蚂蚁科技

移动调度
使用指南

文档版本：20250731

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

未经蚂蚁集团事先书面许可，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及其他蚂蚁集团相关的商标均为蚂蚁集团所有。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。蚂蚁集团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，并在蚂蚁集团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集团授权渠道下载、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。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.产品简介 05
2.接入客户端 06
2.1. 接入 Android 06
2.2. 接入 iOS 10
3.使用控制台 15
3.1. 管理域名分组 15
3.2. 管理域名策略 15
3.3. 管理用户组 17
3.4. 管理系统配置 17
4.常见问题 18

1.产品简介

移动调度中心（Mobile Dispatch Center，MDC）是基于客户端用户业务属性进行用户分组的精细化移动调度服务产品，能快速有效帮助客户达成业务分组灰度测试、A/B 测试、故障调优等目标。

产品优势

移动调度中心的产品优势如下：

- **用户级精准调度**

  使用基于业务属性的用户分组实现用户粒度精准调度，支持客户端唯一标识 DeviceID/UserID 或自定义用户标识识别进行调度。

- **多场景业务适配**

  调度服务可用于用户线路网络故障切换，产品运营开展 A/B 测试、灰度测试以及系统容灾备份。

- **服务高可用**

  服务高可用 99.95%，服务端秒级配置完成，终端拉取秒级配置生效。

- **配置兼容**

  域名调度支持 CNAME 配置和 IPv6/IPv4 平滑切换，且兼容常规调度服务。

应用场景

移动调度中心的应用场景如下：

- **域名防劫持**

  Local DNS 域名劫持的情况已不罕见，使用移动调度服务可以规避这类安全风险，提高移动终端的接入安全性。

- **快速解析 DNS**

  通过提供域名解析服务，客户端会在多个时机进行域名解析请求，在访问资源时，域名已经解析完毕。

- **容灾调度**

  目前移动互联网接入链路复杂多样，亟需实现对移动终端网络接入的精准化和策略化控制。在部分机房出错，业务服务出现中断的情况下，可以根据优先级和权重来实现主备或双活等多种容灾模式，以快速将客户端调度到服务正常的机房。

- **智能化就近择优接入**

  在移动互联网时代，更快更优的接入能带来更优质的用户体验。移动调度中心可以根据用户的运营商信息帮助企业实现客户端择优地接入到更匹配的网络。

产品功能

移动调度中心的功能如下：

- **细粒度域名策略调度**

  通过配置域名、区域、运营商、用户组等域名策略调度条件，按照策略将用户组中的用户调度到指定的接入点。通过配置优先级和权重，完成移动调度中心自身流量的容灾调度。

- **高效管理多域名组**

  客户端可通过域名组向移动调度中心发起请求，一次性获取该域名组全量域名的解析结果，极大地提升请求效率。

- **便捷管理多用户组**

  通过配置多用户组，帮助实现域名的用户粒度调度。

- **灵活管理系统配置开关**

  系统配置开关用于管控组件域名是否使用移动调度中心获取解析结果，默认开启，开启后组件域名直接使用移动调度中心获取，无需 Local DNS 解析，高效防劫持。

2.接入客户端

2.1. 接入 Android

移动调度中心在 10.1.68 及以上版本基线中提供支持。移动调度中心支持 **原生 AAR 接入** 和 **组件化（Portal&Bundle）接入** 两种接入方式。

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。
- 若采用组件化（Portal&Bundle）方式接入，需先完成 组件化接入流程。

添加移动调度 SDK 依赖

接入 mPaaS 支持以下两种接入方式，您可以根据实际情况进行选择。

- 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 **移动调度** 组件。
- 组件化（Portal&Bundle）方式

  在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **移动调度** 组件。更多信息，请参考 接入流程。

使用移动调度 SDK

添加 AndroidManifest 配置

使用 SDK 需要在 AndroidManifest 中添加移动调度中心的 dg 和 host 信息。

```
<!--配置移动调度中心分组名称-->
<meta-data
android:name="amdc.dg"
android:value="${MDC_DG}" />
<!--host配置为业务的移动调度中心服务域名-->
<meta-data
android:name="amdc.host"
android:value="${IP}" />
<!--可选，用于第一次本地dns缓存被污染情况下的备选IP-->
<meta-data android:name="amdc.backup"
android:value="${IP}" />
```

