蚂蚁科技产品手册

移动网关

产品版本： V20201130

文档版本： V20201130

蚂蚁科技技术文档

蚂蚁科技集团有限公司版权所有 © 2020 ，并保留一切权利。

未经蚂蚁科技事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

商标声明

蚂蚁集团
ANT GROUP
及其他蚂蚁科技服务相关的商标均为蚂蚁科技所有。
本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。蚂蚁科技保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在蚂蚁科技授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过蚂蚁科技授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

目录

1 移动网关简介 1
2 基本概念 2
3 快速开始 2
3.1 HTTP API 2
3.2 HRPC API（仅专有云可用） 5
3.3 Dubbo API（仅专有云可用） 10
3.4 TR API（仅专有云可用） 14
4 接入客户端 19
4.1 接入 Android 19
4.1.1 快速开始 19
4.1.2 进阶指南 21
4.2 接入 iOS 24
4.2.1 添加 SDK 24
4.2.2 使用 SDK 25
4.3 H5 JS 编程 30
5 接入服务端 32
5.1 后端签名校验说明 32
5.2 服务定义与开发 34
5.3 网关辅助类使用说明 43
6 使用控制台 48
6.1 API 分组 48
6.2 API 管理 50
6.2.1 注册 API 50
6.2.2 配置 API 51
6.2.3 API 授权 55
6.2.4 API 限流 59
6.2.5 API 缓存 60
6.2.6 API 模拟 62
6.2.7 同步 API 63
6.2.8 导出及导入 API 64
6.3 API 调用 65
6.3.1 API 测试 65
6.3.2 生成代码 67
6.4 网关管理 68
6.4.1 网关管理功能介绍 68
6.4.2 数据加密 70
6.4.3 跨域资源共享 72
6.5 数据模型 75
6.6 API 分析 76
7 网关异常排查 77
8 常见问题 79
9 参考 80
9.1 网关结果码说明 80
9.2 无线保镖结果码说明 82
9.3 网关日志说明 85
9.3.1 网关服务端日志 85
9.3.2 网关 SPI 日志 89
9.4 业务接口定义规范 90
9.5 密钥生成方法 94
9.6 网关签名机制 95

1 移动网关简介

移动网关服务（Mobile Gateway Service，MGS）是移动开发平台（mPaaS）提供的连接移动客户端与服务端的组件产品。该组件简化了移动端与服务端的数据协议和通讯协议，能够显著提升开发效率和网络通讯效率。

功能特点

移动网关是连接移动客户端跟服务端的桥梁，移动客户端通过网关来访问后台服务接口。移动网关能够：

- 自动生成客户端的 RPC 调用代码，用户不需要关心网络通信、协议以及使用的数据格式。
- 将服务端返回的数据自动反解生成 Objective-C 对象，无需额外编码。
- 提供数据压缩、缓存、批量调用等增强服务。
- 统一进行异常处理，如弹出对话框、Toast 提示框等。
- 支持 RPC 拦截器，实现定制化的请求与处理。
- 实行统一的安全加密机制和防篡改的请求签名验证机制。
- 限流管控，保护后台服务器。

价值优势

移动网关服务的优势在于：

- 简单配置即可适配多种终端，连接异构的后端服务。
- 自动生成移动端 SDK，实现前后端分离，提升开发效率。
- 支持服务注册、发现与管控，实现服务聚合与集成，降低管理成本和安全风险。
- 提供优化后的数据协议与通讯协议，提高网络通讯质量和效率。

应用场景

移动网关服务的应用场景如下：

开放移动服务能力

随着移动互联网、普惠金融的迅猛发展，企业越来越迫切地希望将现有成熟的后端服务开放出去。接入移动网关服务，无需额外工作，即可形成移动服务能力。

一套服务，多端输出

移动互联时代，服务需要支持多样化的终端设备，这往往极大地增加了系统复杂性。企业只需在移动网关中定义服务，便能支持多种终端接入。

异构服务，建立标准统一的对外服务接口

企业往往存在多种语言和结构的后端服务，只需遵循一定的标准接入移动网关，就可以对外开放标准统一的服务接口。

2 基本概念

本术语表按拼音首字母对术语进行排序。

API 分组（API group）

API 归属的分组，可以是具体的系统名、模块名或者抽象的标识。

API 服务标识（OperationType）

API 服务标识是 API 服务的唯一标识。

HRPC

HRPC 是基于 HTTP 实现的 RPC 方案。

工作空间标识（workspaceId）

移动平台工作空间的标识，用于隔离不同的环境。

MPC

MPC 为 mpaaschannel 的缩写，是 mPaaS 自行实现的一套 RPC 方案。

移动 App 标识（appId）

移动应用标识是创建 mPaaS 应用时生成的标识。

移动网关服务（（Mobile Gateway Service）

移动网关服务（Mobile Gateway Service，MGS）是提供网关 API 服务的组件名称。

3 快速开始

3.1 HTTP API

快速开始指导您在 10 分钟内，注册并发布一个供移动端调用的 HTTP 类型的 API 服务。整体过程分为 5 步：

1. 注册 API 分组
2. 注册 API 服务
3. 配置 API 服务
4. 测试 API 服务
5. 生成客户端 SDK

准备

1. 登录控制台，在 **产品与服务** 中选择 **移动开发平台 mPaaS** 进入移动开发平台主页。
2. 切换至正确的工作空间后，点击需要接入 API 服务的 APP 名称。
3. 在左侧导航栏选择 **移动网关**，进入移动网关配置页面。

注册 API 分组

1. 选择 **API 分组** 选项卡进入 API 分组列表页，点击 **创建 API 分组** 按钮。

   在弹出的对话框中填写表单信息。

   - **分组类型**：此处选择 HTTP。
   - **API 分组**：必填，提供服务的业务系统的英文名称。
   - **服务地址**：必填，业务系统的 HTTP/HTTPS URL。
   - **超时时间**：选填，发送请求至业务系统时的超时时间，单位毫秒；默认值：3000 ms。

   点击 **确定** 按钮提交。
   如需进一步完善 **API 分组** 相关配置，请阅读 配置 API 分组 。

注册 API 服务

1. 选择 **API 管理** 选项卡进入 API 列表页，点击 **创建 API** 按钮。

   在弹出的对话框中填写 API 信息。

- **API 类型**：此处选择 HTTP。
- **添加方式**：目前只支持手动方式注册 HTTP API。
- **operationType**：必填，当前环境和应用下 API 服务唯一标识。命名规则：组织.产品域.产品.子产品.操作。

点击 **确定** 按钮提交。

配置 API 服务

1. 在 **API 管理** 选项卡中，点击 API 列表操作列中的 **配置**，进入 API 配置页面。

在 API 配置区域，点击 **修改** 按钮进行相应参数的编辑；修改完成后，点击 **保存** 按钮。

说明：

- 为了快速入门，您可以先将 **高级配置** 中的 **签名校验** 关闭。关于签名校验的详细信息，请参考 签名校验说明 。
- 关于 API 配置的详细信息，请参考 API 配置 。

打开右上方开关，使 API 服务处于 **开通** 状态。（只有处于 开通 状态的 API 服务才能被调用）

测试 API 服务

相关信息请参考 API 测试 。

生成客户端 SDK

相关信息请参考 代码生成 。

结果

完成上述几步操作，API 服务即可供客户端调用。有关客户端开发的更多信息，参见下列 **客户端开发指南**：

- Android
- iOS
- JS

3.2 HRPC API（仅专有云可用）

HRPC 服务仅适用于 **专有云**。快速开始指导您注册并发布一个供移动端调用的 HRPC 类型的 API 服务。整体过程分为六步：

1. 服务端开发
2. 注册 API 分组
3. 注册 API 服务
4. 配置 API 服务
5. 测试 API 服务
6. 生成客户端 SDK

服务端开发

引入网关二方包

在项目的主 pom.xml 文件中引入如下二方包（如原工程已经有依赖，请忽略）。mobilegw-unify 系列依赖请使用最新版本，当前最新版本为 1.0.5.20180810。

```
<!-- mobilegw unify dependency-->
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-hrpc</artifactId>
<version>${the-lastest-version}</version>
</dependency>
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-adapter</artifactId>
<version>${the-lastest-version}</version>
</dependency>
<dependency>
```

```
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-log</artifactId>
<version>${the-lastest-version}</version>
</dependency>
<dependency>
<groupId>hessian</groupId>
<artifactId>hessian</artifactId>
<version>3.1.3</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>1.2.48</version>
</dependency>

<!-- 如果使用了pb，请加入如下依赖-->
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.6.1</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-core</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-runtime</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-api</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-collectionschema</artifactId>
<version>1.3.8.20160722</version>
</dependency>
```

定义服务接口并实现接口

按照业务需求，定义服务接口：com.alipay.xxxx.MockRpc。

说明：

- 建议将方法定义中的入参定义为 VO，这样后期可以直接在 VO 中添加参数，而不改变方法的声明格式。
- 服务接口定义的相关规范，请参见 业务接口定义规范 。

提供该接口的实现 com.alipay.xxxx.MockRpcImpl。

定义 OperationType

在服务接口的方法上添加 @OperationType 注解，定义发布服务的接口名称。@OperationType 有三个参数成员，为便于维护，请填写完整：

- **value**：接口唯一标识，在网关全局唯一。定义该参数值时，尽量详细，否则会和其他业务方的 value 值一样，导致无法注册服务。定义规则：组织.产品域.产品.子产品.操作。
- **name**：接口中文名称。
- **desc**：接口描述。

示例如下：

```
public interface MockRpc {

@OperationType(value="com.alipay.mock", name="HRPC mock 接口", desc="复杂 mock 接口")
Resp mock(Req s);

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

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

// ignore getter & setter
}

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

// ignore getter & setter
}
```

声明 API 服务

该步骤目的是将定义好的 RPC 服务，通过网关提供的 SPI 包，声明为对外提供服务的 API。需要如下三个参数：

- **registryUrl**：必填，注册中心的地址。
- **appName**：必填，业务方的应用名。
- **serverPort**：选填，HRPC 服务监听端口，默认为 7079。

您可以通过 Spring 或 Spring Boot 方式声明 API 服务。

Spring 声明方式

在对应 bundle 的 Spring 配置文件中，声明上述服务的 Spring Bean。示例如下：

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

在对应 bundle 的 Spring 配置文件中，声明 com.alipay.gateway.spi.hrpc.HRpcServiceStarter 类型的 Spring Bean。HRpcServiceStarter 会将所有带有 @OperationType 的 bean 通过 HRPC 协议注册到指定的注册中心。示例如下：

```
<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可选，默认为 7079}"/>
</bean>
```

Spring Boot 声明方式

以注解的方式声明上述服务的 Spring Bean。示例如下：

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

2. 以注解的方式声明 com.alipay.gateway.spi.hrpc.HRpcServiceStarter 类型的 Spring Bean。
   HRpcServiceStarter 会将所有带有 @OperationType 的 bean 通过 HRPC 协议注册到指定的注册中心。
   示例如下：

```
@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可选}");
return hRpcServiceStarter;
}
}
```

注册 API 分组

进入移动网关管理页面。

- 登录控制台，在 **产品与服务** 中选择 **移动开发平台 mPaaS** 进入移动开发平台主页。

- 切换至正确的工作空间后，单击需要接入 API 服务的 APP 名称。
- 在左侧导航栏选择 **移动网关**，进入移动网关管理页面。

选择 **API 分组** 选项卡进入 API 分组列表页，单击 **创建 API 分组** 按钮。

在弹出的对话框中填写表单信息。

- **分组类型**：此处选择 HRPC。
- **API 分组**：必填，提供服务的业务系统的英文名称。
- **注册中心**：必填，注册中心 URL。
- **超时时间**：选填，发送请求至业务系统时的超时时间，单位毫秒，默认值为 3000 ms。

单击 **确定** 按钮提交。
如需进一步完善 API 分组相关配置，请阅读 配置分组 。

注册 API 服务

选择 **API 管理** 选项卡进入 API 列表页，单击 **创建 API** 按钮。

在弹出的对话框中，**API 类型** 选择 **HRPC**，选择 **API 分组**，在拉取到的 operationType 列表中勾选需要的服务，单击 **确认** 按钮。

配置 API 服务

1. 单击 API 列表操作列中的 **配置**，进入 API 配置页面。

   在 API 配置区域，单击 **修改** 按钮进行相应参数的编辑；修改完成后，单击 **保存** 按钮。

说明：

- 为了快速入门，您可以先将 **高级配置** 中的 **签名校验** 关闭。关于签名校验的详细信息，请参见 签名校验说明 。
- 关于 API 配置的详细信息，请参见 API 配置 。

检查右上方开关，保证 API 服务处于 **开通** 状态。只有处于开通状态的 API 服务才能被调用。

测试 API 服务

相关信息请参见 API 测试 。

生成客户端 SDK

相关信息请参见 代码生成 。

结果

完成上述几步操作，API 服务即可供客户端调用。有关客户端开发的更多信息，参见下列客户端开发指南：

- Android
- iOS
- JS

3.3 Dubbo API（仅专有云可用）

Dubbo 服务仅适用于 **专有云**。快速开始指导您注册并发布一个供移动端调用的 Dubbo 类型的 API 服务。整体过程分为六步：

1. 服务端开发
2. 注册 API 分组
3. 注册 API 服务
4. 配置 API 服务
5. 测试 API 服务
6. 生成客户端 SDK

服务端开发

引入网关二方包

在项目的主 pom.xml 文件中引入如下二方包（如原工程已经有依赖，请忽略）。其中 mobilegw-unify 系列依赖请使用最新版本，当前最新版本为 1.0.5.20180810。

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

```
<artifactId>commons-lang3</artifactId>
<version>3.5</version>
</dependency>
<dependency>
<groupId>com.alibaba</groupId>
<artifactId>fastjson</artifactId>
<version>1.2.48</version>
</dependency>

<!-- 如果使用了pb，请加入如下依赖-->
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.6.1</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-core</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-runtime</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-api</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-collectionschema</artifactId>
<version>1.3.8.20160722</version>
</dependency>
```

定义服务接口并实现接口

按照业务需求，定义服务接口：com.alipay.xxxx.MockRpc。

注意：

- 建议将方法定义中的入参定义为 VO，这样后期可以直接在 VO 中添加参数，而不改变方法的声明格式。
- 服务接口定义的相关规范，请参见 业务接口定义规范 。

提供该接口的实现 com.alipay.xxxx.MockRpcImpl。

定义 OperationType

在服务接口的方法上添加 @OperationType 注解，定义发布服务的接口名称。@OperationType 有三个参数成员，为便于维护，请填写完整：

- **value**：接口唯一标识，在网关全局唯一。定义该参数值时，尽量详细，否则会和其他业务方的 value 值一样，导致无法注册服务。定义规则：组织.产品域.产品.子产品.操作。

- **name**：接口中文名称。
- **desc**：接口描述。

示例如下：

```
public interface MockRpc {

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

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

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

// ignore getter & setter
}

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

// ignore getter & setter
}
```

声明 API 服务

该步骤目的是将定义好的 RPC 服务，通过网关提供的 SPI 包，声明为对外提供服务的 API。需要如下 2 个参数：

- **registryUrl**： 注册中心的地址。
- **appName**：业务方的应用名。

您可以通过 Spring 或 Spring Boot 方式声明 API 服务。

Spring 声明方式

在对应 bundle 的 Spring 配置文件中，声明上述服务的 Spring Bean。示例如下：

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

在对应 bundle 的 Spring 配置文件中，声明 com.alipay.gateway.spi.dubbo.DubboServiceStarter 类型的 Spring Bean。

DubboServiceStarter 会将所有带有 @OperationType 的 bean 通过 Dubbo 协议注册到指定的注册中心。示例如下：

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

Spring Boot 声明方式

以注解的方式声明上述服务的 Spring Bean。示例如下：

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

2. 以注解的方式声明 com.alipay.gateway.spi.dubbo.DubboServiceStarter 类型的 Spring Bean。DubboServiceStarter 会将所有带有 @OperationType 的 bean 通过 Dubbo 协议注册到指定的注册中心。示例如下：

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

注册 API 分组

进入移动网关管理页面。

- 登录控制台，在 **产品与服务** 中选择 **移动开发平台 mPaaS** 进入移动开发平台主页。
- 切换至正确的工作空间后，单击需要接入 API 服务的 APP 名称。
- 在左侧导航栏选择 **移动网关**，进入移动网关管理页面。

选择 **API 分组** 选项卡进入 API 分组列表页，单击 **创建 API 分组** 按钮。

在弹出的对话框中填写表单信息。

- **分组类型**：此处选择 DUBBO。
- **API 分组**：必填，提供服务的业务系统的英文名称。
- **注册中心**：必填，注册中心地址。
- **超时时间**：选填，发送请求至业务系统时的超时时间，单位毫秒，默认值为 3000 ms。

单击 **确定** 按钮提交。

如需进一步完善 API 分组相关配置，请阅读 配置分组 。

注册 API 服务

选择 **API 管理** 选项卡进入 API 列表页，单击 **创建 API** 按钮。

在弹出的对话框中，**API 类型** 选择 **DUBBO**，选择 **API 分组**，在拉取到的 operationType 列表中勾选需要的服务，单击 **确认** 按钮。

配置 API 服务

1. 在 **API 管理** 选项卡中，单击 API 列表操作列中的 **配置**，进入 API 配置页面。

   在 API 配置区域，单击 **修改** 按钮进行相应参数的编辑；修改完成后，单击 **保存** 按钮。

   注意：
   - 为了快速入门，您可以先关闭 **高级配置** 中的 **签名校验** 开关。关于签名校验的详细信息，请参见 签名校验说明 。
   - 关于 API 配置的详细信息，请参见 API 配置 。

   检查右上方开关，保证 API 服务处于 **开通** 状态。只有处于开通状态的 API 服务才能被调用。

测试 API 服务

相关信息请参见 API 测试 。

生成客户端 SDK

相关信息请参见 代码生成 。

结果

完成上述几步操作，API 服务即可供客户端调用。有关客户端开发的更多信息，参见下列客户端开发指南：

- Android
- iOS
- JS

3.4 TR API（仅专有云可用）

TR 是蚂蚁金服 RPC 框架，仅适用于 **专有云**。快速开始指导您注册并发布一个供移动端调用的 TR 类型的 API 服务。整体过程分为六步：

1. 服务端开发
2. 注册 API 分组
3. 注册 API 服务
4. 配置 API 服务

5. 测试 API 服务
6. 生成客户端 SDK

服务端开发

引入网关二方包

在项目的主 pom.xml 文件中引入如下二方包（如原工程已经有依赖，请忽略）。mobilegw-unify 系列依赖请使用最新版本，当前最新版本为 1.0.5.20180810。

```
<!-- mobilegw unify dependency-->
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-sofa</artifactId>
<version>${the-lastest-version}</version>
</dependency>
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-adapter</artifactId>
<version>${the-lastest-version}</version>
</dependency>
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-log</artifactId>
<version>${the-lastest-version}</version>
</dependency>
<dependency>
<groupId>com.alipay.hybirdpb</groupId>
<artifactId>classparser</artifactId>
<version>1.2.2</version>
</dependency>
<dependency>
<groupId>org.apache.commons</groupId>
<artifactId>commons-lang3</artifactId>
<version>3.5</version>
</dependency>
<dependency>
<groupId>com.alibaba</groupId>
<artifactId>fastjson</artifactId>
<version>1.2.48</version>
</dependency>


<!-- 如果使用了pb，请加入如下依赖-->
<dependency>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
<version>2.6.1</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-core</artifactId>
<version>1.3.8.20160722</version>
</dependency>
```

```
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-runtime</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-api</artifactId>
<version>1.3.8.20160722</version>
</dependency>
<dependency>
<groupId>io.protostuff</groupId>
<artifactId>protostuff-collectionschema</artifactId>
<version>1.3.8.20160722</version>
</dependency>
```

