 -in public_key.der -out public_key_base64.der
*
* ## remember to clear the whitespace chars and line breaks before submit!!!
    
```

Follow the instructions below to configure the RSA public/private keys. The public key must be configured on mAppCenter while the private key file in `${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:
*
* ## Go to the backend application to configure the generated private key
* $ openssl base64 -in private_key.der -out private_key_base64.der
*
* ## Go to Interface key in mappcenter to configure the generated public key
* $ openssl base64 -in public_key.der -out public_key_base64.der
*
* ### remember to clear the whitespace chars and line breaks before submit!!!
```

## Register through Spring

1. In the Spring configuration file of the corresponding bundle, declare the pre-defined service's Spring bean:

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

2. Declare the Starter bean that is to expose the service in the Spring configuration file of the corresponding bundle.

MpcServiceStarter : This interface registers all beans with @OperationType to the specified registration center through mpaaschannel protocol.

```
<bean id="mpcServiceStarter" class="com.alipay.gateway.spi.mpc.MpcServiceStarter">
  <property name="registryUrl" value="{registry_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>
```

## Register through Spring Boot

Spring Boot is essentially the same as Spring, and the only difference is that the registration method is changed to annotation, without configuring xml file.

1. Register the defined service as bean through annotation: @Service

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

2. Define the starter that is to expose service through 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("{registry_url}");
    mpcServiceStarter.setProjectName("{project_name}");
    mpcServiceStarter.setPrivateKeyPath("{privatekey_path}");
    return mpcServiceStarter;
  }
}
```

## Configure MPC log

For better troubleshooting, you can configure the logs related to MPC on demand. For example, to configure log4j :

```
<!-- [MPC Logger] tenant link, recording the association, settings -->
<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 stream-related data (including a pair of tenant stream <-> component stream) -->
<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 log -->
<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>
```

## Register Dubbo API service

The following parameters are required for registering Dubbo API service:

- `registryUrl` : It refers to the address of the registration center.
- `appName` : It refers to the business-side application name.

## Register through Spring

- In the Spring configuration file of the corresponding bundle, declare the pre-defined service's Spring bean:

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

- In the Spring configuration file of the corresponding bundle, declare the starter bean ( `DubboServiceStarter` ) that is to expose service. This interface will register all beans containing `@OperationType` to the specified registration center through Dubbo protocol.

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

## Register through Spring Boot

Spring Boot is essentially the same as Spring, and the only difference is that the registration method is changed to annotation, without configuring xml file.

- Register the defined service as bean through annotation: `@Service`  
`public class MockRpcImpl implements MockRpc{`  
`}`
- Define the starter that is to expose service through annotation:

```
@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 HRPC API service

The following parameters are required for registering HRPC API service:

- `registryUrl` : It refers to the address of the registration center, required.
- `appName` : It refers to the business-side application name, required.
- `serverPort` : It refers to the listening port of HRPC service, it defaults to 7079, optional.

## Register through Spring

1. In the Spring configuration file of the corresponding bundle, declare the pre-defined service's Spring bean:

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

2. In the Spring configuration file of the corresponding bundle, declare the starter bean ( `HRpcServiceStarter` ) that is to expose service. This interface will register all beans containing `@OperationType` to the specified registration center through HRPC protocol.

```
<bean id="hrpcServiceStarter" class="com.alipay.gateway.spi.hrpc.HRpcServiceStarter">
  <property name="registryUrl" value="${registry_url}"/>
  <property name="appName" value="${app_name}"/>
  <property name="serverPort" value="${serverPort (optional)}"/>
</bean>
```

### Register through Spring Boot

Spring Boot is essentially the same as Spring, and the only difference is that the registration method is changed to annotation, without configuring xml file.

1. Register the defined service as bean through annotation:

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

2. Define the starter that is to expose service through annotation:

```
@Configuration
public class HRpcDemo {
    @Bean(name="hrpcServiceStarter")
    public HRpcServiceStarter hRpcServiceStarter(){
        HRpcServiceStarter hRpcServiceStarter = new HRpcServiceStarter();
        hRpcServiceStarter.setAppName("${app_name}");
        hRpcServiceStarter.setRegistryUrl("${registry_url}");
        hRpcServiceStarter.setServerPort("${serverPort (optional)}");
        return hRpcServiceStarter;
    }
}
```

**Note:** The registry center used by HRPC is ZK, and there can be one or multiple registry URLs (separated with commas), for example: "11.163.193.240:2181" or "11.163.193.240:2181,11.163.193.230:2181" .

### Register TR API service

The following parameter is required for registering TR API service:

- `appName` : It refers to the business-side application name, required.

### Register through Spring

1. In the Spring configuration file of the corresponding bundle, declare the pre-defined service's Spring bean:

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

2. In the Spring configuration file of the corresponding bundle, declare the starter bean ( `SofaServiceStarter` ) that is to expose service. This interface will register all beans containing `@OperationType` to the specified registration center through TR protocol.

```
<bean id="SofaServiceStarter" class="com.alipay.gateway.spi.sofa.SofaServiceStarter">
  <property name="appName" value="${app_name}"/>
</bean>
```

### Register through Spring Boot

Spring Boot is essentially the same as Spring, and the only difference is that the registration method is changed to annotation, without configuring xml file.

1. Register the defined service as bean through annotation:

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

2. Define the starter that is to expose service through annotation:

```
@Configuration
public class TRDemo {
    @Bean(name="sofaServiceStarter")
    public SofaServiceStarter sofaServiceStarter(){
        SofaServiceStarter sofaServiceStarter = new SofaServiceStarter();
        sofaServiceStarter.setAppName("${app_name}");
        return sofaServiceStarter;
    }
}
```

### Result

When you complete the above steps, you can operate the defined API services in gateway, and expose them to the client. For more information, see [Register API](#).

## 5.3. Instructions on gateway helper classes

This topic gives an introduction on the related helper classes involved in Mobile Gateway Service (MGS), including interceptor class and `MobileRpcHolder` , and the corresponding error codes in MSG use.

## Implement the function of interceptors

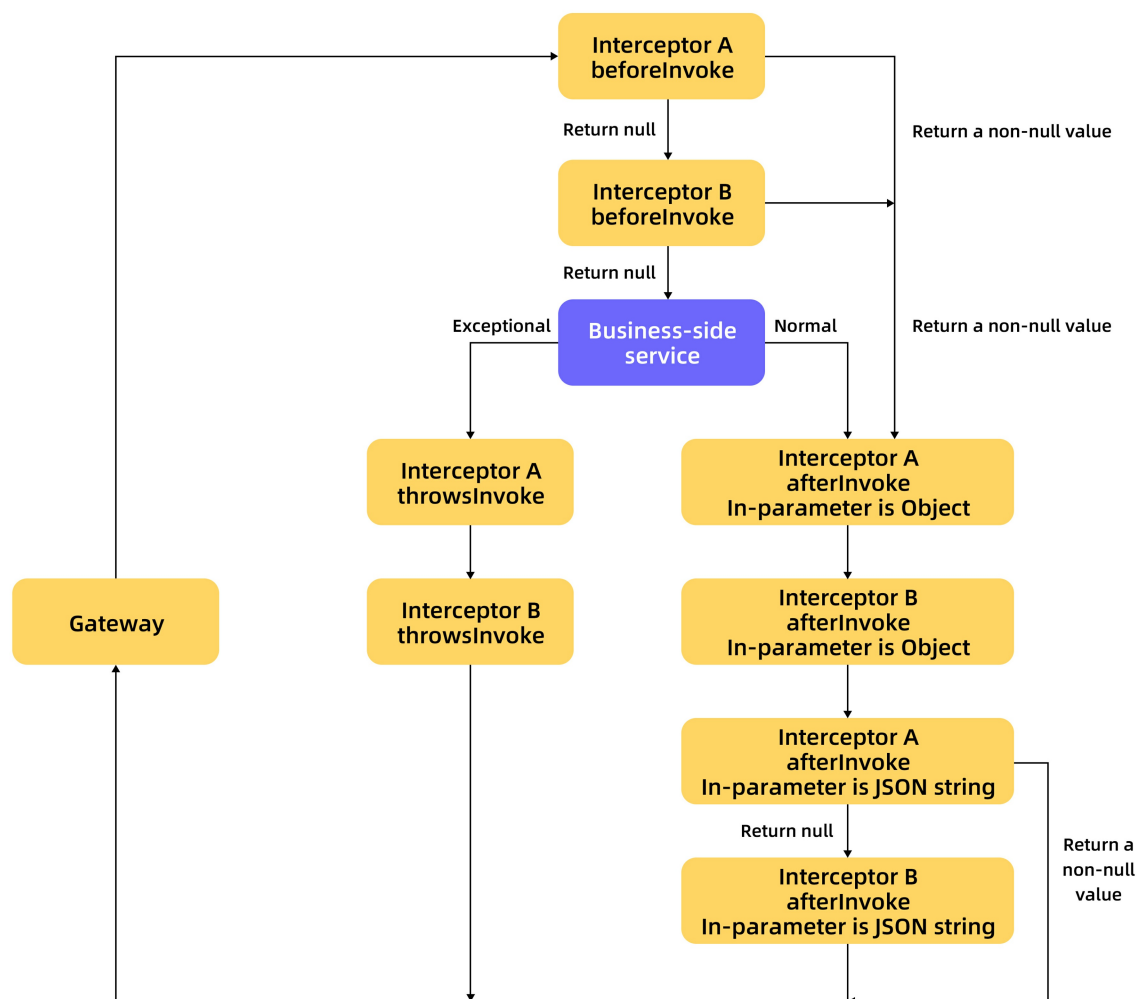
The interceptors are only applicable for non-HTTP services.