定义服务接口并实现接口

按照业务需求，定义服务接口：com.alipay.xxxx.MockRpc。

说明：

- 建议将方法定义中的入参定义为 VO，这样后期可以直接在 VO 中添加参数，而不改变方法的声明格式。
- 服务接口定义的相关规范，请参见 业务接口定义规范 。

提供该接口的实现 com.alipay.xxxx.MockRpcImpl。

定义 OperationType

在服务接口的方法上添加 @OperationType 注解，定义发布服务的接口名称。@OperationType 有 3 个参数成员，为便于维护，请填写完整：

- **value**：接口唯一标识，在网关全局唯一。定义该参数值时，尽量详细，否则会和其他业务方的 value 值一样，导致无法注册服务定义规则：组织.产品域.产品.子产品.操作。
- **name**：接口中文名称。
- **desc**：接口描述。

示例如下：

```
public interface MockRpc {

@OperationType(value="com.alipay.mock", name="MPC mock 接口", desc="复杂 mock 接口")
Resp mock(Req s);

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

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

```
// ignore getter & setter
}

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

// ignore getter & setter
}
```

声明 API 服务

该步骤目的是将定义好的 RPC 服务，通过网关提供的 SPI 包，声明为对外提供服务的 API。需要填写参数 **appName**，参数值为业务方的应用名。

您可以通过 Spring 或 Spring Boot 方式声明 API 服务。

Spring 声明方式

在对应 bundle 的 Spring 配置文件中，声明上述服务的 Spring Bean。示例如下：

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

在对应 bundle 的 Spring 配置文件中，声明 com.alipay.gateway.spi.sofa.SofaServiceStarter 类型的 Spring Bean。SofaServiceStarter 会将所有带有 @OperationType 的 bean 通过 TR 协议暴露给网关调用。示例如下：

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

Spring Boot 声明方式

以注解的方式声明上述服务的 Spring Bean。示例如下：

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

以注解的方式声明 com.alipay.gateway.spi.sofa.SofaServiceStarter 类型的 Spring Bean。SofaServiceStarter 会将所有带有 @OperationType 的 bean 通过 TR 协议暴露给网关调用。示例如下：

```
@Configuration
public class TRDemo {
@Bean(name="sofaServiceStarter")
public SofaServiceStarter sofaServiceStarter(){
SofaServiceStarter sofaServiceStarter = new SofaServiceStarter();
```

```
sofaServiceStarter.setAppName("${app_name}");
return sofaServiceStarter;
}
}
```

注册 API 分组

进入移动网关管理页面。

- 登录控制台，在 **产品与服务** 中选择 **移动开发平台 mPaaS** 进入移动开发平台主页。
- 切换至正确的工作空间后，点击需要接入 API 服务的 APP 名称。
- 在左侧导航栏选择 **移动网关**，进入移动网关管理页面。

选择 **API 分组** 选项卡进入 API 分组列表页，点击 **添加 API 分组** 按钮。

在弹出的对话框中填写表单信息。

- **分组类型**：此处选择 TR。
- **API 分组**：必填，提供服务的业务系统的英文名称。
- **直连地址**：选填，需要直连时填写。由 IP 和端口组成，端口不指定时默认为 12200。
- **超时时间**：选填，发送请求至业务系统时的超时时间，单位毫秒，默认值为 3000 ms。

点击 **确定** 按钮提交。
如需进一步完善 API 分组相关配置，请阅读 配置分组 。

注册 API 服务

选择 **API 管理** 选项卡进入 API 列表页，点击 **添加 API** 按钮。

在弹出的对话框中，**API 类型** 选择 TR，选择 **API 分组**，在拉取到的 operationType 列表中勾选需要的服务，点击 **确认** 按钮。

配置 API 服务

1. 点击 API 列表操作列中的 **配置**，进入 API 配置页面。

在 API 配置区域，点击 **修改** 按钮进行相应参数的编辑；修改完成后，点击 **保存** 按钮。

说明：

- 为了快速入门，您可以先将 **高级配置** 中的 **签名校验** 关闭。关于签名校验的详细信息，请参考 签名校验说明 。
- 关于 API 配置的详细信息，请参考 API 配置 。

检查右上方开关，保证 API 服务处于 **开通** 状态。只有处于开通状态的 API 服务才能被调用。

测试 API 服务

相关信息请参考 API 测试 。

生成客户端 SDK

相关信息请参考 代码生成 。

结果

完成上述几步操作，API 服务即可供客户端调用。有关客户端开发的更多信息，参见下列客户端开发指南：

- Android
- iOS
- JS

4 接入客户端

4.1 接入 Android

4.1.1 快速开始

> **重要**：自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68 升级指南 或 mPaaS 10.1.60 升级指南 进行基线版本升级。

网关是连接客户端与服务端的桥梁，客户端通过网关来访问后台服务接口。通过使用网关，您可以实现以下目的：

- 通过动态代理的方式，封装客户端和服务端之间的通讯。
- 如果服务端和客户端定义了一致的接口，可由服务端自动生成代码并导出给客户端使用。
- 对 RpcException 进行统一的异常处理，弹对话框、toast 消息框等。

移动网关支持 **原生 AAR 接入**、**mPaaS Inside 接入** 和 **组件化（Portal&Bundle）接入** 三种接入方式。

前置条件

- 若采用原生 AAR 方式接入，需先完成 将 mPaaS 添加到您的项目中 的前提条件和后续相关步骤。
- 若采用 mPaaS Inside 方式接入，需先完成 mPaaS Inside 接入流程 。
- 若采用组件化方式接入，需先完成 组件化接入流程 。

添加 SDK

原生 AAR 方式

参考 AAR 组件管理 ，通过 **组件管理（AAR）** 在工程中安装 **H5 容器** 组件。

mPaaS Inside 方式

在工程中通过 **组件管理** 安装 **H5 容器** 组件。
更多信息，请参考 管理组件依赖 。

组件化（Portal&Bundle）方式

在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **H5 容器** 组件。
更多信息，请参考 管理组件依赖 。

初始化 mPaaS

如果您使用 **原生 AAR 接入** 或 **mPaaS Inside 接入** ，您需要初始化 mPaaS。

```
public class MyApplication extends Application {

@Override
protected void attachBaseContext(Context base) {
super.attachBaseContext(base);
// mPaaS 初始化回调设置
QuinoxlessFramework.setup(this, new IInitCallback() {
@Override
public void onPostInit() {
// 此回调表示 mPaaS 已经初始化完成，mPaaS 相关调用可在这个回调里进行
}
});
}

@Override
public void onCreate() {
super.onCreate();
// mPaaS 初始化
QuinoxlessFramework.init();
}
}
```

生成 RPC 代码

当 App 在移动网关控制台接入后台服务后，进入 mPaaS 控制台，从左侧导航栏选择 **移动网关** > **API 管理** > **生成代码**，下载客户端的 RPC 代码。详细说明参见 移动网关 > 服务端管控 相关文档。

下载的 RPC 代码结构如下，包括 RPC 配置、request 模型和 response 模型。

调用 RPC

```
// 获取 client 实例
RpcDemoClient client = MPRpc.getRpcProxy(RpcDemoClient.class);
// 设置请求
GetIdGetReq req = new GetIdGetReq();
req.id ="123";
req.age = 14;
req.isMale = true;
// 发起 rpc 请求
String response = client.getIdGet(req);
```

相关链接

- 代码示例
- 网关结果码说明
- 密钥生成方法

4.1.2 进阶指南

本文对移动网关 RPC 拦截器、RPC 请求头以及 RPC Cookie 的设置进行说明。

RPC 拦截

在业务开发中，如果在某些情况下需要控制客户端的网络请求（拦截网络请求，禁止访问某些接口，或者限流），可以通过 RPC 拦截器实现。

创建全局拦截器

```
public class CommonInterceptor implements RpcInterceptor {

/**
```

```
* 前置拦截：发送 RPC 之前回调。
* @param proxy RPC 代理对象。
* @param clazz rpcface 模型类，通过 clazz 参数可以判断当前调用的是哪个 RPC 模型类
* @param method 当前 RPC 调用的方法。
* @throws RpcException
* @return true 表示继续向下执行，false 表示中断当前请求，抛出 RpcException，错误码：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;
}

/**后置拦截：发起 RPC 成功之后回调。
*@return true 表示继续向下执行，false 表示中断当前请求，抛出 RpcException，错误码：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;
}

/**
* 异常拦截：发起 RPC 失败之后回调。
* @param exception 表示当前 RPC 出错异常。
* @return true 表示将当前异常继续向上抛出，false 表示不要抛出异常，正常返回，没有特殊需求，切勿返回 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;
}
}
```

注册拦截器

在框架启动过程中，初始化 RpcService 时，将拦截器注册上去，例如：

```
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());
}
}
```

设置 RPC 请求头

在 MainActivity 类的initRpcConfig方法中，设置 RPC 请求头。具体参考 代码示例。

```
private void initRpcConfig(RpcService rpcService) {
//设置请求头
Map<String, String> headerMap = new HashMap<>();
headerMap.put("key1","val1");
headerMap.put("key2","val2");
rpcInvokeContext.setRequestHeaders(headerMap);
}
```

设置 RPC cookie

设置 cookie

通过调用以下接口来进行 RPC cookie 设置。其中，Your domain 的规则是网关 url 的第一个 . 及及其后第一个 / 之前的所有内容。例如，网关 URL 为 http://test-cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm，那么 Your domain 则是 .cloud.alipay.com。

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

移除 cookie

通过调用以下接口即可移除设置的 cookie。

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

4.2 接入 iOS

4.2.1 添加 SDK

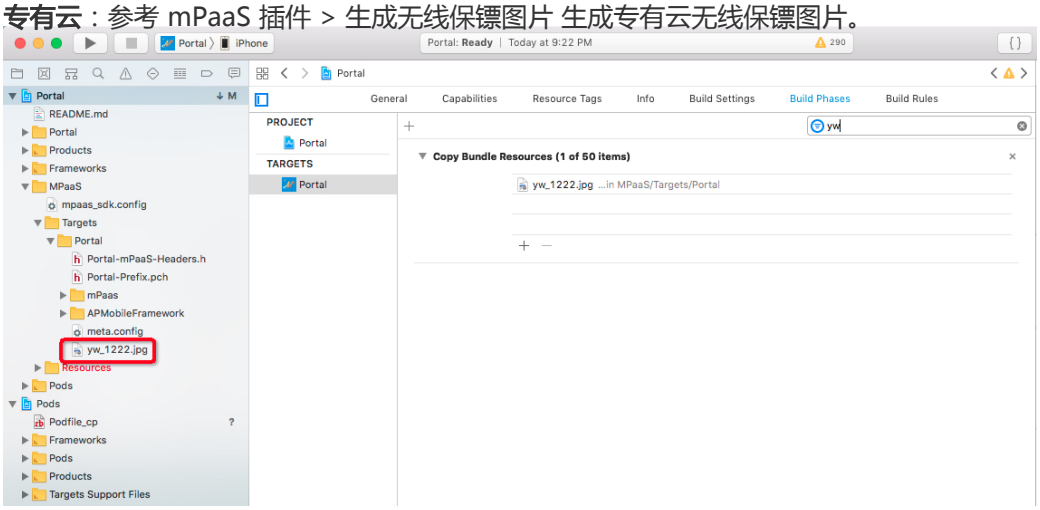
本文介绍如何快速将移动网关组件接入到 iOS 客户端。

移动网关支持基于 mPaaS 框架接入、基于已有工程且使用 mPaaS 插件接入以及基于已有工程且使用 CocoaPods 接入三种接入方式。您可以参考 接入方式介绍 ，根据实际业务情况选择合适的接入方式。

前置条件

添加移动网关 SDK 前，确保已完成以下操作：

1. 接入工程到 mPaaS：
    - 基于 mPaaS 框架接入
    - 基于已有工程且使用 mPaaS 插件接入
    - 基于已有工程且使用 CocoaPods 接入
2. 引入用于 请求加签 的无线保镖图片 yw_1222.jpg：
    - **公有云**：完成第 1 步后，工程中会自动生成无线保镖图片，您无需额外操作。
    - **专有云**：参考 mPaaS 插件 > 生成无线保镖图片 生成专有云无线保镖图片。



添加 SDK

根据您采用的接入方式，请选择相应的添加方式。

使用 mPaaS Xcode Extension 插件

此方式适用于 **基于 mPaaS 框架接入** 或 **基于已有工程且使用 mPaaS 插件接入** 的接入方式。

1. 点击 Xcode 菜单项 **Editor** > **mPaaS** > **编辑工程**，打开编辑工程页面。
2. 选择 **移动网关**，保存后点击 **开始编辑**，即可完成添加。

使用 cocoapods-mPaaS 插件

此方式适用于 **基于已有工程且使用 CocoaPods 接入** 的接入方式。

1. 在 Podfile 文件中，使用 mPaaS_pod"mPaaS_RPC" 添加移动网关组件依赖。

2. 在命令行中执行 pod install 即可完成接入。

后续步骤

使用 SDK

4.2.2 使用 SDK

> **重要**：自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请使用 10.1.68 或 10.1.60 系列基线。可以参考 mPaaS 10.1.68 升级指南 或 mPaaS 10.1.60 升级指南 进行基线版本升级。

RPC 相关模块为 APMobileNetwork.framework、MPMgsAdapter，推荐使用 MPMgsAdapter 中的接口。

本文引导您通过以下步骤使用移动网关 SDK：

1. 初始化网关服务
2. 生成 RPC 代码
3. 发送请求
4. 请求自定义配置
5. 自定义 RPC 拦截器
6. 数据加密

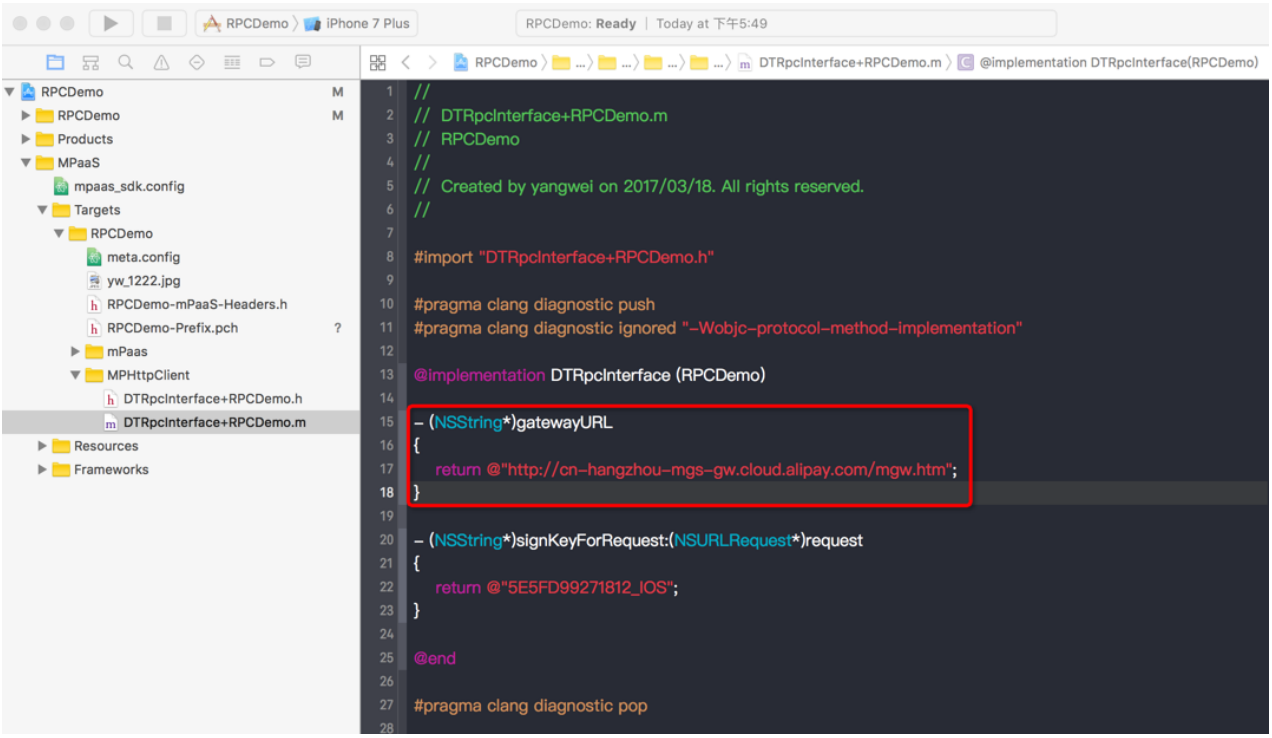
初始化网关服务

调用以下方法初始化网关服务：

```
[MPRpcInterface initRpc];
```

旧版本升级注意事项

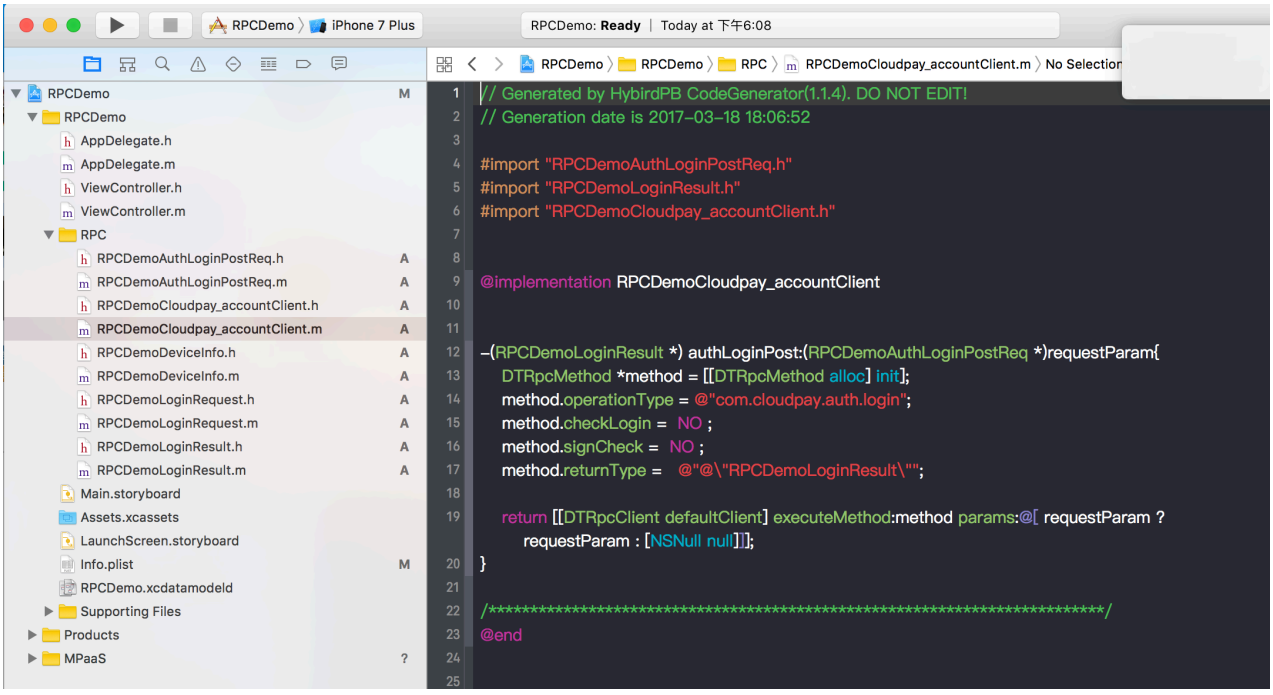
10.1.32 版本之后不再需要添加 DTRpcInterface 类的 Category 文件，中间层会实现包装从 meta.config 中读取，升级版本后请检查工程中是否存在旧版本配置，如果有请移除。下面为新版本应移除的 DTRpcInterface 类的 Category 文件。



生成 RPC 代码

当 App 在移动网关控制台接入后台服务后，即可下载客户端的 RPC 代码。更多信息请参考 生成代码 。

下载的 RPC 代码结构如下：



其中：

- RPCDemoCloudpay_accountClient 为 RPC 配置。
- RPCDemoAuthLoginPostReq 为 request 模型。
- RPCDemoLoginResult 为 response 模型。

发送请求

RPC 请求必须在子线程调用，可使用中间层中 MPRpcInterface 封装的子线程调用接口，回调方法默认为主线程。示例代码如下：

```
- (void)sendRpc
{
__block RPCDemoLoginResult *result = nil;
[MPRpcInterface callAsyncBlock:^{
@try
{
RPCDemoLoginRequest *req = [[RPCDemoLoginRequest alloc] init];
req.loginId = @"alipayAdmin";
req.loginPassword = @"123456";
RPCDemoAuthLoginPostReq *loginPostReq = [[RPCDemoAuthLoginPostReq alloc] init];
loginPostReq._requestBody = req;
RPCDemoCloudpay_accountClient *service = [[RPCDemoCloudpay_accountClient alloc] init];
result = [service authLoginPost:loginPostReq];
}
@catch (NSException *exception) {
NSLog(@"%@", exception);
NSError *error = [userInfo objectForKey:@"kDTRpcErrorCauseError"]; // 获取异常详细信息
NSInteger code = error.code; // 获取异常详细信息错误码
}
} completion:^{
NSString *str = @"";
if (result && result.success) {
str = @"登录成功";
} else {
str = @"登录失败";
}

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

说明：要使用 try catch 捕获异常；当网关异常时会抛出，根据 结果码 查询原因。

请求自定义配置

DTRpcMethod 为 RPC 请求方法描述，记录 RPC 请求的方法名、参数、返回类型等信息。

如果发送请求时，不需要加签，可以将 DTRpcMethod 的 signCheck 属性设置为 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:@[ ]];
```

```
}
```

如果需要设置超时时间，可以配置 DTRpcMethod 的 timeoutInterval 属性。

```
-(MPDemoUserInfo *) dataPostSetTimeout:(MPDemoPostPostReq *)requestParam
{
DTRpcMethod *method = [[DTRpcMethod alloc] init];
method.operationType = @"com.antcloud.request.post";
method.checkLogin = NO ;
method.signCheck = YES ;
method.timeoutInterval = 1; // 这个超时时间是客户端收到网关返回的时间，服务端配置的超时时间是后端业务系统的返回时间；默认 20s，设置小于 1 时无效即为默认值
method.returnType = @"@\"MPDemoUserInfo\"";

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

如果需要为接口添加 Header，可以使用下面 DTRpcClient 的扩展方法。

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

// 针对接口添加 header
NSDictionary *customHeader = @{@"testKey": @"testValue"};
return [[DTRpcClient defaultClient] executeMethod:method params:@[ ] requestHeaderField:customHeader responseHeaderFields:nil];
}
```

- 如果需要为所有接口添加 Header，可以参考下方 拦截器 的使用，采用拦截器的方式实现。具体实现方法请参考移动网关 代码示例。
- checkLogin 属性为接口 session 校验使用，需要配合网关控制台完成，默认设置为 NO 即可。

自定义 RPC 拦截器

基于业务需求，可能需要在 RPC 发送前，或 RPC 处理完成后进行相关逻辑处理，RPC 模块提供拦截器机制处理此类需求。

自定义拦截器

创建拦截器，并实现 <DTRpcInterceptor> 协议的方法，用来处理 RPC 请求前后的相关操作。

```
@interface HXRpcInterceptor : NSObject<DTRpcInterceptor>

@end
```

```
@implementation HXRpcInterceptor

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

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

注册拦截器

您可通过调用中间层的扩展接口，在拦截器容器中注册自定义的子拦截器。

```
HXRpcInterceptor *mpTestIntercaptor = [[HXRpcInterceptor alloc] init]; // 自定义子拦截器
[MPRpcInterface addRpcInterceptor:mpTestIntercaptor];
```

数据加密

RPC 提供多种数据加密配置功能，详情参考 数据加密配置 。

相关链接

- 无线保镖结果码说明
- 网关结果码说明

4.3 H5 JS 编程

概要

目前很多移动端的前端都是用 JS 进行编码。mPaaS 也提供了移动端 web 解决方案：H5容器。H5是承载于 Android 和 iOS 之上，客户端接入详见 mPaaS 概述 。

在客户端接入H5后，前端可以很方便地使用网关：

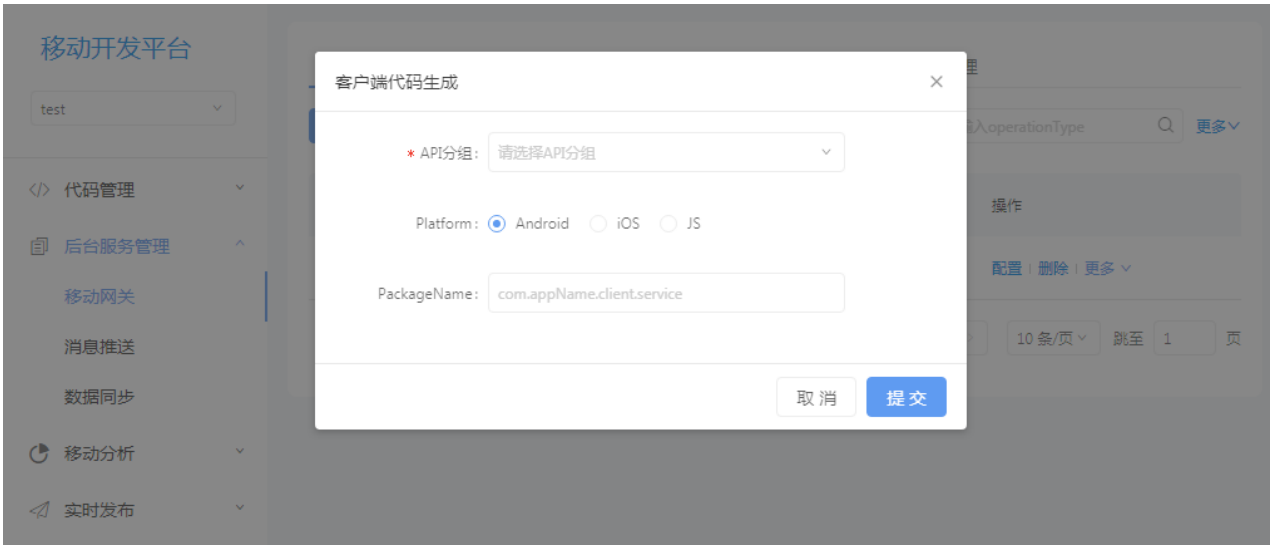
- 通过动态代理的方式，封装客户端和服务端之间的通讯。
- 如果服务端和客户端定义了一致的接口，可由服务端自动生成代码并导出给客户端使用。
- 对 RpcException 进行统一的异常处理，弹对话框、toast 消息框等。

前置条件

Android/iOS 客户端已经接入 H5 容器。

生成 JS 代码

当 App 在移动网关控制台接入后台服务后，即可在控制台自动生成 RPC 的 JS 调用代码，详细说明参见 代码生成 。



目前针对每个 API，根据约定的接口参数，都会生成如下模版代码：

```
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);
});
```

前端需要使用到 RPC 时，直接使用上面的模版，填入调用的请求参数。

使用说明

JS 调用 RPC 如下：

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

参数说明

名称	类型	描述	可选	默认值
operationType	string	RPC 服务名称	N	
requestData	array	RPC 请示的参数。需要开发者根据具体 RPC 接口自行构造	N	
headers	dictionary	RPC 请求设置的 headers	Y	{}

gateway	string	网关地址	Y	alipay 网关
compress	boolean	是否支持 request gzip 压缩	Y	true
disableLimitView	boolean	RPC 网关被限流时是否禁止自动弹出统一限流弹窗。	Y	false

结果

结果	类型	描述
result	dictionary	RPC 响应的结果（非字典结构的字符串值会被放入一个字典结构，key 为 resData）

错误

error	描述
10	网络错误
11	请求超时
其他	由 mobilegw 网关定义

5 接入服务端

5.1 后端签名校验说明

移动网关提供服务端 HTTP 服务签名验证功能，提高从网关到服务端的数据安全性。

- 在网关控制台开启某一 API 分组的签名校验后，移动网关会对该分组里面的每一个 API 请求创建签名信息，签名使用的公私钥可在网关控制台创建；
- 服务端读取签名字符串后，对收到的请求进行本地签名计算，比对与收到的签名是否一致，以此来判断请求是否合法。

读取签名

移动网关计算的签名保存在 Request 的 Header 中，Header Key 为 X-Mgs-Proxy-Signature。

API 分组中配置的 **密钥 Key** 可用来区分和获取不同的密钥值对应的 Key，Header Key 为 X-Mgs-Proxy-Signature-Secret-Key。

验签方法

组织加签数据

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

HTTPMethod：全大写的 HTTPMethod，如 PUT 或 POST 等。

Content-MD5：请求 Body 的 MD5 值。计算方法如下：

- 若 HTTPMethod 非 PUT 或 POST 之一，则 MD5 为空字符串 ""；否则执行第二步。
- 若请求有 Body 且 Body 为 Form 表单，则 MD5 为空字符串 ""；否则执行第三步。
- 使用以下方式计算 MD5。其中，当请求无 Body 时，bodyStream 为字符串 "null"。

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

重要：即使 content-MD5 为空字符串 ""，加签方法中 content-MD5 后面的换行符“\n” 也不能省略，即此时签名中会有连续两个 “\n” 。

Url：由 Path、Query 以及 Body 中的 Form 参数组装而成。假设请求格式为 http://ip:port/test/testSign?c=3&a=1 且 Form 中的参数为 b=2&d=4，组装步骤如下：

- 获取 Path：Path 是 ip:port 之后、? 之前的部分。此例中即为 /test/testSign。
- 若请求 Query 和 Form 参数均为空，则 Url 即为 Path；否则进行下一步。
- 拼接参数：将 Query 和 Form 中的参数根据 Key 按照字典序排序，然后拼接为 Key1=Value1&Key2=Value2&...&KeyN=ValueN。此例中即为 a=1&b=2&c=3&d=4。

  重要：Query 或 Form 参数的 Value 可能有多个，只取第一个 Value 即可。

- 拼接 Url：Url 为 Path?Key1=Value1&Key2=Value2&...&KeyN=ValueN。此例中即为 /test/testSign?a=1&b=2&c=3&d=4。

验证签名

- 采用 MD5 算法验签

```
String sign ="xxxxxxx";//移动网关传过来的签名
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;
```

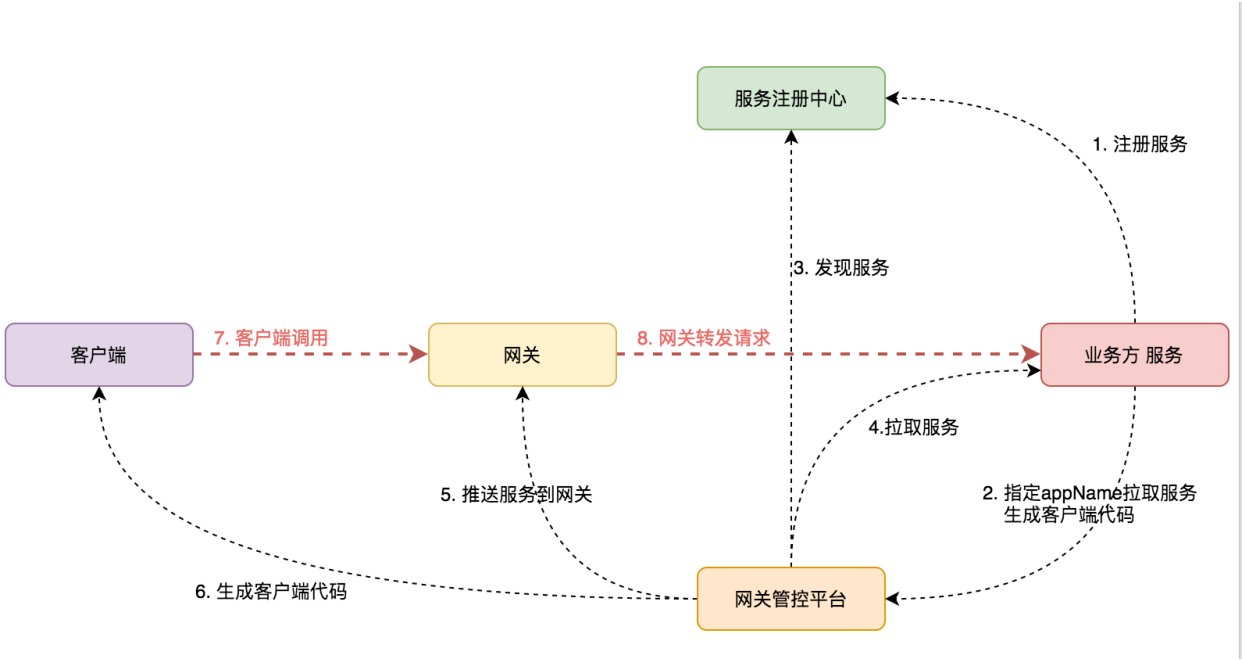
- 采用 RSA 算法验签

```
String sign ="xxxxxxx"; //移动网关传过来的签名
String publicKey ="xxx"; //移动网关的 RSA 公钥

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"")));
```

5.2 服务定义与开发

此文档只针对集成了网关 SPI 的系统，如对外暴露 mpaaschannel 或 dubbo 类型 API 服务的业务系统。对于使用 HTTP API 的业务系统无需查看此文档。



引入网关二方包

在项目的主 pom.xml 文件中引入如下二方包（若原工程已经有依赖，请忽略）。

基础依赖都需要引用，请根据实际需要集成的 API 类型，选择 MPC、Dubbo、HRPC 或 TR 依赖。

说明：引入依赖前，请确认您完成了 Maven 的配置。

基础依赖

```
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-adapter</artifactId>
<version>1.0.5.20180503</version>
</dependency>
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-log</artifactId>
<version>1.0.5.20180503</version>
</dependency>
<dependency>
<groupId>com.alipay.hybirdpb</groupId>
```

```
<artifactId>classparser</artifactId>
<version>1.2.1</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>1.2.48</version>
</dependency>
```

MPC 依赖

```
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-mpc</artifactId>
<version>1.0.5.20180503</version>
</dependency>
<dependency>
<groupId>com.alipay.mpaaschannel</groupId>
<artifactId>common</artifactId>
<version>2.3.20180501001</version>
</dependency>
<dependency>
<groupId>com.alipay.mpaaschannel</groupId>
<artifactId>tenant-client</artifactId>
<version>2.3.20180501001</version>
</dependency>
```

Dubbo 依赖

```
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-dubbo</artifactId>
<version>1.0.5.20180503</version>
</dependency>
```

说明：dubbo 请使用原生版本，不要使用 dubbox（不兼容）。

HRPC 依赖

```
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-hrpc</artifactId>
<version>1.0.5.20180810</version>
</dependency>
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-registry</artifactId>
```

```
<version>1.0.5.20180810</version>
</dependency>
<dependency>
<groupId>hessian</groupId>
<artifactId>hessian</artifactId>
<version>3.1.3</version>
</dependency>
```

TR 依赖

```
<dependency>
<groupId>com.alipay.gateway</groupId>
<artifactId>mobilegw-unify-spi-sofa</artifactId>
<version>1.0.5.20180810</version>
</dependency>
```

定义服务接口并实现

按照业务需求，定义服务接口：com.alipay.xxxx.MockRpc； 并提供该接口的实现 com.alipay.xxxx.MockRpcImpl。

说明：

- 方法定义中的入参尽量定义为 VO，后期添加参数，就可以在 VO 中添加参数，而不改变方法的声明格式。
- 服务接口定义的相关规范，请参见 业务接口定义规范 。

定义 operationType

在服务接口的方法上添加 @OperationType 注解，定义发布服务的接口名称。@OperationType 有三个参数成员：

- value：RPC 服务的唯一标识，定义规则为 组织.产品域.产品.子产品.操作。
- name：接口中文名称。
- desc：接口描述。

说明：

- value 在网关为全局唯一，尽量定义详细，否则可能会和其他业务方的 value 值一样，导致无法注册服务。
- 为便于维护，请务必填写完整 @OperationType 的三个字段。

样例：

```
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
}
```

注册 API 服务

通过网关提供的 SPI 包，将定义好的 API 服务注册到指定的注册中心。

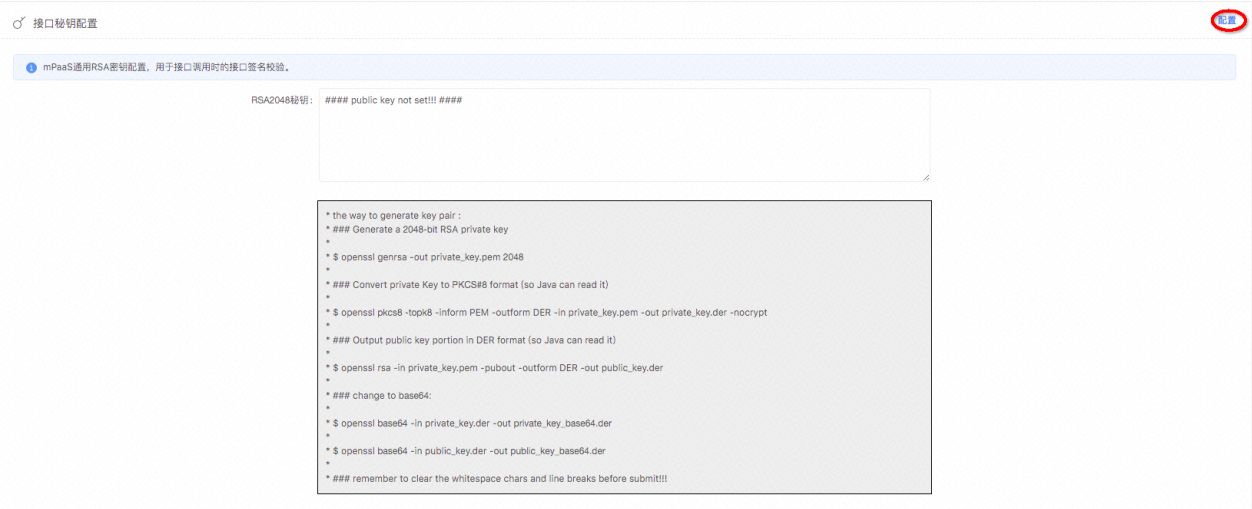
注册 MPC API 服务

注册 MPC API 服务需要如下参数：

- registryUrl： 该值为注册中心的地址，共享式金融科技注册中心地址为 116.62.81.246。
- appName：该值为业务方的应用名，与 API 分组名相同。
- workspaceId：应用所处环境的 workspaceId。
- projectName：应用所属租户的 projectName，与 API 分组里的 Project Name 相同。
- privateKeyPath：存放 RSA 私钥的 ClassPath，与 mpaaschannel 建立连接时用于校验合法性。
    - **推荐**: 放置在 /META-INF/config/rsa-mpc-pri-key-{env}.der 中，{env} 为不同的环境，如 dev、sit、prod 等。

配置公钥

前往对应环境的控制台，从左侧导航栏点击 **接口密钥** > **配置** ，配置私钥对应的 RSA 公钥。



RSA 公私钥的方法如下，其中公钥在控制台上配置，私钥文件在后端应用的 ${privateKeyPath} 中配置：

```
* 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:
*
* ## 生成的私钥，后端应用中配置
* $ openssl base64 -in private_key.der -out private_key_base64.der
*
* ## 生成的公钥，在控制台 接口密钥 中配置
* $ openssl base64 -in public_key.der -out public_key_base64.der
*
* ### remember to clear the whitespace chars and line breaks before submit!!!
```

spring 方式

在对应 bundle 的 spring 配置文件中，声明定义好的服务的 spring bean。

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

在对应 bundle 的 spring 配置文件中，声明暴露服务的 starter bean。

MpcServiceStarter：该接口会将所有带有 OperationType 的 bean 通过 mpaaschannel 协议注册到指定的注册中心。

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

spring-boot 方式

spring-boot 和 spring 本质上一样，只是注册的方式改为注解的方式，不用配置 xml 文件。

通过注解的方式，将定义的服务注册成 bean：

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

以注解的方式，定义暴露服务的 starter：

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

配置 MPC 日志

为了便于排查问题，可酌情配置 MPC 相关日志，下面以 log4j 配置为例：

```
<!-- [MPC Logger] tenant link，记录建联，settings 信息 -->
<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] 记录一个流相关的数据 (包括一对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 日志 -->
<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 日志 -->
<logger name="STREAM-DATA-DIGEST"additivity="false">
<level value="INFO"/>
<appender-ref ref="MPC-STREAM-DATA-APPENDER"/>
<appender-ref ref="ERROR-APPENDER"/>
</logger>
```