`Mobilegw-unify-spi-adapter.jar` actually calls the service method through Java reflection. The service method is also the method specified in `OperationType`. In the process of method invocation, the business system can implement the interceptors that are defined in SPI package to realize extension.

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

### AbstractMobileServiceInterceptor

`MobileServiceInterceptor` mainly provides four methods: `beforeInvoke`, `afterInvoke` (two types: one has the object returned by the business system as incoming parameter and the other one has the JSON string converted from object as incoming parameter), `throwsInvoke` and `getOrder`.



As shown in the above figure, interception mainly takes place at the following points:

- Before method invocation: Use the `beforeInvoke` method, which has a return value. If the return value of this method is not null, the gateway regards the interception a success, skips the `beforeInvoke` method of other interceptors and the methods of the business system, and then directly goes to the `afterInvoke` method.
- After method invocation: Use the `afterInvoke` method. There are two types of `afterInvoke` method. One takes the object returned by the business system as incoming parameter. This method has no return value. Such method in all interceptors will be executed. The other one takes the JSON string as incoming parameter, and this method can change the passed-in JSON data and return them. If the return value of this method is not null, the gateway regards the interception a success, and skips the subsequent `beforeInvoke` methods.
- Method exception: Use the `throwsInvoke` method. This method has no return value. Such method in all interceptors will be executed. The method will be called in case of business system exception.

### MobileServiceInterceptor

`MobileServiceInterceptor` inherits the framework's `Ordered` interface. Therefore, the interceptors that are implemented by the business system can be executed in a specified order by implementing `getOrder` method. The smaller value, the higher execution priority.

### Code sample

1. Encode your own interceptor class, inherit `AbstractMobileServiceInterceptor` class or implement `MobileServiceInterceptor` class.



```
public class MyInterceptor implements MobileServiceInterceptor {

    /*
    Parameter description
    Method: Method of the business system (method defined in @OperationType)
    args: An object array, namely the incoming parameters of the business system's method. The number of the incoming parameters is the array size.
    The business system transforms the type based on demand in use.
    bean: The interface instance of the business system.
    Return value description:
    Object: Return data in the interceptor. If the return value is not null, then the gateway regards the interception a success, and will not invoke the business method.
    Meanwhile, the gateway skips the `beforeInvoke` method of other interceptors, and executes the `afterInvoke` method in the interceptors.
    */

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

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

    @Override
    public String afterInvoke(String returnValue, Method method, Object[] args, Object target) {
        //Note: The incoming parameter here is the JSON string that is converted from the object returned by the business system
        //New JSON data can be returned
        return null;
    }

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

    @Override
    public int getOrder() {
        //Highest (with smallest value) and lowest (with largest value)
        return 0;
    }
}
```

## 2. Publish the implemented `MyInterceptor` class, and make it a Bean.

### o Spring Boot:

Add annotation `@service` on the class.

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

### o Spring:

Make a declaration in the `xml` configuration file.

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

## MobileRpcHolder helper class

`MobileRpcHolder` is a static helper class provided in `mobilegw-unify-spi-adapter.jar`. This helper class defines the relevant information of a request. The main definition is as follows:

```
Map<String, String> session    Save the request session
Map<String, String> header     Save the request header related information
Map<String, String> context    Save the context that is called by the gateway
String    operationType       Save the operationType of this request
```

Before the business system's service ( `OperationType` ) is called, the SPI service will set the above information of `MobileRpcHolder` according to the `MobileRpcRequest` forwarded by the gateway. The information will be cleared after the `OperationType` is called.

The lifecycle of `MobileRpcHolder` is the whole service invocation process. The information will be cleared after service invocation.