注册 Dubbo API 服务

注册 Dubbo API 服务需要如下参数：

- registryUrl：该值为注册中心的地址。
- appName：该值为业务方的应用名，与 API 分组名相同。

spring 方式

在对应 bundle 的 spring 配置文件中，声明定义好的服务的 spring bean：

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

在对应 bundle 的 spring 配置文件中，声明暴露服务的 starter bean —DubboServiceStarter，该接口会将所有带有 OperationType 的 bean 通过 Dubbo 协议注册到指定的注册中心。

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

spring-boot 方式

spring-boot 和 spring 本质上一样，只是注册的方式改为注解的方式，不用配置 xml 文件。

通过注解的方式，将定义的服务注册成 bean：

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

以注解的方式，定义暴露服务的 starter:

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

注册 HRPC API 服务

注册 HRPC API 服务需要如下几个参数：

- registryUrl：该值为注册中心的地址，必填。
- appName：该值为业务方的应用名，必填。
- serverPort：该值为 HRPC 服务监听端口，选填，默认 7079。

spring 方式

在对应 bundle 的 spring 配置文件中，声明定义好的服务的 spring bean：

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

在对应 bundle 的 spring 配置文件中，声明暴露服务的 starter bean。

HRpcServiceStarter：该接口会将所有带有 @OperationType 的 bean 通过 HRPC 协议注册到指定的注册中心。

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

spring-boot 方式

spring-boot 和 spring 本质上一样，只是注册的方式改为注解的方式，不用配置 xml 文件。

通过注解的方式，将定义的服务注册成 bean：

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

以注解的方式，定义暴露服务的 starter：

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

说明：HRPC 使用的注册中心为 ZK，地址可以填一个或者多个，用逗号分割。如 “11.163.193.240:2181” 或 “11.163.193.240:2181,11.163.193.230:2181” 。

注册 TR API 服务

注册 TR API 服务需要如下 1 个参数：

- appName：该值为业务方的应用名，必填。

spring 方式

在对应 bundle 的 spring 配置文件中，声明定义好的服务的 spring bean：

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

在对应 bundle 的 spring 配置文件中，声明暴露服务的 starter bean。

SofaServiceStarter：该接口会将所有带有 @OperationType 的 bean 通过 TR 协议暴露给网关调用。

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

spring-boot 方式

spring-boot 和 spring 本质上一样，只是注册的方式改为注解的方式，不用配置 xml 文件。

通过注解的方式，将定义的服务注册成 bean：

```
@Service
```

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

以注解的方式，定义暴露服务的 starter：

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

结果

完成上述步骤后，就可以通过在网关进行一系列的操作，将定义的 API 服务对客户端进行暴露。具体请参见 注册 API 。

5.3 网关辅助类使用说明

实现拦截器功能

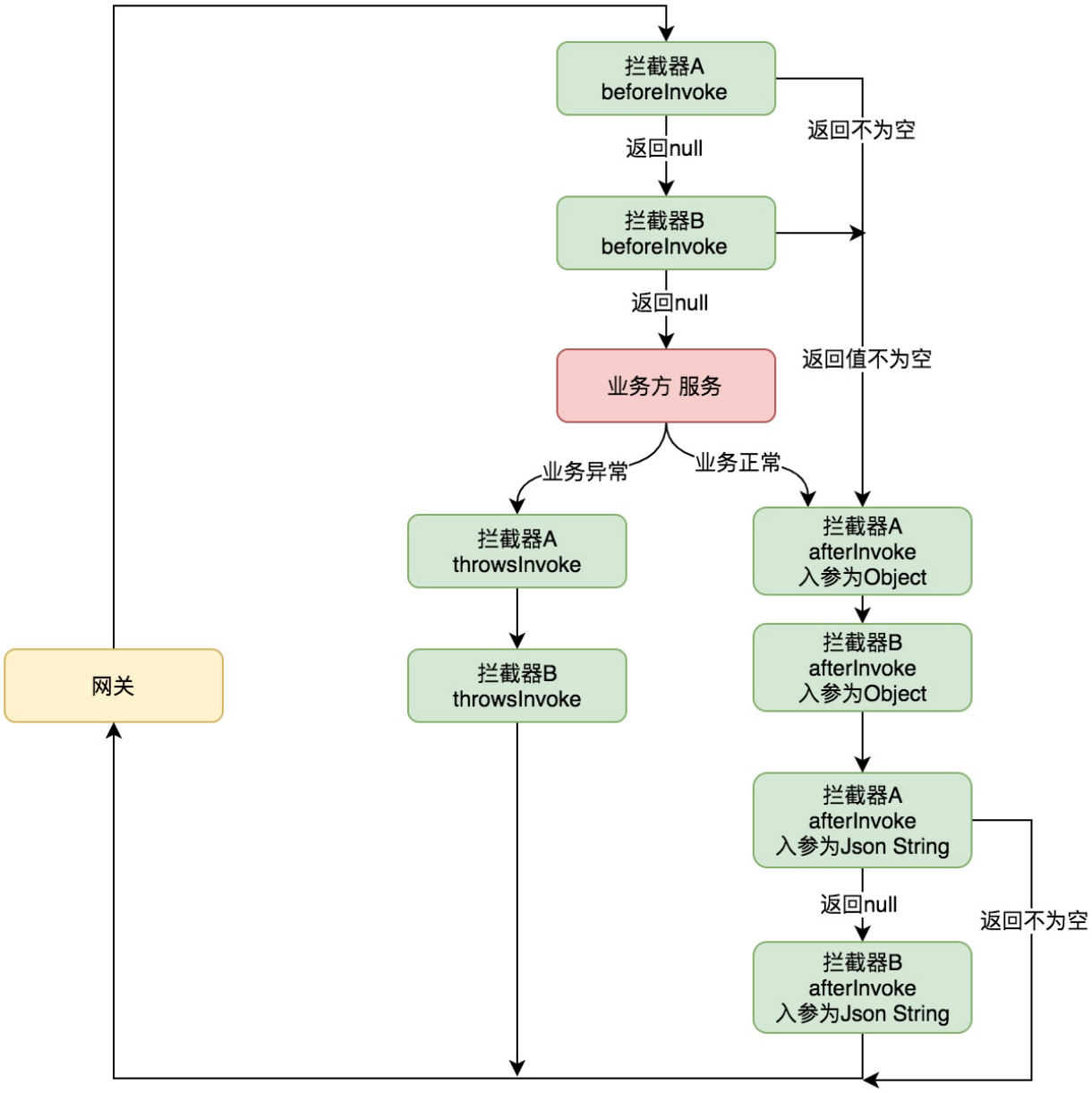
> 拦截器只适用于非 HTTP 类型服务。

说明

mobilegw-unify-spi-adapter.jar 实际上是通过 Java 的反射调用业务方法，即OperatioinType 所指定的方法。在方法调用的过程中，业务方可以实现 SPI 包中定义的拦截器，从而实现扩展。

网关的 SPI 包定义了两个拦截器：AbstractMobileServiceInterceptor 抽象类和 MobileServiceInterceptor 接口。

MobileServiceInterceptor 主要提供了四个方法，分别是beforeInvoke、afterInvoke（入参为业务返回的 Object）、afterInvoke（入参为 Object 转成的 JSON string）、throwsInvoke，getOrder。



如上图所示，拦截器主要在以下几个点进行拦截：

方法调用前：即 beforeInvoke方法，该方法有返回值。一旦该方法的返回值不为空，那么网关认定拦截成功，将会跳过剩余拦截器的 beforeInvoke 方法，同时跳过调用业务方的方法，直接进入拦截器的 afterInvoke 方法。

方法调用后： 即 afterInvoke 方法。afterInvoke 有两种，一种入参是Object，即业务方返回的对象，该方法没有返回值，所有的拦截器的该方法都会执行；另一种入参是JSON string，该方法可以改变传入的 JSON 数据并返回。一旦返回值不为空，那么网关认定拦截成功，后续的拦截器将被忽略。

方法出现异常：即 throwsInvoke 方法。该方法没有返回值，所有拦截器的该方法都会被执行。在业务方出现异常时会被调用。

MobileServiceInterceptor 继承了框架的 Ordered 接口，因此，业务方实现的拦截器还可以通过实现 getOrder方法

指定执行顺序，设置的数值越小，执行的优先级越高；设置的数值越大，执行的优先级越低。

示例

编码自己的拦截器类，继承 AbstractMobileServiceInterceptor 类，或者实现MobileServiceInterceptor 接口。

```
public class MyInterceptor implements MobileServiceInterceptor {

/*
参数说明
method：即业务方的方法（@OperatioinType 定义的方法）
args: 一个对象数组，即业务方方法的传入参数，传入参数个数即等于数组大小。
使用时业务方根据需要进行类型转换。
bean： 即业务方的接口实例。
返回值说明：
Object：可以在拦截器中返回数据，一旦返回值不为空，则网关认为已被拦截，就不会再调用业务方法
同时，直接跳过其他拦截器的 beforeInvoke 方法，执行拦截器中的 afterInvoke 方法。
*/

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

/*
*参数说明
*returnValue: 业务方法返回的对象
* 其它参数同上
*/
@Override
public void afterInvoke(Object returnValue, Method method, Object[] args, Object target) {
//注意：这里入参是业务方返回的 Object
}

@Override
public String afterInvoke(String returnJsonValue, Method method, Object[] args, Object target) {
//注意：这里入参是由业务方返回的 object，转换而成的 JSON 格式的 string
//可以返回新的 JSON 数据
return null;
}

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

@Override
public int getOrder() {
//最高级（数值最小）和最低级（数值最大）。
return 0;
}
}
```

发布实现的类 MyInterceptor，成为 Bean。

- spring-boot：直接在该类加注解 @service。

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

- spring：在配置的 xml 文件声明。

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

MobileRpcHolder 辅助类

说明

MobileRpcHolder 是 mobilegw-unify-spi-adapter.jar 中提供的一个静态辅助类，该类定义了一个请求过程中的相关信息，最主要的定义如下：

```
Map<String, String> session 保存请求的 session
Map<String, String> header 保存请求的头部相关信息
Map<String, String> context 保存网关调用的上下文信息
String operationType 保存此次请求的 operationType
```

在业务方的服务（即 OperationType）被调用之前，SPI 服务会根据网关转发的请求 MobileRpcRequest 去设置 MobileRpcHolder 这些信息。在调用业务方服务之后这些信息会被清除。

MobileRpcHolder 的生命周期为整个服务的调用过程，调用后清除。

业务方也可以根据需要去设置这些信息，这些信息会在调用业务的服务过程中一直存在，在调用过程中业务服务可以获取这些信息。具体的设置可以通过拦截器，在方法调用前后动态地修改 MobileRpcHolder 中保存的信息。

以下通过例子说明 MobileRpcHolder 如何使用。

示例

这里以修改和获取 session 为例。

修改 session
创建拦截器，具体过程看上面的拦截器例子。以下以在方法调用前为例：

```
@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);
}
```

这样就能修改 MobileRpcHolder 中的 session 信息。

获取 session业务方在自己定义的服务中可以获取 session 信息。

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

其他信息（如 header、context）的修改和获取跟上面的一样。

```
// 获取 header 所有信息
Map<String,String> headers = MobileRpcHolder.getHeaders();
// 这里上下文信息指的是请求中的上下文信息
Map<String,String> context = MobileRpcHolder.getRequestCtx();
// 获取 OperationType
String opt = MobileRpcHolder.getOperationType();
```

网关错误码使用

说明

网关有自己的一套错误码规范。详见 网关结果码说明 。

这里需要注意 BizException 6666，这个错误是业务方出现异常后，网关会抛出的异常。如果业务方想在具体出错时，返回其他错误码的需求。业务方可以通过抛出RpcException(ResultEnum resultCode)来控制 RPC 层的错误，比如 resultCode = 1001，会返回给客户端没有权限访问。

示例

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

自定义错误码

如果业务方想使用自定义错误码，那么在调用业务方法时不能往外抛异常。

业务方法只要出现异常，就会返回状态码 6666。同时客户端收到该状态码，即认为服务出错，不会去解析业务返回的数据。客户端只有在接收到 1000 状态码时，才会去解析返回的数据。

因此，具体做法是，服务端和客户端约定好具体的错误码，然后在调用业务方法时 catch 掉所有的异常，将自定义错误码放在返回的数据中。这样业务就算出异常，网关也会返回 1000 成功。同时客户端去解析返回的数据，提取自定义错误码。

6 使用控制台

6.1 API 分组

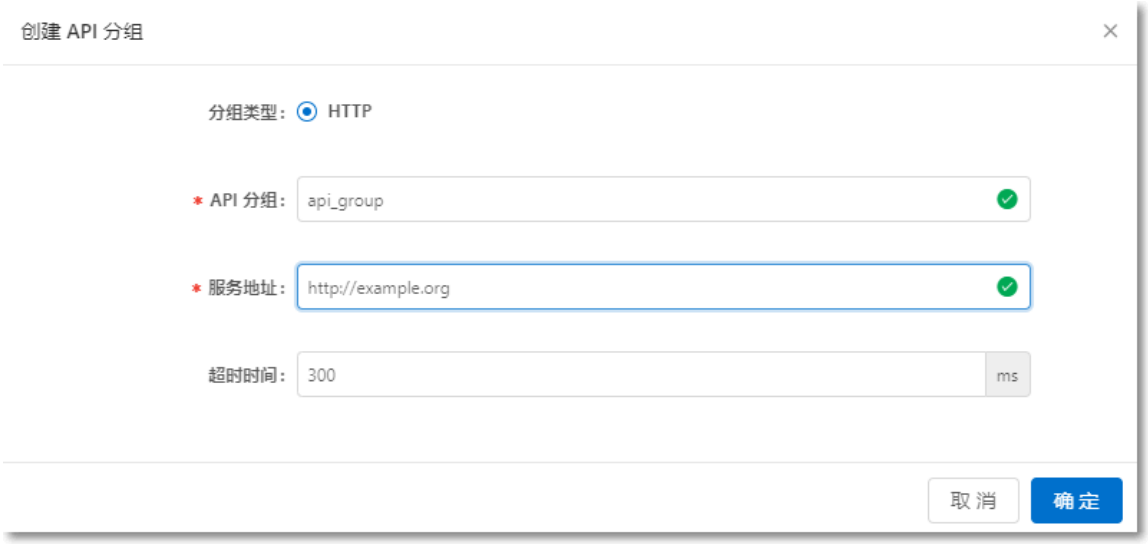
移动网关支持以下类型的服务接入：

- HTTP：符合 RESTful 风格的 HTTP 服务。
- DUBBO：**仅专有云可配置**，Apache Dubbo 服务。
- TR：**仅专有云可配置**，蚂蚁金服 RPC 框架。

创建分组

1. 选择 **API 分组** 选项卡进入 API 分组列表页。

   点击 **创建 API 分组** 按钮，在弹出的对话框中填写表单信息。



   - 公共表单项：
     - **分组类型**：必填，支持 HTTP、Dubbo、TR；其中，Dubbo、TR 仅在专有云中可用。
     - **API 分组**：必填，提供服务的业务系统的英文名称。
     - **超时时间**：选填，发送请求至业务系统时的超时时间，单位毫秒，默认值：3000 ms。
   - 其他表单项：
     - **服务地址**：HTTP 必填，业务系统的 HTTP/HTTPS URL。
     - **注册中心**：Dubbo 必填，支持 ZK 集群或者直连。
     - **直连地址**：TR 选填，满足直连需求。

   填写完整后，点击 **确定** 按钮提交。

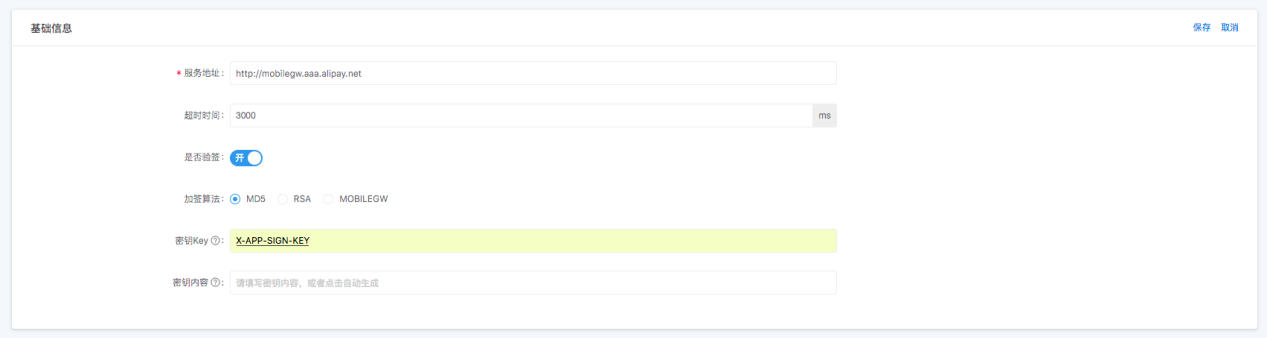
配置分组

您可根据不同的分组类型，完成相应的 API 分组配置：

- 配置 HTTP 分组
- 配置 Dubbo 分组（仅专有云可用）
- 配置 TR 分组（仅专有云可用）

配置 HTTP 分组

在分组列表中，找到类型为 **HTTP** 的分组，在其右侧的 **操作** 列中点击 **详情**，进入 HTTP API 分组详情页面。点击右上方的 **修改** 可配置分组。HTTP 分组的配置项如下：



- **服务地址**：HTTP 服务的 URL 地址。
- **超时时间**：单位毫秒，默认 3000 ms。
- **是否验签**：业务系统如需验证调用者的身份，请开启该项。如何验证参见 后端签名校验说明 。
- **加签算法**：生成签名的算法，公有云支持 MD5 和 RSA 算法，专有云还支持 MOBILEGW 的签名算法。
- **密钥 Key**：后端签名使用的密钥 Key，可以自定义。
- **密钥内容**：后端签名使用的密钥 Value。
  - 当加签算法是 MD5 时，可以自定义。
  - 当加签算法是 RSA 时，为移动网关的公钥。
  - 当加签算法是 MOBELGW 时，请向域内网关维护人员申请。

配置 Dubbo 分组

说明：配置 Dubbo 分组仅在专有云中可用。

在分组列表中，找到类型为 **Dubbo** 的分组，在其右侧的 **操作** 列中点击 **详情**，进入 Dubbo API 分组详情页面。点击右上方的 **修改** 可配置分组。Dubbo 分组的配置项如下：

- **注册中心**：Dubbo 服务的配置中心地址。
- **超时时间**：单位毫秒，默认 3000 ms。

配置 TR 分组

说明：配置 TR 分组仅在专有云中可用。

在分组列表中，找到类型为 **TR** 的分组，在其右侧的 **操作** 列中点击 **详情**，进入 TR API 分组详情页面。点击右上方的 **修改** 可配置分组。TR 分组的配置项如下：

- **直连地址**：TR 服务的直连地址，由 IP 和端口组成，端口不指定时默认为 12200。
- **超时时间**：单位毫秒，默认 3000 ms。

6.2 API 管理

6.2.1 注册 API

要注册 API， 点击 **API 管理** 选项卡进入 API 列表页。在该页面点击 **添加 API** 按钮，在弹出的对话框中添加 API。

您可以添加以下类型的 API 服务。根据不同类型的 API 服务，您需要进行相应的操作：

- HTTP API
- Dubbo API (仅
- TR API (仅

添加 HTTP API

登录 mPaaS 控制台，完成以下步骤注册 HTTP API:

1. 在左侧导航栏，点击 **移动网关**。
2. 在 **API 管理** 标签页，点击 **创建 API**。
3. 勾选 **HTTP** API 类型，将 **添加方式** 切换到 **手动** 模式。
4. 在 **operationType** 栏，输入值，点击 **确定** 提交。
    - operationType 为当前环境和 APP 下 API 服务的唯一标识。
    - operationType 定义规则：组织.产品域.产品.子产品.操作。

添加 Dubbo API

登录 mPaaS 控制台，完成以下步骤添加 Dubbo API:

1. 在左侧导航栏，点击 **移动网关**。
2. 在 **API 管理** 标签页，点击 **创建 API**。
3. 勾选 **Dubbo** 的 API 类型。
4. 选择对应的 API 分组。在 API 分组中，从获取到的 API 列表中选择需要注册的 API 服务。
5. 点击 **确定** 提交。