The business system can set the information based on requirement, and the information always exists in the process of business service invocation. In the invocation process, the business service can obtain that information. You can dynamically modify the information saved in `MobileRpcHolder` before or after the method invocation through the interceptors.

The following example illustrates how to use `MobileRpcHolder`.

## Code sample

Here is an example of modifying and obtaining a session.

## 1. Modify a session.

Create an interceptor. For specific procedure, see the above example. The following step assumes to intercept before method invocation:

```
@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);
}
```

In this way, you can modify the session information in `MobileRpcHolder`.

## 2. Obtain a session.

The business system can obtain the session information from the service that is defined by itself.

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

For other information like header and context, the operation is the same as the above.

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

## MGS error codes

Mobile Gateway Service has a set of error code specifications. To learn more, see [Gateway result codes](#).

What you have to note is `BizException 6666`. This error is the exception thrown by the gateway upon business system exception.

To return other error codes when an error occurs, the business system can throw `RpcException(ResultEnum resultCode)` to control the error on RPC layer. For example, `resultCode=1001`, the error "have no access privilege" will be returned to the client.

## Code sample

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

## Custom error codes

If the business system is to use the custom error codes, it cannot throw exceptions during service method invocation.

In case of service method exception, the status code 6666 is returned. When the client receives the status code, it regards that an error occurs in the service and will not parse the data returned by the business system. The client parses the returned data only when receiving the status code 1000.

Therefore, the reasonable approach should be that the server and client appoint the specific error codes, catch all exceptions when invoking the service method, and then put the custom error codes in the returned data. In this way, even if the business system has exceptions, the gateway still returns 1000. Meanwhile, the client parses the returned data, and extracts the custom error codes.

## 6. 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 /mgs/mgw.htm HTTP/1.1
Host:
AppId: 910143 App ID
Cookie: JSESSIONID=0A01E89E58541077C1710E980F07D2D974E6548800; __NRF=6CC07DC9C28B1B66AABB76
DId: WSA2rRADetoDAJgw5zOfU8Uq Device ID
User-Agent: /7 CFNetwork/893.14.2 Darwin/17.3.0
Ts: M1u7iGR Signature timestamp
tk: Srwj5yEomKzICCEke9Hb7TjJ19Kpt2wTrREK5pC3g451210
nbappid: 60000003
UniformGateway: https:// /mds/mgw.htm
Content-Length: 265
WorkspaceId: product Workspace ID
Sign: 4c49624c8fb776ec7fa7e51c49891a46 Signature
Platform: iOS Platform
Operation-Type: com.queryOrder RPC interface name
Connection: keep-alive
Accept-Language: zh-cn
x-mgs-encryption: 1 RPC data self-encryption identifier
Accept: */*
Content-Type: application/json RPC data serialization format
Accept-Encoding: br, gzip, deflate
nbversion: 1.1.1.0
```

Headers Cookies Text Hex JavaScript JSON JSON Text Raw

- 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 MGS trace ID
Cache-Control: no-cache
Tips: %E6%93%8D%E4%BD%9C%E6%88%90%E5%8A%9F%E3%80%82 Message of RPC call result
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Encoding: gzip
Result-Status: 1000 RPC call result code
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
```

Headers Text Hex Compressed HTML Raw

#### 2. Query MGS log by TraceId

- Obtain Mgw-TraceId from the response header.
- 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.
- 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
```

- 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.

- Download [PSSH](#) tool.
- 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
```

- Run the following command:

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

## 7.FAQ

### 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 referenced, 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));
      });
    });
  });
}
```

## 8.Reference

### 8.1. Gateway result codes

This article introduces the result codes in the process of using the gateway to facilitate troubleshooting.

#### Result codes for gateway

- 1000 means that API call succeeded, while all other result codes indicate failures.
- 1001-5999 and 7000-7999 are gateway errors.
  - 7000-7999 means that error occurs in the signature verification or decryption performed by mobile security guard. See the [Security Guard result codes](#) for troubleshooting.
  - Other than result codes, you can also check the `Memo` and `tips` fields in the response Header to learn more about the error.
  - Besides, AntStack users can check `~logs/gateway/gateway-error.log` on the gateway server for error details.
- When an exception occurs, you can troubleshoot it with reference to [Gateway exception troubleshooting](#).

Result code	Description	Explanation
1000	Processing succeeded	Succeeded in calling the gateway API.
1001	Access denied	Incorrect Mock format, <code>resultStatus</code> missed; WAF validation failed, or the user has no permission to access the authentication interface.
1002	The maximum call times exceeded	If <a href="#">traffic limiting configuration</a> is enabled, the exception may occur when the traffic limit is triggered.
1005	Unauthorized	Authorization verification failed after enabling <a href="#">API authorization</a> .
2000	Login timed out	If authorization verification function is enabled, the exception may occur if you are not logged in.
3000	RPC interface doesn't exist or is closed	In the workspace of the current workspaceId, the mobile application of the appId has no API service with such operationType configured, or the API service is not in the <code>Open</code> status.
3001	Null request data	<code>requestData</code> in the client-side requests is null. Check if the client-side RPC works normally. For iOS client, it is required to <a href="#">initialize gateway service</a> .
3002	Incorrect data format	The RPC request format is incorrect. AntStack users can check server log <code>gateway-error.log</code> for more information.
3003	Data decryption failed	Data decryption failed.
4001	Service request timed out	MGS timed out while calling the business system. It is caused by high backend business system load. Check the running condition of the backend system. If the timeout setting is not appropriate, you can adjust it appropriately. Note: The timeout period is 3 seconds by default.
4002	Remote call on the business system failed	An exception occurred when MGS called the business system. AntStack users can check server log <code>gateway-error.log</code> for more information.
4003	API group HOST exception	An <code>UnknownHostException</code> exception occurred when MGS called HTTP business system. Check if the domain name configured for the API group exists.

5000	Unknown exception	Unknown errors. AntStack users can check server log <code>gateway-error.log</code> for more information.
7000	No public key available	Mobile App's mobile security guard has no secret key of the appld, or the gateway failed to obtain the corresponding signing key of the appld.
7001	Not enough parameters for signature verification	The gateway server-side signature verification failed.
7002	Signature verification failed	The gateway server-side signature verification failed.
7003	Signature verification - Timeliness failure	The timestamp of API request parameter <code>ts</code> exceeds the valid time set in the system. Check if the client time is consistent with the system time.
7007	Signature verification - <code>ts</code> parameter is missing	API request doesn't provide the <code>ts</code> parameter for signature verification.
7014	Signature verification - <code>sign</code> parameter is missing	API request doesn't provide the <code>sign</code> parameter for signature verification. Generally, the missing of <code>sign</code> parameter is caused by client-side signature data failure. Check if the client-side mobile security guard image is correct.
8002	Cross-domain pre-check request (CORS preflight)	Cross-domain pre-check request.

### Result codes for business system

For the following result codes, you can view the corresponding error information in the business systems' servers.

You can find specific information about the exceptions by checking `~/logs/mobileservice/monitor.log` log on each business system.

Result code	Applicable protocol	Description	Explanation
6000	MPC, DUBBO	RPC - target service not found	Cannot find the released service; the server failed to access the service; or the service has been migrated to somewhere else.
6001	MPC, DUBBO	RPC - target method not found	Cannot find the method in the released service.
6002	MPC, DUBBO	RPC - Incorrect number of parameters	The number of input parameters is not consistent with the number of declared parameters.
6003	MPC, DUBBO	RPC - target method is inaccessible	The target method cannot be called.
6004	HTTP, MPC, DUBBO	RPC - JSON parsing exception	HTTP: Failed to convert RPC parameters to backend HTTP request parameters. MPC/DUBBO: Failed to deserialize RPC JSON data to business parameter objects.
6005	MPC, DUBBO	RPC - Invalid parameters exist when the system calls target method	Invalid parameters exist in call reflection.
6007	MPC, DUBBO	RPC - Login verification service unavailable	You haven't implemented the login verification interface in SPI package, or the login verification interface is incorrectly configured.

6666	HTTP, MPC, DUBBO	RPC - Business exception	<p>HTTP: The HTTP status code returned by the backend system is not 200.</p> <p>MPC/DUBBO: The exception reported by the business system. The exceptions that cannot be handled by RPC are regarded as business exceptions.</p>
------	------------------	--------------------------	---

### Result codes for Android client

Result code	Description	Prompt
0	Unknown error	Unknown error. Please try again later.
1	Client failed to find the communication object, no Transport available.	Network error. Please try again later.
2	Client has no network access. For example, the user shuts down network or disabled the application's network access.	Failed to connect to network.
3	SSL relevant errors, including SSL handshake error, SSL certificate error	Incorrect client certificate. Check if the mobile time setting is correct.
4	Client network connection timeout; TCP connection timeout. Currently, the timeout period is 10s.	Poor network connection
5	Slow client network connection, Data read/write timeout, socketTimeout	Poor network connection
6	No response from server, NoHttpResponseException	Network error. Please try again later.
7	Client network IO error, corresponds to IOException	Network error. Please try again later.
8	Client network request scheduling error, execution thread interrupted	Network error. Please try again later.
9	Client processing error, including serialization error, annotation processing error, thread execution error	Network error. Please try again later.
10	Client data deserialization error, incorrect server data format	Network error. Please try again later.
13	Request interruption error. For example, when the thread is interrupted, the request is interrupted accordingly	Network error. Please try again later.