添加 TR API

登录 mPaaS 控制台，完成以下步骤自动拉取 TR API:

1. 在左侧导航栏，点击 **移动网关**。

2. 在 **API 管理** 标签页，点击 **创建 API**。
3. 勾选 **TR** 的 API 类型。
4. 选择对应的 API 分组。在 API 分组中，从获取到的 API 列表中选择需要注册的 API 服务。
5. 点击 **确定** 提交。

6.2.2 配置 API

注册 API 服务后，您需要进行相关配置才能使用 API 服务，尤其是 HTTP 类型的 API 服务。只有 API 服务状态为 **开通** 才能被调用。您需要手动开通 HTTP 类型的 API。

关于此任务

API 包括以下类型的参数配置：

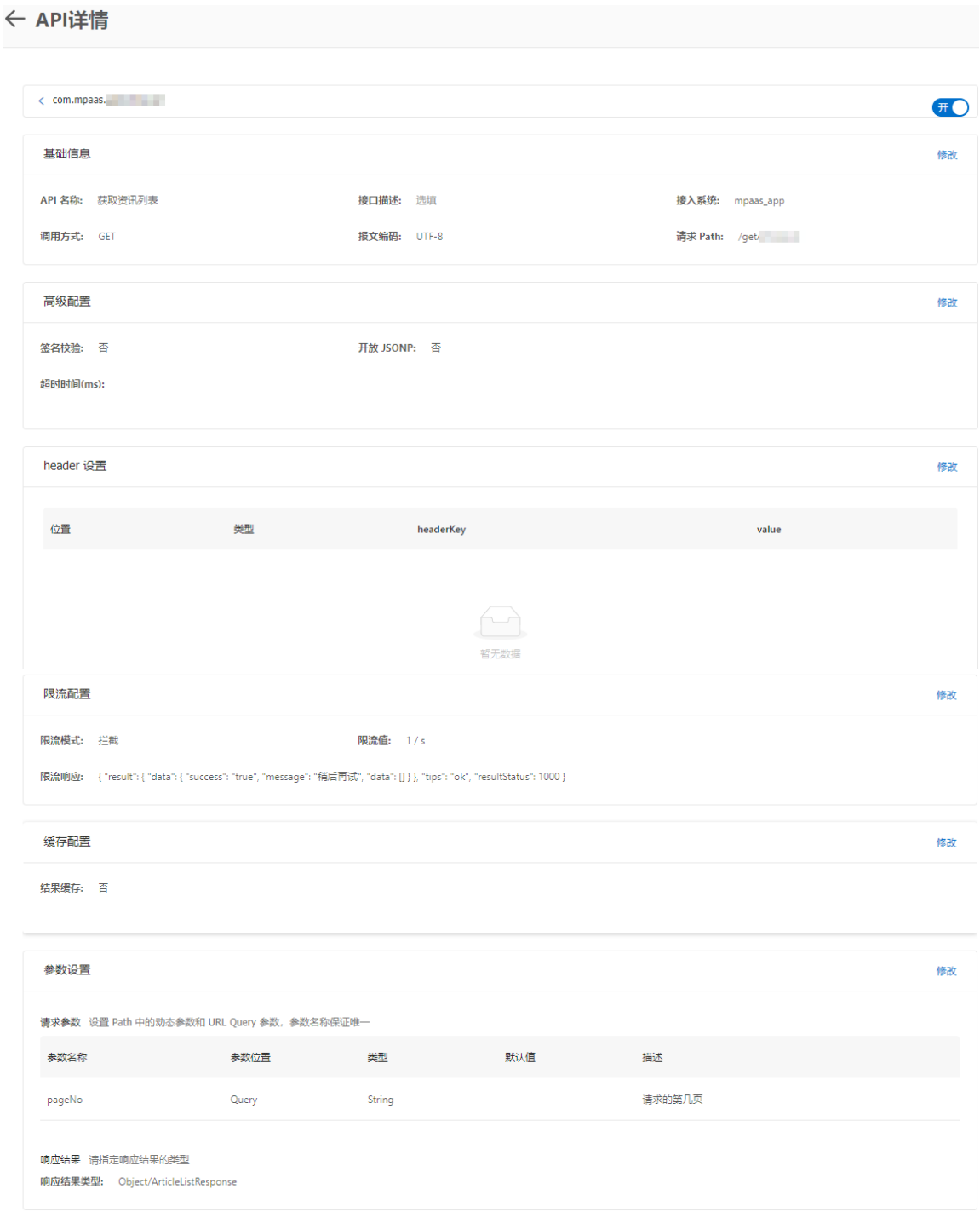
- 基础信息 ：API 名称、接口描述、接入系统等，不同类型的 API 服务还具备不同的属性。
- 高级配置 ：签名校验、ETag 缓存、超时时间。
- header 设置 ：网关支持为所有请求添加或者删除 header。
- 限流配置 ：对 API 调用进行限流配置。
- 缓存配置 ：缓存 API 的响应，减轻业务系统压力。
- 参数设置 ：请求参数设置、响应设置等，此项配置为 HTTP 类型的 API 服务特有。

欲了解详细的参数介绍及配置规则，点击以上配置类型名称。

操作步骤

要配置 API，完成以下步骤：

1. 登录 mPaaS 控制台，在左侧导航栏，点击 **移动网关**。
2. 在 API 列表中，选择要配置的 API 名称，点击操作列中的 **配置**，进入 API 详情页面。
3. 在以下 API 不同的配置区域，点击 **修改** 进行相应参数的编辑。具体的配置方法，参见下方的 配置信息 。



4. 点击 **保存** 完成配置。

配置信息

基础信息

根据不同类型的 API 服务，编辑相应的参数值：

HTTP API

- **API 名称**：必填，该 API 的接口名称，方便后续维护。

- **接口描述**：选填，更详细的 API 描述。
- **接入系统**：必填，API 所属的业务系统。
- **请求 Path**：必填，URL Path，可使用 ${} 包含 path 参数，比如：/pets/${id}。
- **请求方式**：必填，支持 GET、POST、PUT、DELETE 和 HEAD。
- **报文编码**：必填，UTF-8 或 GBK 格式。

MPC API

- **API 名称**：必填，该 API 的接口名称，方便后续维护。
- **接口描述**：选填，更详细的 API 描述。
- **接入系统**：API 所属的业务系统。
- **接口方法**：API 服务端方法。
- **接口名称**：API 服务端接口。

Dubbo API

- **API 名称**：必填，该 API 的接口名称，方便后续维护。
- **接口描述**：选填，更详细的 API 描述。
- **接入系统**：API 所属的业务系统。
- **接口方法**：API 服务端方法。
- **接口名称**：API 服务端接口。

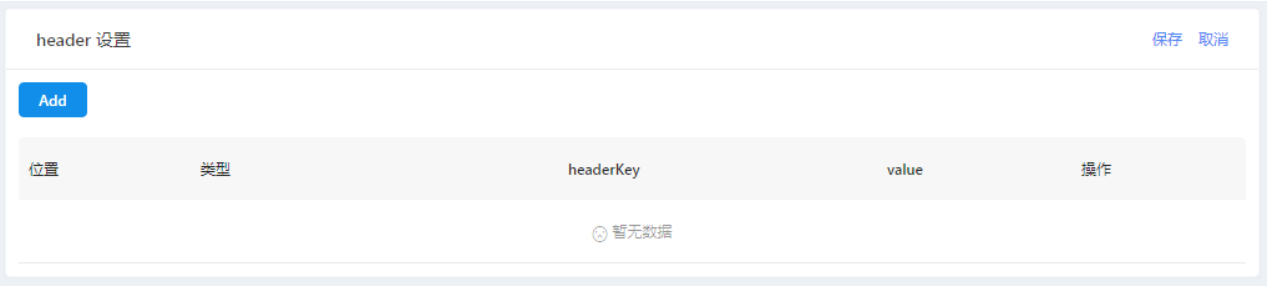
- TR API
  - **API 名称**：必填，该 API 的接口名称，方便后续维护。
  - **接口描述**：选填，更详细的 API 描述。
  - **接入系统**：API 所属的业务系统。
  - **接口方法**：API 服务端方法。
  - **接口名称**：API 服务端接口。

高级配置

- **签名校验**：选择是否开启签名校验。若开启，将对客户端请求的签名进行校验。
- **超时时间**：设置服务超时时间，单位毫秒（ms）。超时优先级：接口超时设置 > 系统超时设置 > 默认 3000 ms。
- **API 授权**：选择 API 授权规则验证请求，通过则转发至后端系统，不通过则返回 2000 错误码。授权规则配置参见 API 授权 。

header 设置

网关支持为所有请求添加或者删除 header。在 **header 设置** 区域，点击 **修改** 进入打开编辑模式后，即可通过点击 **Add** 可以添加一条规则。每条规则包含 **位置**、**类型**、**headerKey**、**value** 和 **操作** 五个属性。



- 位置：可选 request header 或 response header。
  - 添加 request header 会自动在 request 中增加 header，业务可以通过 MobileRpcHolder 获取该 header。
  - 添加 response header 会自动在 response 中增加 header，客户端可以从 response 中获取该 header。
- 类型：可选增加（add）或删除（delete）。
  - 增加（add）：添加一个新的 header，如果原请求已有 header，则会被新增的 header 覆盖。
  - 删除（delete）：删除一个 header。删除 header 可以配置 value 也可以不配置。如果配置 value，那么只有在 vulue 匹配时才会删除。
- headerKey：headerKey 可以为符合 RFC 定义的任意字符串，除 http 协议中的特有 header 之外，比如 host、content-Type 等；也不可以是 mpaasgw 中特有 header，比如 operation-Type 等。
- value：可以为任意字符串。
- 操作：删除当前 header 规则。

说明：定义 HTTP header 时，请不要使用下划线“_”。

限流配置

限流配置包括限流模式、限流值、限流响应：

限流模式

- **关闭**：不限制 API 调用。
- **拦截**：当调用频次超过限流值，拦截请求。

限流值

根据业务需求设置合理的限流阈值（单位：秒）。限流模式为拦截且超过此值时，请求会被限流。

限流响应

限流默认的响应为：{"resultStatus":1002,"tips":"顾客太多，客官请稍候"}
如需定制限流响应，使用如下格式：

```
{
"result":"==此处为定制响应内容,请填写==",
"tips":"ok",
"resultStatus": 1000
}
```

其中，

- result 为定制的响应数据，JSON 格式。只有 resultStatus 为 1000 时，客户端才会取此字段处理。
- tips 为定制的限流提示。若 resultStatus 为 1002，会取此字段提示用户。
- resultStatus 为限流返回的结果码，具体含义请参见 网关结果码 。

缓存配置

缓存 API 的响应，减轻业务系统压力。具体配置参见 API 缓存 。

参数设置

说明：此部分适用于 HTTP 类型的 API 服务。通过自动导入方式创建的 API 无需配置参数。

请求参数设置

- **参数名称**：必填，参数的名称。

  说明：如果为 Path 参数，请保证名称与请求 Path 中的一致。例如，请求 Path 为 /pets/${id}，此时参数名称应为 id。
- **参数位置**：必填，参数位于 Path 或者 Query String 中。
- **类型**：必填，可选择的类型有 String、Int、Long、Float、Double、Boolean。
- **默认值**：选填，参数的默认值。
- **描述**：选填，参数描述。

响应结果设置
响应结果类型为基础类型或者自定义的数据模型。

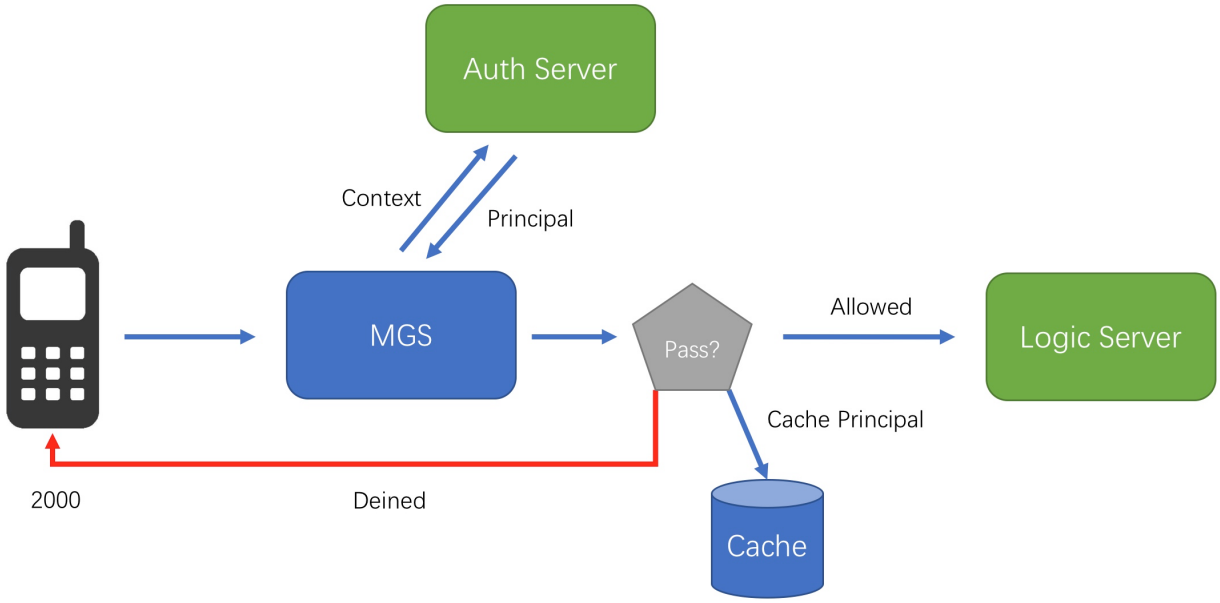
6.2.3 API 授权

了解 API 授权的使用场景。根据您的业务需求，开启 API 授权、配置授权规则，定义授权方接口，并将授权规则应用到 API。

功能介绍

API 授权功能允许业务在 MGS 上定义通用的 API 访问授权规则：

1. 创建授权 API A 并在网关管理中配置，然后到业务 API B 配置中做关联。
2. 客户端发起对后端业务 API B 的请求时，MGS 会根据 API 授权配置从该请求 Header 或 Cookie 中取出授权参数放到 Context 里然后调用业务 API B 关联的授权 API A，授权 API A Server 需要根据 Context 中的参数做业务权限校验。
3. 如校验合法，MGS 会将校验结果 Principal 添加在请求 Header 中，传递给后端业务 API B。如需缓存，MGS 会缓存校验结果 Principal，提高授权的性能。



使用场景

场景一

客户有分布式 session，登录后会产生会话 ID。授权过程如下：

1. 用户 A 请求登录接口，登录成功后，产生会话 ID 和会话信息保存到分布式缓存中，sessionId: {username:A, age:18, ...} ，并且下发 sessionId 到客户端。
2. 用户 A 请求一个需要登录授权的接口，网关从请求 Header 中获取 sessionId，发送给授权系统，授权系统根据 sesssionId 从分布式缓存中获取到用户信息，并且将 {username:A, age:18,...} 返回给网关。
3. 网关判断登录成功，将 {username:A, age:18,...} 添加在 Header 中，转发请求到后端的业务 Server。

场景二

客户端基于 HMAC 的授权方案，授权过程如下：

1. 用户 A 登录成功后，下发一个 token 到客户端，token=hmac(username+password) 。
2. 用户 A 请求一个需要登录授权的接口，网关从 Header 中获取 token，发送给授权系统，授权系统根据再算一遍 HMAC，如果符合则返回用户信息 {username:A, age:18,...} 给网关。
3. 网关判断登录成功，将 {username:A, age:18,...} 添加在请求 Header 中，转发请求到后端的业务 Server。

操作步骤

1. 配置授权规则

1. 登录 mPaaS 控制台。在左侧导航栏，点击 **移动网关**。

   点击 **网关管理** 标签，在 **API 授权** 下方，点击 **创建授权方** 或点击已存在的授权规则记录列表中操作列中的 **详情**，进入授权规则配置页面：

   - **授权方名称**：必填，授权规则的名称。
   - **授权方接口**：必填，用于验证请求授权情况的接口。
   - **授权缓存**：是否缓存授权的验证结果。
   - **缓存TTL**：验证结果的缓存存活时间。

     身份来源：如果点击 **添加来源字段**，填写用于授权的请求参数，表明请求身份的信息，由以下字段组成：

     - **位置**：参数所处位置，header 或者 cookie。
     - **字段**：参数名称。

       注意：如果实际 API 请求时的身份来源字段缺失，授权验证无法通过。

2. 定义授权方接口

如果后端系统提供的授权接口为 HTTP 类型，需要将授权 API 配置为 POST 方法。

在添加授权关系前，业务系统需要提前开发一个 Auth API。当 API 需要验证授权关系时，会调用 Auth API 进行授权校验。Auth API 的定义（请求和响应）遵循以下的标准：

AuthRequest

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

AuthResponse

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

接口示例

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

说明：

当验证授权的响应中 success 字段值为 true 时，网关会根据缓存策略缓存 principal 信息，然后将 principal 信息放入这次请求的 header 中，透传到后端业务系统中。没有principal也需要传个空 Map。

当验证授权的响应中 success 字段值为 false 时，网关会返回 2000 错误码，客户端需要根据 2000 做相应的操作，例如弹出登录框。

3. 使用授权规则

当授权规则配置后，可以在 API 配置页面中的 **高级配置** > **API 授权** 中选择对应的规则，为该 API 启用授权功能。

注意：要使用 API 授权，确保在 **网关管理** 页面，**API 授权** 功能开启：

1. 登录 mPaaS 控制台。在左侧导航栏，点击 **移动网关**。
2. 点击 **网关管理** 标签，确保 **API 授权** 按钮开启。

该 API 在请求后端系统前，会进行授权验证。通过则接受请求，网关将请求路由到后端系统。否则，请求会被拒绝，调用方将收到授权失败的错误响应。

6.2.4 API 限流

API 限流指对某一 API 的访问量进行限制，避免高峰期时后台服务器被压垮。

前置条件

要使用限流配置，确保在 **网关管理** 页面，API 限流 功能开启：

1. 登录 mPaaS 控制台。在左侧导航栏，点击 **移动网关**。
2. 点击 **网关管理** 标签，确保 **API 限流** 按钮开启。

关于此任务

限流配置包括限流模式、限流值、限流响应：

限流模式

- **关闭**：不限制 API 调用。
- **拦截**：当调用频次超过限流值，拦截请求。

限流值

根据业务需求设置合理的限流阈值（单位：秒）。限流模式为拦截且超过此值时，请求会被限流。

限流响应

限流默认的响应为：{"resultStatus":1002,"tips":"顾客太多，客官请稍候"}

如需定制限流响应，使用如下格式：

```
{
"result":"==此处为定制响应内容,请填写==",
"tips":"ok",
"resultStatus": 1000
}
```

其中，

- result 为定制的响应数据，JSON 格式。只有 resultStatus 为 1000 时，客户端才会取此字段处理。
- tips 为定制的限流提示。若 resultStatus 为 1002，会取此字段提示用户。
- resultStatus 为限流返回的结果码，具体含义请参见 网关结果码 。

6.2.5 API 缓存

配置 API 缓存信息，缓存 API 的响应，减轻业务系统压力。

关于此任务

API 缓存包含整个后端请求的响应，包括响应头和响应体，所以要求被缓存的 API 的响应头中排除状态类信息，比如 cookie 中的用户态数据。该缓存功能只适合缓存无状态数据。

后端业务系统可以通过在响应头中添加 Pragma: no-cache 告知网关不缓存该次响应。

操作步骤

1. 登录 mPaaS 控制台，在左侧导航栏，点击 **移动网关**。
2. 在 API 列表中，选择要配置的 API 名称，点击操作列中的 **配置**，进入 API 详情页面。
3. 点击 **缓存配置** 区域的 **修改** 按钮，配置如下规则：

- **结果缓存**：是否开启缓存。
- **缓存时间**：缓存的存活时间，单位：秒。
- **缓存键值**：用于缓存的键值表达式。点击 **修改** 定义缓存键值。在弹出的模态框中，填写缓存所需的主键，可以拖曳进行排序。关于键值语法的信息，参见本文的 键值语法 。