15	Client network authorization error, HttpHostConnectException, Connection to "xxx" refused, no network access, or the corresponding server refused connection.	Failed to connect to network.
16	DNS parse error	Failed to connect to network, please try again later.
18	Network limited; client request limited. When the volume of client requests exceeds the threshold, the network requests will be limited.	Network has been limited. Please try again later.
code >= 400 & code < 500	HTTP response code is 4xx	Failed to connect to network.
400 > code >= 100 & 500 < code < 600	Reponse codes for HTTP failure	Failed to connect to network. Please try again later.

## 8.2. Security guard result codes

The security guard error codes listed in this article are applicable to both Android and iOS operating systems. The error codes are divided into the following three types according to different error types:



- [Common error codes](#)
- [Static data encryption and decryption error codes](#)
- [Security signature API error codes](#)

If an error occurs, error information `SG ERROR: xxxx` is displayed in the Xcode console. This topic describes related error codes.

### Common error codes

Error code	Description
101	Incorrect parameter.
102	Main plug-in initialization failed.
103	Dependent plug-ins are not introduced. The system will provide the name of the missing plug-ins while printing the error code. Please introduce the plug-in as prompted.
104	Failed to load plug-ins that are introduced. Generally, this error code is printed when <code>-all_load</code> or <code>-ObjC</code> is not added to other linker flags. You can add it to resolve this error.
105	The required plug-in is introduced, but its dependent plug-in is not. The system will provide the name of the missing plug-in while printing the error code. Introduce the plug-in as prompted.
106	The required plug-in is introduced, but its dependent plug-in does not meet the version requirement. The system will provide the required version number of the dependent plug-in while printing the error code. Introduce the dependent plug-in of the correct version as prompted.
107	The required plug-in is introduced, but it does not meet the version requirement.
108	The required plug-in is introduced, but its dependent resource is not.
109	The required plug-in is introduced, but its dependent resource does not meet the version requirement.
121	Image file error. Generally, this error code is printed when the bundle ID used for generating the image file is different from that of the App.
122	Image file not found. Ensure that the image file exists in the project directory.
123	Incorrect image file format. Generate the image file again.
124	The image version is earlier than required.
125	init with authcode Initialization error.
199	Unknown error. Try again.
201	Incorrect parameter.
202	Image file error. Generally, this error code is printed when the bundle ID used for generating the image file is different from that of the App.
203	Image file not found. Ensure that the image file exists in the project directory.
204	Incorrect image file format. Generate the image file again.
205	Incorrect image file content. Generate the image file again.
206	The key in the parameter is not found in the image file. Please use an existing key.

207	The input key is invalid.
208	Insufficient memory. Try again.
209	The key of the specified index does not exist.
212	The version of the image file is earlier than required. Upgrade it.
299	Unknown error. Try again.

### Static data encryption and decryption error codes

Error code	Description
301	Incorrect parameter.
302	Image file error. Generally, this error code is printed when the APK signature used for generating the image file is different from that of the App. Use the APK signature of the App to generate the image file again.
303	Image file not found. Ensure that the image file exists in <code>res\drawable</code> .
304	Incorrect image file format. Generate the image file again. A common scenario where this error occurs is that second-party images are used together with third-party images, while they are incompatible with each other and must be generated separately.
305	Incorrect image file content. Generate the image file again.
306	The key in the parameter is not found in the image file. Use an existing key.
307	The input key is invalid.
308	Insufficient memory. Try again.
309	The key of the specified index does not exist.
310	The data cannot be decrypted.
311	The data to be decrypted does not match the key.
312	The version of the image file is earlier than required. Generate an image file of a later version.
399	Unknown error. Try again.
401	Incorrect parameter.
402	Insufficient memory. Try again.
403	Failed to obtain system attributes. Ensure that no software blocks this operation.
404	Failed to obtain the key in the image file. Ensure that the format and content of the image file are correct.
405	Failed to obtain the dynamic encryption key. Try again.
406	The format of the data to be decrypted does not meet the decryption requirements.

407	The data to be decrypted does not meet the decryption requirements. Ensure that the data is generated after dynamic encryption by Security Guard on this device.
499	Unknown error. Try again.
501	Incorrect parameter.
502	Insufficient memory. Try again.
503	Failed to obtain system attributes. Ensure that no software blocks this operation.
504	Failed to obtain the key in the image file. Ensure that the format and content of the image file are correct.
505	Failed to obtain the dynamic encryption key. Try again.
506	The data cannot be decrypted.
507	The data to be decrypted does not match the key. Try again.
508	No value is found for the input key.
599	Unknown error. Try again.

#### Security signature API error codes

Error code	Description
601	Incorrect parameter.
602	Insufficient memory. Try again.
606	When the top signature with a seed key is used, the corresponding seed secret is not found.
607	<code>yw_1222.jpg</code> image file error. Generally, this error code is printed when the bundle ID used for generating the image file is different from that of the App.
608	Image file <code>yw_1222.jpg</code> not found. Ensure that the image file exists in the project directory. If it exists, ensure that the <code>base64Code</code> field in the <code>meta.config</code> file of the project is specified. If the field is not specified, manually generate the <code>yw_1222.jpg</code> image file again.
609	<code>yw_1222.jpg</code> image file is in an incorrect format. Generate the image file again. A common scenario where this error occurs is that second-party images are used together with third-party images, while they are incompatible with each other and must be generated separately.
610	<code>yw_1222.jpg</code> image file contains incorrect content. Generate the image file again.
611	The key in the parameter is not found in the image file. Use an existing key.
615	The version of the image file is earlier than required. Generate an image file of a later version.
699	Unknown error. Try again.

## 8.3. Gateway log instructions

### 8.3.1. 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 - (/mqw.htm,response,Y,61ms,alipay.mcdp.space.initSpaceInfo,-,B4EFA9A281942,default,1000,-,-,-,Wz4Zak5peDgDAGRNW5rFFGhT,Wz4Zak5peDgDAGRNW5rFFGhTN9uqCLa,Wz4Zak5peDgDAGRNW5rFFGhTN9uqCLa,223.104.210.136,HTTP,JSON,2,2406,F,0a1d7667159116484
```