键值语法

支持的语法

当 API 请求到达网关后，网关会根据键值配置获取相应的数据，作为缓存的键值，其语法如下：

语法	描述
$	根对象，例如：$.bar
[num]	数组访问，其中 num 是数字。例如：$[0].bar.foos[1].name
.	属性访问，例如：$.bar
[‘key’]	属性访问，例如：$[‘bar’]
$.header	API 请求头对象，用于获取请求头中的字段，例如：$.header.remote_addr
$.cookie	API 请求 cookie 对象，用于获取 cookie 中的值，例如：$.cookie.session_id
$.http_body	后端为 HTTP 类型的请求体对象，用于获取请求体中的字段，例如：$.http_body.name
$.http_qs	后端为 HTTP 类型的请求参数对象，用于获取请求参数，例如：$.http_qs.name

代码示例

以如下请求数据为例，从请求报文中获取对象：

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

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

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

如下显示表达式的示例：

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

6.2.6 API 模拟

要使用 API 模拟（Mock），需先前往 **网关管理** > **功能开关** 页面打开 API Mock 开关。

操作步骤

完成以下操作步骤配置 API Mock：

1. 选择 **API 管理** 选项卡 > API 列表操作列的 **更多** > **API Mock**。

2. 在 **Mock 配置** 页面，完成以下参数配置：
   - **API Mock**：开启 API Mock 开关。
   - **命中规则**：当前支持 **百分比** 规则。
   - **规则配置**：具体的百分比数值，取值在 0 ~ 100 之间。
   - **Mock 数据**：Mock 的 API 响应数据。
     注意：Mock 数据格式如下：

```
{
"resultStatus": 1000,
"tips":"ok",
"result":"==此处为Mock的业务数据,请填写=="
}
```

     其中，
     - resultStatus为响应结果码，具体含义请参见 网关结果码 。
     - tips为响应提示。
     - result为定制的响应数据，JSON 格式。

3. 点击 **提交**。

6.2.7 同步 API

关于此任务

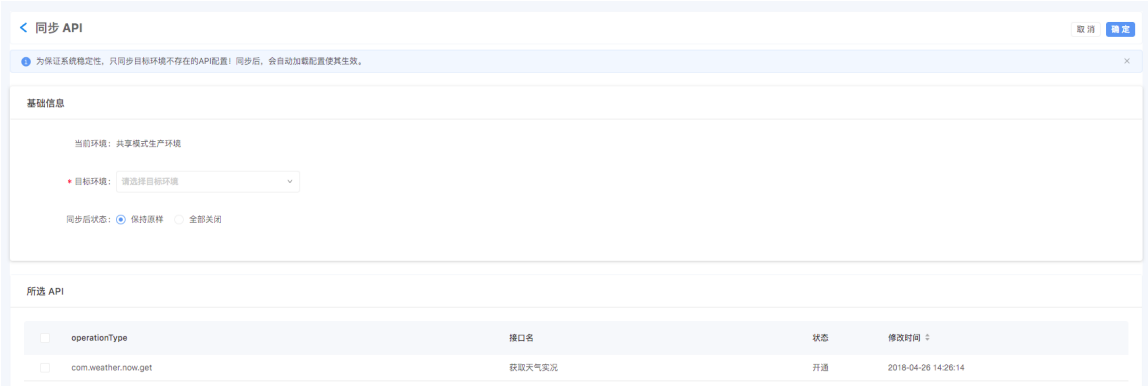
为选中的 API 选择目标环境和同步后状态，完成 API 同步。

说明：
- 为保证系统稳定性，只同步目标环境不存在的 API 配置。
- 同步后，会自动加载配置使其生效。

操作步骤

1. 登录 mPaaS 控制台，在左侧导航栏，点击 **移动网关**。

   在 **API 管理** 标签页中，点击 **更多** > **同步 API**。



3. 在 **基础信息** 栏，选择将当前环境的 API 配置同步到的 **目标环境**。
4. 选择 **同步后状态**：
    - **保持原样**：跟当前环境的 API 配置保持不变。
    - **全部关闭**：同步后的 API 状态置为关闭。
5. 在 **所选 API** 栏，选择需要同步的 API。
6. 点击 **确定** 按钮开始同步 API。

结果

同步成功后，当前页面会有 Alert 弹出，显示同步的结果。

6.2.8 导出及导入 API

关于此任务

- 导出 API：为便于将当前 API 配置应用于其他环境或其他应用，您可将 API 导出为 .txt 文件。
- 导入 API：将 API 配置导入并应用到当前环境。选择导入策略和 API 文件后，导入所需的 API。

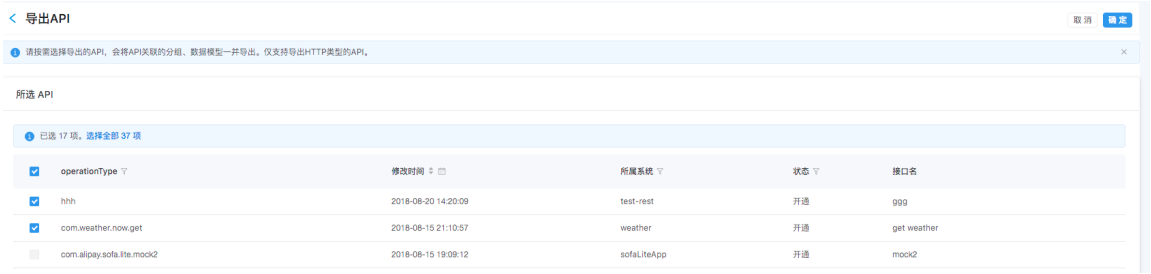
导出 API

按需选择要导出的 API，导出操作会将 API 关联的分组、数据模型一并导出。

说明：仅支持导出 HTTP 类型的 API。
操作步骤如下：

1. 登录 mPaaS 控制台，在左侧导航栏，点击 **移动网关**。

    在 **API 管理** 标签页中，点击 **更多** > **导出 API**。



3. 在 **所选 API** 栏，选择需要导出的 API。
4. 点击 **确定** 按钮开始导出 API。导出的 API 保存在一个 .txt 文件中。

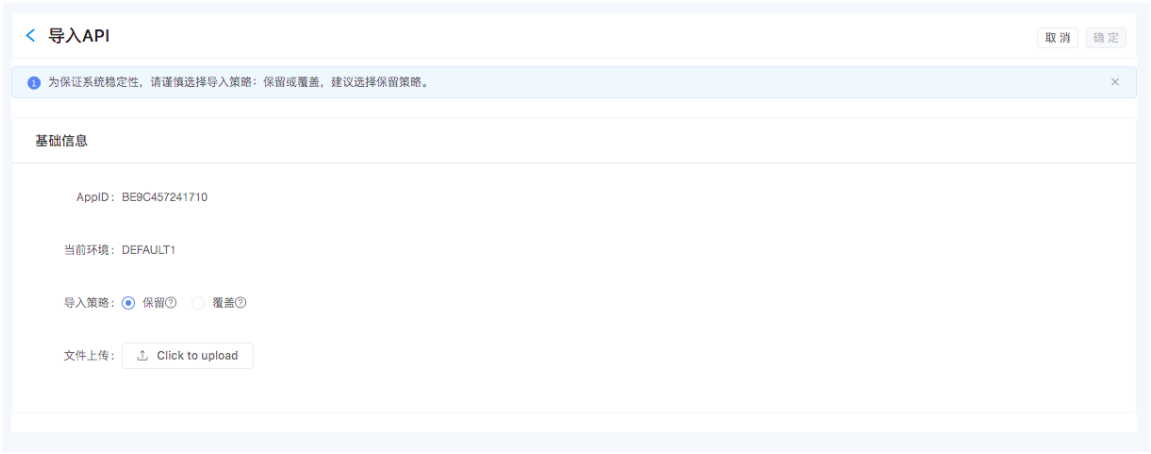
导入 API

为保证系统稳定性，在 **保留** 或 **覆盖** 中，建议选择 **保留** 作为导入策略。

操作步骤如下：

1. 登录 mPaaS 控制台，在左侧导航栏，点击 **移动网关**。

   在 **API 管理** 标签页中，点击 **更多** > **导入 API**。



3. 确认 AppID 和当前环境是否正确。
4. 选择 **导入策略**：当导入配置与已有配置存在冲突时所采用的策略。
   - **保留**：当导入的配置与已有配置冲突时，将保留已有配置、放弃导入配置。
   - **覆盖**：当导入的配置与已有配置冲突时，将采用导入配置、替换已有配置。
5. 点击 **文件上传**，选择要导入的 API 文件。
6. 点击 **确定** 按钮开始导入 API。导入成功后，会显示导入的结果。

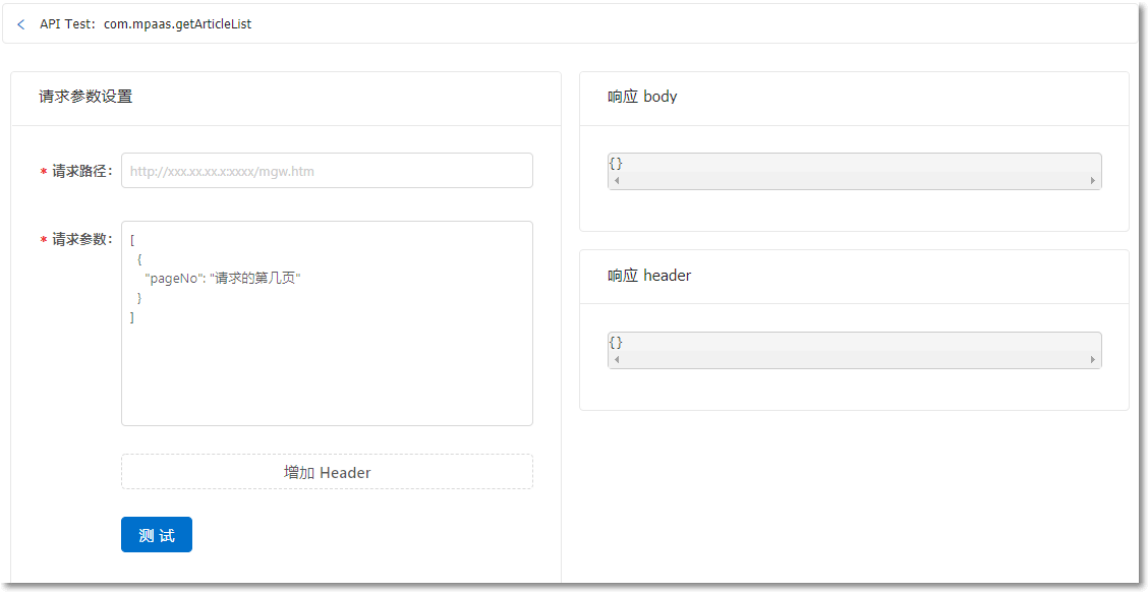
6.3 API 调用

6.3.1 API 测试

要进行 API 测试（Test），在 API Test 页面完成相关配置。

操作步骤

选择 **API 管理** 选项卡 > API 列表操作列 **更多** > **API Test**，打开如下页面：



在 **API Test** 页面中，完成以下配置：

- **请求路径**：网关地址。公有云的地址为 https://mpaas-mgs.aliyuncs.com/mgw.htm。

  注意：地址必须带有 /mgw.htm，否则请求会失败。

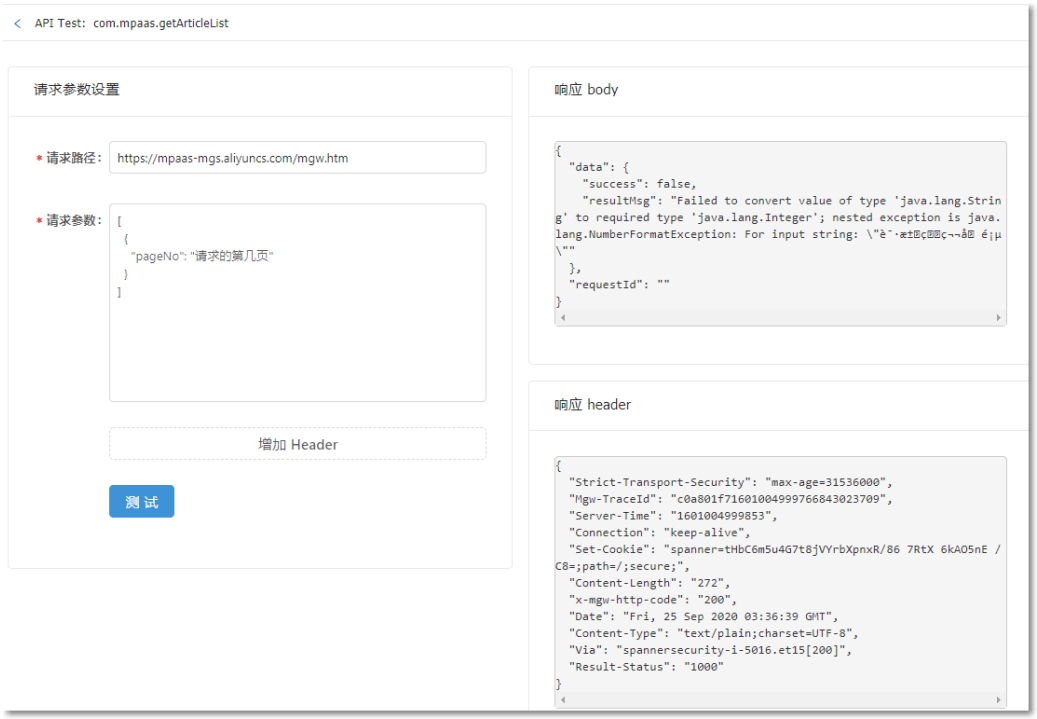
- **请求参数**：默认会模拟符合请求参数格式的数据，具体请按照业务含义调整。
- **Header**：您可以按需增加 Request Header。

点击 **测试** 按钮，右侧会得到请求执行的如下结果：

- **响应 body**：业务返回的响应数据。

  响应 header：包括网关约定以及业务返回的 Response Header。

  - Result-Status：具体参见 网关结果码 。
  - Mgw-TraceId：该请求的 TraceId，可以据此进行链路分析。



6.3.2 生成代码

移动网关提供生成 API 的客户端 SDK 功能。

关于此任务

只有 HTTP 类型的 API 支持单个代码生成。

操作步骤

1. 通过以下的操作方式打开 **客户端代码生成** 窗口：
   - 方式一：在 **API 分组** 标签页中，点击系统列表操作列中的 **生成代码** 按钮。
   - 方式二：在 **API 管理** 标签页中，点击列表上方的 **生成代码** 按钮。
   - 方式三：在 **API 管理** 标签页中，点击列表操作列中的 **更多** > **生成代码** 按钮。该方式用于生成单个 API 代码（非 API 分组），仅适用于 HTTP 类型。

2. 在 **客户端代码生成** 窗口，完成以下信息配置：
    - **API 分组**：选择需要生成 SDK 的 API 分组。
    - **Platform**：选择 Android、iOS 或 JS：
        - 若选择 **Android**，在 **PackageName** 中，填写客户端代码的包名；若不填，默认为 com.client.service。
        - 若选择 **iOS**，在 **Prefix** 中，填写唯一前缀；若不填，默认不加前缀。
3. 点击 **提交** 生成 API SDK 供客户端调用。

6.4 网关管理

6.4.1 网关管理功能介绍

网关管理包含以下操作：

功能开关

功能开关是全局的，可以根据需要暂时地开启或者关闭所有的 API 相关功能。

签名校验

对客户端到移动网关的请求进行验签，以验证调用者身份保证安全，默认 **打开**。

API 限流

对某一 API 的访问量进行限制，避免高峰期时后台服务器被压垮，默认 **关闭**。

API Mock

对某一 API 的返回值进行 Mock，以提供特定的响应结果，默认 **关闭**。

API 授权

在网关将客户端请求路由到后端业务系统之前，校验该次请求的合法性，验证通过才予以放行，默认 **关闭**。

具体配置参见 API 授权 。

数据加密

对客户端到移动网关的请求进行加密，确保数据在传输过程中的安全性，默认 **关闭**；目前支持的加密算法有 ECC 和 RSA，该功能需要与客户端配合使用。如果客户端数据加密方式与此处设置的方式不一致，网关将无法解析客户端请求。

具体配置参见 数据加密 。

CORS

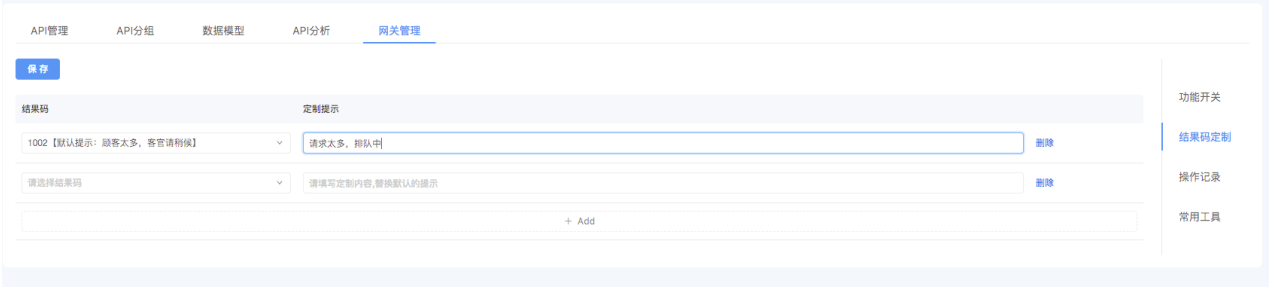
跨域资源共享，根据规则进行跨域访问控制。如果需要支持跨域请求，请配置该规则。

具体配置参见 跨域资源共享 。

结果码定制

网关结果码都有默认的提示文案，您也可以根据实际需求定制结果码提示。

选择 **网关管理** 选项卡，点击右侧 **结果码定制** 进入定制页面。

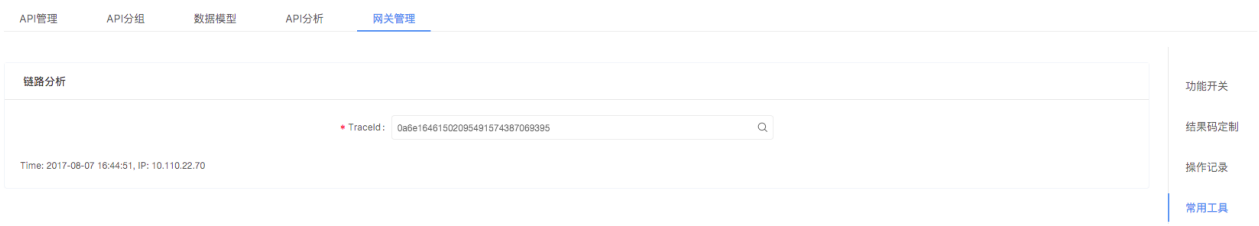


操作记录

操作记录 对配置人员在网关上的相关动作进行记录和展示，方便进行追溯。

常用工具

链路分析 可以对 TraceId 进行分析，解析出对应的时间和网关服务器。



6.4.2 数据加密

要进行数据加密，在服务端，您需要进行相关配置生成密钥；在客户端，根据不同的操作平台，完成相应的配置。

- 服务端配置
- 客户端配置：
    - Android 配置
    - iOS 配置

服务端配置

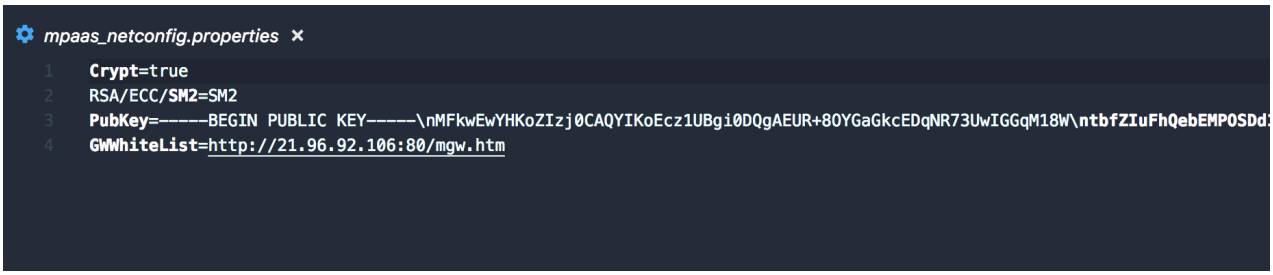
1. 登录 mPaaS 控制台，在左侧导航栏，点击 **移动网关**。
2. 选择 **网关管理** 选项卡，点击右侧的 **功能开关** 标签页。
3. 将 **数据加密** 状态切换至 **开**。
4. 在弹出的 **配置加密算法** 窗口，完成以下配置：
    - 加密算法：支持 ECC、RSA 和国密（SM2）。
    - 密钥内容：
        - 当加密算法为 ECC 或 国密时，填写私钥内容。
        - 当加密算法为 RSA 时，分别填写公私钥内容。

注意：有关加密算法的密钥生成方法，参见 密钥生成方法 。

客户端配置

Android 配置

在 assets 目录下新建 mpaas_netconfig.properties 文件，用于存放网络相关全局配置。



其中，

- Crypt：表示是否使用自加密，true 表示使用，false 表示关闭自加密功能。
- RSA/ECC/SM2：表示要使用的非对称加密算法，其值只能填充 RSA 或 ECC 或 SM2。
- PubKey：表示选择的非对称加密算法的公钥。
- GWWhiteList: 需要进行加密的网关，如果没有这个 key ，则所有的请求 **都不会加密**。

注意：由于 Android 中 properties 文件 value 需要在同一行，故填充公钥时请注意。

iOS 配置

iOS 端加密配置是从 info.plist 里面读取，如下图所示：

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

其中，

- mPaaSCrypt：加密配置的主 key，value 是 Dictionary 类型，里面包含了客户端加密所需设置的相关信息。
- Crypt：是否进行加密，value 是 Boolean 类型，YES 代表加密，NO 代表不加密。
- GWWhiteList: 需要进行加密的网关，如果没有这个 key ，则所有的请求 **都不会加密**
- RSA/ECC/SM2：非对称加密算法选择，value 是 String 类型，只能填 RSA 或 ECC 或 SM2。
- PubKey：非对称加密公钥。value 是 String 类型，与选择的非对称加密算法保持一致。

提醒：

- Crypt 设置为 NO 时，RPC 不进行加密，RSA/ECC/SM2 和 PubKey 的设置会忽略。
- Crypt 设置为 YES 时，RSA/ECC/SM2 和 PubKey 必须设置且不能为空字符，否则 Debug 时会中断言，程序直接退出。

    - RSA/ECC/SM2 和 PubKey 的设置必须一一对应：
    - 选择 RSA 算法，PubKey 填 RSA 对应的公钥。
    - 选择 ECC 算法，PubKey 填 ECC 对应的公钥。
    - 选择 SM2 算法，PubKey 填 SM2 对应的公钥。
- PubKey 格式必须携带 -----BEGIN PUBLIC KEY----- 及 -----END PUBLIC KEY-----，格式如下：

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

6.4.3 跨域资源共享

CORS

跨域访问是指请求一个与自身资源不同源（不同的域名、协议或端口）的资源。不同源可以是不同的域名、协议或端口。

浏览器出于安全考虑设置了同源策略，限制了从脚本内发起跨域请求。但在实际应用中，经常会发生跨域访问。为此，W3C 提供了一个标准的跨域解决方案：跨域资源共享（Cross-Origin Resource Sharing，CORS），支持安全的跨域请求和数据传输。

浏览器将 CORS 请求分为以下两类：

- 简单请求
- 预检请求 ：防止资源被本来没有权限的请求修改的保护机制。浏览器会在发送实际请求之前使用 OPTIONS 方法发送一个预检请求，从而获知服务端是否允许该跨域请求。服务端确认允许后，才会发起实际的 HTTP 请求。

简单请求

请求满足如下所有条件，为 **简单请求**：

- 请求方法是如下之一：
    - HEAD
    - GET
    - POST
- HTTP 头信息不超过以下几种字段：
    - Cache-Control
    - Content-Language

- Content-Type
- Expires
- Last-Modified
- Pragma
- DPR
- Downlink
- Save-Data
- Viewport-Width
- Width

- Content-Type 的值仅限下列几种：
  - text/plain
  - multipart/form-data
  - application/x-www-form-urlencoded

预检请求

不符合简单请求条件的请求，会在正式通信之前触发一个 OPTIONS 请求进行预检。这类请求为预检请求。

预检请求会在请求头中附带一些正式请求的信息给服务端，主要有：

- Origin：请求源信息。
- Access-Control-Request-Method：接下来的请求类型，如 POST、GET 等。
- Access-Control-Request-Headers：接下来的请求中包含的用户显式设置的 Header 列表。

服务端收到预检请求后，会根据上述附带的信息判断是否允许跨域，通过响应头返回对应的信息：

- Access-Control-Allow-Origin：允许跨域的 Origin 列表。
- Access-Control-Allow-Methods：允许跨域的方法列表。
- Access-Control-Allow-Headers：允许跨域的 Header 列表。
- Access-Control-Expose-Headers：允许暴露的 Header 列表。
- Access-Control-Max-Age：最大的浏览器缓存时间，单位：秒。
- Access-Control-Allow-Credentials：是否允许发送 Cookie。

浏览器会根据返回的 CORS 信息判断是否继续发送真实的请求。以上行为都是浏览器自动完成的，服务端只需要配置特定的 CORS 规则。

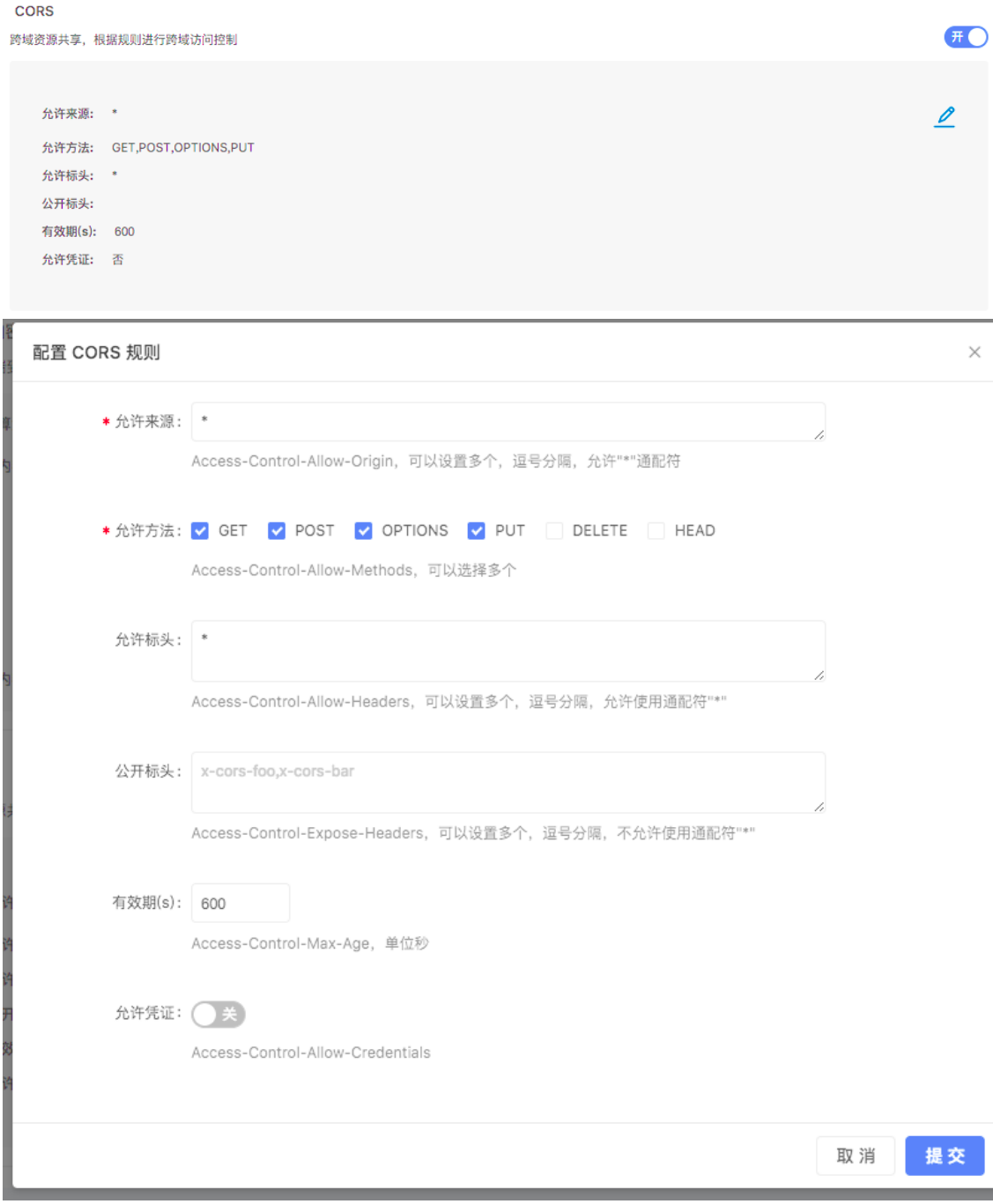
网关对 CORS 的支持

网关提供了配置 CORS 规则的功能，让业务方自行决定是否允许特定的跨域请求。该规则以 appId + workspaceId 维度配置。

配置 CORS

登录 mPaaS 控制台，完成以下步骤：

1. 在左侧导航栏，点击 **移动网关**。
2. 选择 **网关管理** 标签页，点击右侧的 **功能开关** 标签页，进行 CORS 配置。



开启 CORS 后，app 在该 workspace 下的所有 API 服务都将支持符合以下配置的跨域请求：

- 允许来源： Access-Control-Allow-Origin，可以设置多个，逗号分割，允许“*”通配符。
- 允许方法： Access-Control-Allow-Methods，可以选择多个。
- 允许标头： Access-Control-Allow-Headers，可以设置多个，逗号分割，允许“*”通配符。
- 公开标头： Access-Control-Expose-Headers：可以设置多个，逗号分割，不允许“*”通配符。

- 有效期： Access-Control-Max-Age，最大的浏览器缓存时间，单位：秒。
- 允许凭证： Access-Control-Allow-Credentials，是否允许发送 Cookie。

跨域请求

跨域的 API 请求要添加 X-CORS-${appId}-${workspaceId} 请求头。当预检请求到达网关后，网关解析 Access-Control-Request-Headers 中的 X-CORS-${appId}-${workspaceId} 获取 appId 和 workspaceId，再进一步获取对应的 CORS 配置。网关跨域请求的请求头中要包含如下内容：

- X-CORS-\${AppId}-\${WorskapceId}：一定要有此请求头，并用实际的 AppId 和 WorkspaceId 替换占位符内容；
- Operation-Type
- WorkspaceId
- AppId
- Content-Type
- Version

```
$.ajax({
url: 'http://${mpaasgw_host}/mgw.htm',// 请填写网关地址
headers: {
'X-CORS-${appId}-${workspaceId}':'1' // 一定要设置这个请求头
'Operation-Type':${operationType}, // 请填写 operationType
'AppId':${appId}, // 请填写 appId
'WorkspaceId':${worksapceId}, // 请填写 worksapceId
'Content-Type':'application/json',
'Version':'2.0',
},
type: 'POST',
dataType: 'json',
data: JSON.stringify(reqData),
success: function(data){}
});
}
```

说明：CORS 配置中的 **允许标头** 配置，根据实际情况添加或者设置 “*” 。

6.5 数据模型

作为业务人员，您可以将 API 服务的请求与响应定义成 **数据模型**，通过模型复用减少繁琐的参数设置。该功能用在 定义 HTTP 类型的 API 服务的参数，其他类型的 API 服务无需手动定义。

关于此任务

目前支持以下数据模型定义方式：

- 可视化编辑：逐条添加模型参数。
- 样本数据编辑：从样本数据解析出数据模型，推荐使用这种方式。

操作步骤

完成以下步骤配置数据模型：

1. 在移动网关主页，选择 **数据模型** 选项卡，进入数据模型列表页。

2. 点击 **添加数据模型** 按钮或者列表中的 **详情** 链接，添加或编辑数据模型定义：

   - **模型名称**：模型名称，以字母或下划线开头，由字母、下划线、数字组成。
   - **模型描述**：模型的描述信息。
   - **模型参数**：
     - **名称**：必填，模型中参数的名称。
     - **类型**：必填，可选择的类型有 String、Int、Long、Float、Double、Boolean、List 及已定义的数据模型。
     - **默认值**：选填，参数的默认值。
     - **描述**：选填，参数描述。

       注意：目前尚不支持 Map 类型。

3. 点击 **提交** 保存修改内容。

6.6 API 分析

在移动网关主页，选择 **API 分析** 选项卡进入 API 分析界面。API 分析界面提供以下几种 API 的统计数据：

- API 调用量：统计某一 API 分钟级的调用量。
- API 报错量：统计某一 API 分钟级的报错量。

- 平均调用耗时：统计某一 API 分钟级的平均调用耗时。
- API 调用同比：统计 API 当前时刻调用量相对昨天的同比。
- API 报错同比：统计 API 当前时刻报错量相对昨天的同比。
- API 耗时同比：统计 API 当前时刻平均耗时相对昨天的同比。

7 网关异常排查

单请求问题排查

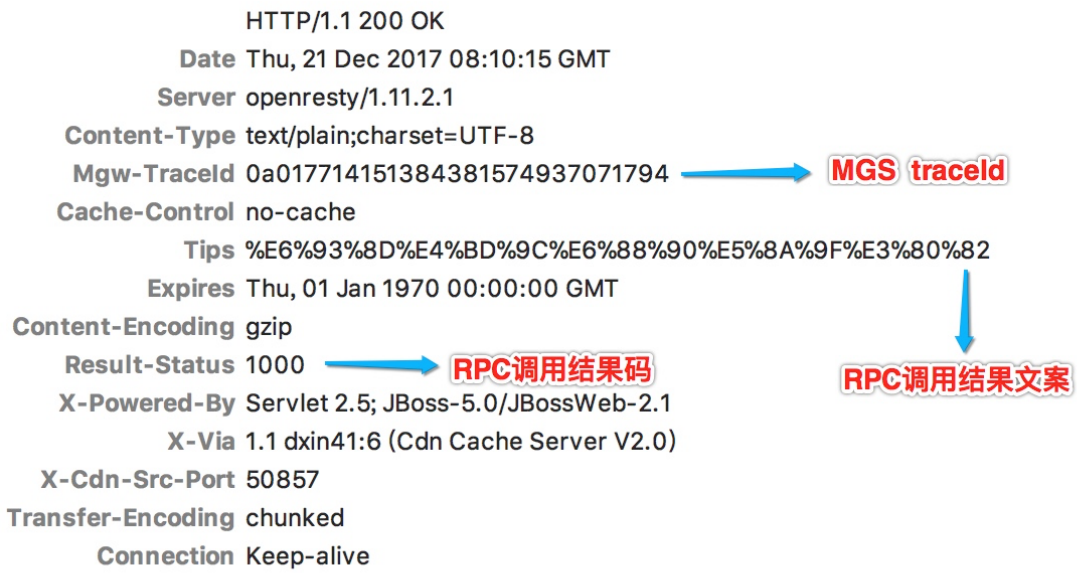
1. 客户端请求抓包

客户端抓包一般采用 Charles【推荐】或 Fiddler 工具，通过抓包工具，可以看到 RPC 请求的一些关键数据。

下面的是一个抓包的案例：

请求 Header 样例：

响应 Header 样例：



2. 根据 TraceId 查询 MGS 日志

1. 从响应 Header 中获取 Mgw-TraceId。
2. 进入 mPaaS 控制台的 **移动网关** > **网关管理** > **常用工具** > **链路分析** 页面，输入 TraceId 可以解析出处理该请求所在的 MGS 服务器 IP 和处理时间。

3. 通过 SSH 到 MGS 的服务器，根据 TraceId 查询请求相关的日志。

```
ssh -p2022 log@#ip# 帐号：log/mpaas123456
cd /home/admin/logs/gateway
grep #traceid# *.log
```

4. 根据 网关日志说明 和 网关结果码说明 分析日志。

集群 GREP 问题排查（仅适用于专有云）

有些时候，我们需要在 MGS 集群搜索某个日志。这时候，可以使用开源的 pssh 工具。

1. 下载 pssh 工具。

   从 Gamma 平台导出 MGS 所有服务器 IP 列表到 mgs_host.txt 文件中，如下：

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

3. 运行以下执行命令：pssh -i -h mgs_host.txt -A -P 'grep"xxxx"/home/admin/logs/gateway/xxx.log'。

8 常见问题

调用失败，如何排查？

参见 网关异常排查 。

API 返回的错误码是什么意思？

参见 网关结果码说明 。

如果引用了 okhttp 存在 okio 和 mpaas 有冲突该怎么解决？

您需要完成以下两步操作以解决该冲突：

1. 注掉 mpaas 的 wire 组件。

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

2. 使用公网提供的 wire 组件。

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

通过 JSAPI 调用 MGS RPC 接口向后端发送 POST 请求时，如何把参数放到POSTBODY 中。

MGS 正确配置好 POST BODY 及对应的数据模型后，通过 JSAPI 发送请求时需要把POSTBODY的内容作为 _requestBody 的值放在 requestData 参数中，参见下面的样例

```
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));
});
});
}
```

9 参考

9.1 网关结果码说明

本文对使用移动网关过程中出现的结果码进行说明，方便您进行问题排查。

网关侧结果码

- 1000 为 API 调用成功，其他都是失败的错误码。
- 1001-5999、7XXX 为网关错误。
  - 其中，7XXX 表示无线保镖验签或解密报错，具体参见 无线保镖结果码说明 进行排查。
  - 除结果码外，您还可以查看响应 Header 中的 Memo 和 tips 字段，以了解更多错误信息。
  - 专有云用户还可以通过网关服务器上的 ~/logs/gateway/gateway-error.log 日志查看详细错误信息。
- 当发生异常时，可以尝试通过网关异常排查进行排查。欲了解具体信息，参见 网关异常排查 。

结果码	描述	解释
1000	处理成功	网关 API 调用处理成功。
1001	拒绝访问	开启 数据加密 后，当 RPC 数据解密失败会异常。
1002	调用次数超过限额	开启 限流配置 后，当触发限流时会导致该异常。
1005	未授权	开启 API 授权 后，API 调用时授权校验失败。
2000	登录超时	开启授权校验功能，非登录状态会触发该异常。
3000	RPC 接口不存在或关闭	在当前 workspaceId 对应的环境下，appId 对应的移动应用没有配置该 operationType 的 API 服务，或者该 API 服务不处于 **开放** 状态。
3001	请求数据为空	客户端请求数据中的 requestData 为空。请检查客户端 RPC 是否正常，iOS 端需确认已 初始化网关服务 。
3002	数据格式有误	RPC 请求格式有问题。专有云用户可以在服务端日志 gateway-error.log 中查看详细信息。
3003	数据解密失败	数据解密失败。
40	服务请求超时	MGS 调用业务系统服务超时。后端业务系统负载过高导致，需排查后端系统的运行情况。若超时设

01		置不合理，可以适当调整。注：默认超时时间为 3s。
4002	远程调用业务系统异常	MGS 调用业务系统服务出现异常。专有云用户可以在服务端日志 gateway-error.log 中查看详细信息。
4003	API 分组 HOST 异常	MGS 调用 HTTP 业务系统服务出现 UnknownHostException 异常。请检查 API 分组配置的域名是否存在。
5000	未知异常	其他严重错误。专有云用户可以在服务端日志 gateway-error.log 中查看详细信息。
7000	没有设置公钥	移动 APP 中无线保镖中无 appId 对应的密钥或者网关无法获取 appId 对应的签名密钥。
7001	验签的参数不够	网关服务端验证签名不通过。
7002	验签失败	网关服务端验证签名不通过。
7003	验签-时效性失败	API 请求入参 ts 时间戳超过系统设置的时间有效性。需要检查客户端时间是否为系统时间。
7007	验签-缺少 ts 参数	API 请求缺少验签 ts 参数。
7014	验签-缺少 sign 参数	API 请求缺少验签 sign 参数。一般情况下是客户端签名数据失败，导致缺失 sign 参数。请检查客户端无线保镖图片是否正确。
8002	跨域预检请求（CORS preflight）	跨域预检请求。