### 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,sessionId=-,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.

## 8.3.2. Gateway SPI logs

The log description in this section is specific for the business systems which have integrated `mpaasgw-spi-mpc` or `mpaasgw-spi-dubbo`.

### API summary log

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

- Log printing time
- operationType
- Client productId
- Client productVersion
- Time span (ms)
- Result (Y/N)
- Result code
- uniqueId

Format:

Time - (operationType,productId,productVersion,time span,result (Y/N),result code,uniqueId)

Sample:

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

### SPI boot log

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

The boot log records the registration and startup process of the business system `mobileservice`. The process falls into the following stages:

- Start-To-Register-Service: Start parsing API service interface
- Start-To-Analyze-Method: Start parsing methods in the API service interface
- Analyze-Method-Parameter: Parse method parameters
- Method-Info: Method information
- Registered-OperationType: Complete registering single API operationType
- Register-Service-Success: Complete registering all API operationType in the interface

You can check if operationType is successfully registered by viewing this log.

Sample:

```
2017-12-20 11:25:59,746 [Start-To-Register-Service] target: com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpcImpl@5b490d5e, interface: interface com.alibaba.mpaasgw.biz.shared.rpc.test.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.rpc.test.MockRpc$Req],paramNames=[s],returnType=class com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpc$Resp,target=com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpcImpl@5b490d5e,method=public abstract com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpc$Resp com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpc.mock(com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpc$Req),interfaceClass=interface com.alibaba.mpaasgw.biz.shared.rpc.test.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.rpc.test.MockRpcImpl@5b490d5e, interface=interface com.alibaba.mpaasgw.biz.shared.rpc.test.MockRpc
```

### API monitor log

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

The API monitor log records API requests, including the API requests' debug log and the exception stacks in case of errors.

### SPI default log

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

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

### SPI error log

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

The log records errors and exception stacks.