业务侧结果码

以下结果码可在业务系统服务器内部查看错误信息。

通过查看各个业务系统上的 ~/logs/mobileservice/monitor.log 日志可确定异常具体信息。

结果码	适用协议	描述	解释
6000	MPC、DUBBO	RPC-目标服务找不到	发布的服务（service）无法找到，服务器无法访问或者服务已迁移。
6001	MPC、DUBBO	RPC-目标方法找不到	发布的该 service 内的方法无法找到。
6002	MPC、DUBBO	RPC-参数数目不正确	传入的参数个数，与声明的参数个数不相等。
6003	MPC、DUBBO	RPC-目标方法不可访问	目标方法不能被调用。
6004	HTTP、MPC、DUBBO	PRC-JSON 解析异常	HTTP：将 RPC 参数转换为后端 HTTP 请求参数时发生异常。 MPC/DUBBO：将 RPC JSON 数据反序列化为业务参数对象时失败。
6005	MPC、DUBBO	PRC-调用目标方法时参数不合法	反射调用时，参数不合法。
6007	MPC、DUBBO	PRC-验证登录服务不可用	没有实现 SPI 包中验证登录接口或者验证登录接口配置出错。
6666	HTTP、MPC、DUBBO	RPC-业务抛出异常	HTTP：后端系统返回 HTTP status code 不等于200。 MPC/DUBBO：业务方抛出的异常。RPC 无法处理，统一为业务异常。

~~Android 客户端结果码~~

结果码	描述	提示文案
0	未知错误	未知错误，请稍后再试
1	客户端找不到通讯对象，没有设置 Transport	网络出错，请稍后再试
2	客户端没有网络，如用户关闭了网络或者禁止了应用的网络权限	网络无法连接
3	SSL相关错误，包括 SSL 握手错误，SSL 证书错误	客户端证书有误，请检查手机的时间设置是否准确。
4	客户端网络连接超时，TCP 建连超时，目前超时时间为 10s	网络欠佳
5	客户端网络速度过慢，数据读写超时，socketTimeout 的场景	网络欠佳
6	客户端请求服务端无响应，NoHttpResponseException	网络出错，请稍后再试
7	客户端网络 IO 错误，对应 IOException	网络出错，请稍后再试
8	客户端网络请求调度错误，执行线程中断异常	网络出错，请稍后再试
9	客户端处理错误，包括序列化错误、注解处理错误、线程执行错误	网络出错，请稍后再试
10	客户端数据反序列化错误，服务端数据格式有误	网络出错，请稍后再试
13	请求中断错误，例如线程中断时网络请求会被中断	网络出错，请稍后再试
15	客户端网络授权错误 ，HttpHostConnectException，Connection to xxx refused，无网络或者对应服务器拒绝连接	网络无法连接
16	DNS 解析错误	网络无法连接，请稍后再试
18	网络限流，客户端限流，当客户端请求流量超过阈值后会被限制网络请求	网络限流，请稍后再试
code >= 400 和 code < 500	HTTP 响应码为 4xx	网络无法连接
400 > code >= 100 和 500 < code < 600	HTTP 非成功的响应码	网络无法连接，请稍后再试

9.2 无线保镖结果码说明

本文所列的无线保镖错误码同时适用于 Android 及 iOS 操作系统。根据不同错误类型，下面将错误码分为以下三类：

- 一般错误码
- 静态数据加解密错误码
- 安全签名接口错误码

如果发生错误，Xcode 的 console 中会打印“SG ERROR: xxxx” 格式的错误码，具体含义如下所示。

一般错误码

错误码	含义

101	参数不正确，请检查输入的参数。
102	主插件初始化失败。
103	有依赖的插件没有引入。本错误码打印的同时会提示缺失的插件名，请按提示进行操作。
104	引入了插件，但加载失败。一般是因为 other linker flags 中没有添加 -all_load 或 -ObjC 所致，添加后可以解决。
105	引入了对应插件，但该插件的依赖插件没有引入。本错误码打印的同时会提示缺失的插件名，请按提示进行操作。
106	引入了对应插件，但该插件的依赖插件版本不符合要求。本错误码打印的同时会提示依赖的版本号，请按提示进行操作。
107	引入了对应插件，但该插件的版本不符合要求。
108	引入了对应插件，但该插件的依赖资源没有引入。
109	引入了对应插件，但该插件的依赖资源版本不符合要求。
121	图片文件有问题。一般是生成图片文件时的 bundle id 和当前应用的 bundle id 不一致。
122	没有找到图片文件，请确保图片文件在项目目录下。
123	图片文件格式有问题，请重新生成图片文件。
124	当前图片的版本太低。
125	init with authcode 初始化错误。
199	未知错误，请重试。
201	参数不正确，请检查输入的参数。
202	图片文件有问题。一般是生成图片文件时的 bundle id 和当前应用的 bundle id 不一致。
203	没有找到图片文件，请确保图片文件在项目目录下。
204	图片文件格式有问题，请重新生成图片文件。
205	图片文件的内容不正确，请重新生成图片文件。
206	参数中的 key 在图片文件中找不到，请确认图片文件中有这个 key。
207	输入的 key 非法。
208	内存不足，请重试。
209	不存在指定索引的 key。
212	请升级新版本图片，当前图片的版本太低。
299	未知错误，请重试。

静态数据加解密错误码

错误码	含义
301	参数不正确，请检查输入的参数。
302	图片文件有问题。一般是获取图片文件时的 apk 签名和当前程序的 apk 签名不一致。请使用当前程序的 apk 重新生成图片。
303	没有找到图片文件，请确保图片文件在 res\drawable 目录下。
304	图片文件格式有问题，请重新生成图片文件，，生成方法请参考 mPaaS 插件 > 生成无线保镖图片 。 一种常见场景就是二方和三方图片混用。二方和三方的图片不兼容，需要各自生成。
305	图片文件内的内容不正确，请重新生成图片文件。

306	参数中的 key 在图片文件中找不到，请确认图片文件中有这个 key。
307	输入的 key 非法。
308	内存不足，请重试。
309	不存在指定索引的 key。
310	待解密数据不是可解密数据。
311	待解密数据与密钥不匹配。
312	当前图片的版本太低，请升级新版本的图片，生成方法请参考 mPaaS 插件 > 生成无线保镖图片 。
399	未知错误，请重试。
401	参数不正确，请检查输入的参数。
402	内存不足，请重试。
403	获取系统属性失败，请确认是否有软件拦截，获取系统参数。
404	获取图片文件的密钥失败，请确认图片文件的格式和内容是否正确。
405	获取动态加密密钥失败，请重试。
406	待解密数据格式不符合解密要求。
407	待解密数据不符合解密要求，请确认该数据是本设备上保镖动态加密产生。
499	未知错误，请重试。
501	参数不正确，请检查输入的参数。
502	内存不足，请重试。
503	获取系统属性失败，请确认是否有软件拦截，获取系统参数。
504	获取图片文件的密钥失败，请确认图片文件的格式和内容是否正确。
505	获取动态加密密钥失败，请重试。
506	待解密数据不是可解密数据。
507	待解密数据与密钥不匹配，请重试。
508	传入 key 对应的 value 不存在。
599	未知错误，请重试。

安全签名接口错误码

错误码	含义
601	参数不正确，请检查输入的参数。
602	内存不足，请重试。
606	使用带 seedkey 的 top 签名时，没有找到 seedkey 对应的 seedsecret。
607	yw_1222.jpg 图片文件有问题。一般情况是生成图片时的 bundle id 和应用的 bundle id 不匹配。
608	没有找到 yw_1222.jpg 图片文件，请确保图片文件在项目目录下。若工程中已存在此图片，请确认工程 meta.config 文件中 base64Code 字段不为空。若为空，请参考 无线保镖 重新手动生成 yw_1222.jpg 图片。
609	yw_1222.jpg 图片文件格式有问题，请重新生成图片文件，生成方法请参考 mPaaS 插件 > 生成无线保镖图片 。 一种常见场景就是二方和三方图片混用。二方和三方的图片不兼容，需要各自生成 。

610	yw_1222.jpg 图片文件内的内容不正确，请重新生成图片文件。
611	参数中的 key 在图片文件中找不到，请确认图片文件中有这个 key。
615	当前图片的版本太低，请升级新版本的图片，生成方法请参考 mPaaS 插件 > 生成无线保镖图片 。
699	未知错误，请重试。

9.3 网关日志说明

9.3.1 网关服务端日志

本文包含对各类服务端日志的说明。注意，只有专有云用户才有权限查看服务端日志。

API 摘要日志

日志路径：~/logs/gateway/gateway-page-digest.log

- 日志打印时间
- 请求地址
- 响应
- 结果（Y/N）
- 耗时：单位 ms
- operationType
- 系统名
- appId
- workspaceId
- 结果码
- 客户端 productId
- 客户端 productVersion
- 渠道
- 用户 ID
- 设备 ID
- UUID
- 客户端 trackId
- 客户端 IP
- 网络协议：HTTP 或 HTTP2
- 数据协议：JSON 或 PB
- 请求数据大小：字节
- 响应数据大小：字节
- 压测标

- TraceId：请求的唯一标识，可以将所有的摘要日志、详细日志或者异常日志串起来
- cpt 标
- 客户端系统类型
- 后端系统耗时
- clientIp 类型：4 或 6
- RPC 协议版本：1.0 或 2.0

格式：

时间 - (请求地址,响应,结果（Y/N）,耗时,operationType,系统名,appId,workspaceId,结果码,客户端productId,客户端productVersion,渠道,用户ID,设备ID,UUID,客户端trackId,客户端IP,网络协议,数据协议,请求数据大小,响应数据大小,是否压测,TraceId,是否组件API,客户端系统类型,后端系统耗时,IP协议版本,RPC协议版本)

样例：

2020-06-03 14:14:08,001 - (/mgw.htm,response,Y,61ms,alipay.mcdp.space.initSpaceInfo,-,84EFA9A281942,default,1000,-,-,-,-,Wz4Zak5peDgDAGRNW5rFFGhT,Wz4Zak5peDgDAGRNW5rFFGhTN9uqCLa,Wz4Zak5peDgDAGRNW5rFFGhTN9uqCLa,223.104.210.136,HTTP,JSON,2,2406,F,0a1d7667159116484794082982065 8,T,ANDROID,61,4,2.0)

API 详细日志

日志路径：~/logs/gateway/gateway-page-detail.log

详细日志分为以下类别：

- 请求日志： [request]
- 响应日志： [response]

请求日志

- 日志打印时间
- 客户端 IP
- TraceId
- 日志级别
- 日志类型：request
- operationType
- appId
- workspaceId
- requestData
- sessionId
- did：设备 ID
- contentType

- mmtp：T 或者 F，是否使用 MMTP 协议
- async：是否异步调用，T 或者 F

响应日志

- 日志打印时间
- 客户端 IP
- TraceId
- 日志级别
- 日志类型：response
- operationType
- appId
- workspaceId
- responseData
- resultStatus：结果码
- contentType
- sessionId
- did：设备 ID
- mmtp：T 或者 F，是否使用 MMTP 协议
- async：是否异步调用，T 或者 F

样例：

```
2017-12-21 15:37:10,208 [100.97.90.113][79c731d5151384183020882931425 8] 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 [][79c731d5151384183020882931425 8] INFO - [response]operationType=com.alipay.gateway.test,appId=2A9ADA1045,workspaceId=antcloud,responseData=***,resultStatus=1000,contentType=JSON,sessionId=-,did=WjtkmWe1uHsDADl7BEleyK2L,mmtp=F,async=T
```

API 统计日志

日志路径：~/logs/gateway/gateway-page-stat-s.log

- 日志打印时间
- operationType
- appId
- workspaceId
- 结果：Y/N
- 结果码

- 压测标识
- 请求总量
- 请求总耗时：ms

格式：

时间 - operationType,appId,workspaceId,结果(Y/N),结果码,压测标识（T/F）,请求总量,请求总耗时（ms）

样例：

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

网关线程统计日志

日志路径：~/logs/gateway/gateway-threadpool.log

- 日志打印时间
- 线程名
- 活动线程数
- 当前线程池的线程数
- 创建过的最大线程数量
- 核心线程数
- 最大线程数
- 任务队列容量
- 剩余队列容量

格式：

时间 [线程名
,ActiveCount,PoolSize,LargestPoolSize,CorePoolSize,MaximumPoolSize,QueueSize,QueueRemainingCapacity]

样例：

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

网关配置日志

日志路径：~/logs/gateway/gateway-config.log

此日志记录网关配置变更的相关通知。

网关默认日志

日志路径：~/logs/gateway/gateway-default.log

网关默认日志，未指定特定日志的埋点都会打到此日志。

网关错误日志

日志路径：~/logs/gateway/gateway-error.log

此日志记录错误和异常堆栈。

9.3.2 网关 SPI 日志

此部分日志说明，只针对集成了 mpaasgw-spi-mpc 或 mpaasgw-spi-dubbo 的业务系统。

API 摘要日志

日志路径：~/logs/mobileservice/page-digest.log

- 日志打印时间
- operationType
- 客户端 productId
- 客户端 productVersion
- 耗时：单位 ms
- 结果（Y/N）
- 结果码
- uniqueId

格式：

时间 - (operationType,productId,productVersion,耗时,结果（Y/N）,结果码,uniqueId)

样例：

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

SPI 启动日志

日志路径：~/logs/mobileservice/boot.log

启动日志记录业务系统 mobileservice 的注册和启动情况，分为如下六个阶段：

- Start-To-Register-Service：开始解析 API 服务接口
- Start-To-Analyze-Method：开始解析 API 服务接口中的方法
- Analyze-Method-Parameter：解析方法参数
- Method-Info：方法信息
- Registered-OperationType：完成 单个 API operationType 注册
- Register-Service-Success：完成此接口中所有的 API operationType 注册

此日志可以帮助排查 operationType 是否注册成功。

样例：

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

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

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

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

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

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

API 监控日志

日志路径：~/logs/mobileservice/monitor.log

记录 API 请求的详细日志，包括 API 请求的 debug 日志以及出错情况下的异常堆栈。

SPI 默认日志

日志路径：~/logs/mobileservice/common-default.log

SPI 默认日志，未指定特定日志的埋点都会打到此日志。

SPI 错误日志

日志路径：~/logs/mobileservice/common-error.log

记录错误和异常堆栈。

9.4 业务接口定义规范

考虑到手机端开发环境的限制（尤其是 iOS 系统），以及保持接口定义的简单，服务端在定义移动服务接口时，不能使用 Java 语法的全集。接口定义规范涉及三类定义：

- 内部支持数据类规范 ：已支持的 Java 原生类和包装类。
- 用户接口类规范 ：用户定义的 interface，包含 API 调用的 method 声明。
- 用户定义实体类规范 ：用户定义的实体 class（包含 field 声明），接口类中 method 参数或返回值、其它用户定义实体类将会引用到。

内部支持数据类规范

不支持的数据类型

- 容器类型不能多层嵌套。
- List 或 Map 必须有泛型信息。
- List 或 Map 的泛型信息不能是 array 类型
- 不支持单字节（字节数据 byte [] 是支持的。）
- 不支持对象数组，请用 list 代替。
- 属性名不能是 data 和 description，会与 iOS 的属性冲突。
- Map 类型的 key 必须是 String 类型。
- 类型不能是抽象类。
- 类型不能是接口类。

错误的写法：

```
public class Req {
private Map<String,List<Person>> map; //容器类型不能多层嵌套。
private List<Map<Person>> list; //容器类型不能多层嵌套。
private List list1; //List 或 Map 必须有泛型信息。
private Map map1; //List 或 Map 必须有泛型信息。
private List<Person[]> listArray; //List 或 Map 的泛型信息不能是 Array 类型。
private byte b; //不能为单字节
private Person[] personArray; //不支持对象数组，请用 List 代替
private String description; //属性名不能为 description
}
```

支持的数据类型

```
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，但：必须使用类型参数；不能使用其具体子类 以下简称 List
java.util.Map，但：必须使用类型参数；不能使用其具体子类；key 类型必须是 String 以下简称 Map
Enum
byte[]
```

正确的写法：

```
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;
```

用户接口类规范

method 的参数

不可以引用：

- 枚举类型
- 除上文提到的 Map、List、Set 之外的泛型
- 抽象类
- 接口类
- 原生类型的数组

可以引用：

- 具体的实体类，要求引用类型与实际的对象类型保持一致；不可使用父类引用类型指向子类对象。
- 内部支持数据类, 但数组、Map、List、Set这些集合类型不可以嵌套。

如下是错误示例：

```
Map<String,String[]>
Map<String,List<Person>>(Person为一个具体的实体类)
List<Map<String,Persion>>
List<Persion[]>
```

method 的返回值

不可以引用：

- 枚举类型
- 除上文提到的 Map、List、Set 之外的泛型
- 抽象类
- 接口类
- 原生类型的数组

可以引用：

- 具体的数据类，要求引用类型与实际的对象类型保持一致；不可使用父类引用类型指向子类对象 (比如，不能用 Object 引用指向其它对象)

注意：如果父类为具体类，生成工具将检查不出此类错误。

- 内部支持数据类见文章开头定义。数组、Map、List、Set 集合类型不可以嵌套，参见上文相关示例。

method 的定义

- 使用 @OperationType 注解，未加此注解的方法将被生成工具忽略。
- method 不可 overloading。

代码生成工具限制

- 允许接口类定义的继承关系，但会合并层次关系。
- 允许但忽略接口类中定义变量。
- 允许但忽略接口类中的方法声明抛出异常。
- 一个源文件中只能包含一个接口类的定义，不能包含其它类（内部类、匿名类等）的定义。
- 接口类定义本身和其引用到的类型必须为内部支持数据类或可以从源码获得定义。

用户定义实体类规范

field 定义

不可以引用：

- 枚举类型
- 除上文提到的 Map、List、Set 之外的泛型
- 抽象类
- 接口类
- 原生类型的数组

可以引用：

- 具体的实体类，要求引用类型与实际的对象类型保持一致；不可使用父类引用类型指向子类对象。
- 内部支持数据类。数组、Map、List、Set 集合类型不可以嵌套，参见上文相关示例。
- 修饰符包括 transient 的属性将会被忽略。

final static int 定义的常量（其它不符合该要求定义的常量或静态变量将被忽略）。

建议：不推荐 is 开头的成员变量定义。

类的定义

- 可以继承自其它实体类。
- 忽略其方法声明，生成工具将会自动根据实体类字段生成 setter/getter 方法。

代码生成工具限制

- 属性的声明必须一行一个。
- 允许但忽略用户定义实体类实现的接口。
- 一个源文件中只能包含一个用户定义实体类的定义，不能包含其它类（内部类、匿名类等）的定义。
- 接口类定义本身和其引用到的类型必须为内部支持数据类或可以从源码获得定义。

9.5 密钥生成方法

根据您的业务需求，查看生成密钥的方法。密钥包括 RSA 密钥、ECC 密钥、国密密钥。

前置条件

您已通过 OpenSSL 官网 下载并安装 OpenSSL 工具（1.1.1 或以上版本）。

生成 RSA 密钥

1. 打开 OpenSSL 工具，使用以下命令行生成 RSA 私钥。您可以选择生成 1024 或 2048 位的私钥。

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

2. 根据 RSA 私钥生成 RSA 公钥。

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

生成 ECC 密钥

1. 打开 OpenSSL 工具，使用以下命令行生成 ECC 的密钥对。您必须选择 secp256k1 椭圆曲线算法。

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

2. 根据 secp256k1-key.pem 密钥对生成 ECC 公钥。

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

生成国密密钥

1. 打开 OpenSSL 工具，使用以下命令行生成国密 SM2 私钥 sm2-key.pem。

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

2. 根据 sm2-key.pem 密钥对生成国密 SM2 公钥 sm2pubkey.pem。

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

9.6 网关签名机制

为保证客户端请求不被篡改和伪造，RPC 请求有签名机制，RPC 模块会自动实现加签功能。

基本的加签、验签过程如下：

1. 将 requestBody 中的内容转换为字符串。
2. 使用无线保镖安全模块，通过保存在加密图片（即无线保镖图片）中的加密密钥，对转化的字符串进行加签。
3. 将加密后的签名放在请求中发给网关。
4. 网关使用相同方式签名，校验两个签名是否相等。

蚂蚁集团
ANT GROUP