## 8.4. Service interface definition specifications

Given the limitations on mobile development environment (especially iOS system) and to keep simple interface definition, you are not allowed to use the full collection of Java syntax when defining mobile service interfaces on servers. The interface definition specifications involve the following three categories:

Given the limitations on mobile development environment (especially iOS system) and to keep simple interface definition, you are not allowed to use the full collection of Java syntax when defining mobile service interfaces on servers. The interface definition specifications involve the following three categories:

- **Specifications for internally supported data classes:** Applicable for the supported Java native classes and wrapper classes.
- **Specifications for user-defined interface classes:** Applicable for the user-defined interfaces, including the method declaration of API call.
- **Specifications for user-defined entity classes:** Applicable for the user-defined entity classes (including field declaration). Method parameters or returned value, and other user-defined entity classes will be referenced.

## Specifications for internally supported data classes

### Unsupported data types

- Container type cannot be multilayer nested.
- List or Map must have generic information.
- The generic information of List/Map cannot be array.
- Single byte is not supported. Byte data `byte []` is supported.
- Object array is not supported. You must replace it with List.
- Attribute name cannot be `data` or `description`, otherwise it might conflict with iOS attribute.
- The key of Map type must be String.
- Type cannot be abstract class.
- Type cannot be interface class.

Incorrect example:

```
public class Req {
    private Map<String,List<Person>>> map; //Container type cannot be multilayer nested.
    private List<Map<Person>>> list; //Container type cannot be multilayer nested.
    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 List/Map cannot be array.
    private byte b; //It cannot be single byte.
    private Person[] personArray; //Object array is not supported. You must replace it with List.
    private String description; //Attribute name cannot be `description`.
}
```

### Supported data types

```
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 (hereinafter referred to as "List"), but it must use type parameter and cannot use its concrete sub classes.
java.util. Map (hereinafter referred to as " Map"), but it must use type parameter and cannot use its concrete sub classes. The key type must be string.
Enum
byte[]
```

Correct example:

```
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;
}
```

## Specifications for user-defined interface classes

### Parameters of method

Reference not allowed for:

- Enum type
- Generic types except Map, List, and Set mentioned above
- Abstract class

- Interface class
- Native array

Reference allowed for:

- Concrete entity class. The reference type and actual object type must keep consistent. It is not allowed to use parent class to point to sub class objects.
- Internally supported data class, but the collection types such as array, Map, List, and Set cannot be nested.

Incorrect example:

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

## Returned value of method

Reference not allowed for:

- Enum type
- Generic types except Map, List, and Set mentioned above
- Abstract class
- Interface class
- Native array

Reference allowed for:

- Concrete data class. The reference type and actual object type must keep consistent. It is not allowed to use parent class to point to sub class objects. For example, you cannot use Object reference to point to other objects.

### Note

Note: If the parent class is a concrete class, the generation tool cannot detect the error of such class.

- Internally supported data class, see the content at the beginning of this article. The collection types such as array, Map, List, and Set cannot be nested. See the [example](#) mentioned above.

## Definition of method

- Use `@OperationType` annotation. The method without this annotation will be ignored.
- The method cannot be overloaded.

## Limitations on code generation tools

- Allow the inheritance that is defined in the interface class, but the hierarchical relations will be merged.
- Allow but ignore the variables that are defined in the interface class.
- Allow but ignore the method declaration in the interface class.
- One source file can only contain the definition of one interface class, and the definition of other classes (internal class, anonymous class, etc.) are not allowed.
- The interface class defines itself and the types it references must be the internally supported data class, or can obtain the definition from source codes.

## Specifications for user-defined entity classes

### Definition of field

Reference not allowed for:

- Enum type
- Generic types except Map, List, and Set mentioned above
- Abstract class
- Interface class
- Native array

Reference allowed for:

- Concrete entity class. The reference type and actual object type must keep consistent. It is not allowed to use parent class to point to sub class objects.
- Internally supported data class, but the collection types such as array, Map, List, and Set cannot be nested. See the [example](#) mentioned above.
- The attributes with modifier including `transient` will be ignored.
- Constants defined by using `final static int`. Other constants that don't meet this requirement or static variables will be ignored.

### Note

It is not recommended to define the member variables beginning with `is`.

## Definition of class

- A class can inherit from other entity class.
- Ignore its method declaration. The generation tool will automatically generate `setter/getter` method based on the entity class' fields.

## Limitations on code generation tools

- The attribute declaration must be one per line.
- Allow but ignore the interfaces that are implemented by user-defined entity class.



- One source file can only contain the definition of one user-defined entity class, and the definition of other classes (internal class, anonymous class, etc.) are not allowed.
- The interface class defines itself and the types it references must be the internally supported data class, or can obtain the definition from source codes.

## 8.5. 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
```

## 8.6. 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.