调用移动调度中心接口

MPMDC 类提供了所有移动调度中心相关 API，调用移动调度中心接口的操作步骤如下：

1. mPaaS 框架初始化完成后，在 `Application.onCreate` 中调用 `init` 方法初始化 DMC。

   ```
   void init(Context context);
   ```

   示例如下：

```
@Override
public void onCreate() {
    super.onCreate();

    MP.init(this,
        MPInitParam.obtain().setCallback(new MPInitParam.MPCallback() {
            @Override
            public void onInit() {
                MPMDC.init(App.this);
            }
        })
    );
}
```

2. 获取 IP 信息。

```
MPHttpIp getIpsByHost(String host); // 获取对应host的IP信息
List<MPHttpIp> getAll(); // 获取所有移动调度中心的IP配置信息
class MPHttpIp {
    public MPHttpIpEntry[] ipEntries; // IP列表
    public String host; // host
    public String ip; //本地IP
}
class MPHttpIpEntry {
    public static final int IP_TYPE_V4 = 4;
    public static final int IP_TYPE_V6 = 6;
    public String ip; // IP
    public int port; // 端口
    public int ipType; // IP类型，v4/v6
}
```

3. 设置 UID。移动调度中心内部会根据 UID 来区分数据，提供用户灰度功能。

```
MPLogger.setUserId(xxxx);
```

RPC 开启 MDC 功能

RPC 内部会使用移动调度中心相关配置，并提供 API。业务方基于 RPC 使用 RPC 提供的接口。业务方可以使用 mPaaS 的开关配置动态控制移动调度中心和 IPv6 的开关。

- 开启移动调度中心开关，RPC 使用移动调度中心配置；关闭移动调度中心开关，RPC 不使用移动调度中心配置。开关默认开启。

  ```
  MPRpc.openMDC(isOpen);
  ```

- 关闭 IPv6 开关，RPC 会过滤掉移动调度中心 IPv6 的 IP，只使用移动调度中心 IPv4 的 IP；开启 IPv6 开关，则 IPv6 和 IPv4 IP 均使用。开关默认关闭。

  ```
  MPRpc.openIPV6(isOpen);
  ```

单独接入 MDC 示例

通过 IP 完成 Socket 建连

```
MPHttpIp httpIp = MPHttpIp.getIpsByHost(String host);
MPHttpIpEntry[] ipEntries = httpdnsIP.ipEntries;
String ips = new String[ipEntries.length];
for (int i = 0; i < httpdnsIPEntries.length; i++) {
 ipArr[i] = httpdnsIPEntries[i].ip;
}
//使用socket建连，
//selectAddressIndex 可以根据请求失败情况自行设置index
Socket newSocket = new Socket(Proxy.NO_PROXY);
newSocket.connect(new InetSocketAddress(ips[selectAddressIndex], port));

//通过ip建连后，如果访问https，还需进行ssl建连
//ssl建连,可以参考下方ssl建连参考代码，
//httpUrlConnection和okhttp同样需要设置sslSocketFactory来解决https问题
//获取socketFactory同样可以参考下面代码，均为标准模版代码，可以结合自己项目的情况

//使用httpUrlConnection
URL url = new URL("http://ips[selectAddressIndex]/index.jsp");
//如果是https，需要设置
HttpsURLConnection.setDefaultSSLSocketFactory(sslSocketFactory);
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.connect();

//使用okhttp
Request request = new Request.Builder()
                .get()
                .url("http://ips[selectAddressIndex]/index.jsp")
                .build();
OkHttpClient client = new OkHttpClient.Builder()
         //如果是https，需要设置
            .sslSocketFactory(sslSocketFactory).build();
Call call = client.newCall(request);
```

通过 SSL 建连

```
//获取socketFactory
 SSLSocketFactory sslSocketFactory = createZSSLContext().getSocketFactory();

 //ssl握手
 if (sslSocketFactory != null) {
        SSLSocket sslSocket = (SSLSocket) sslSocketFactory.createSocket(socket, host, p
ort, true);
        sslSocket.setUseClientMode(true);
        sslSocket.startHandshake();
        Log.d(LOGTAG, "tls socket ssl handshake suc");
        sslSocket.setSoTimeout(0);
        return sslSocket;
 }

 public static final SSLContext createZSSLContext() throws NoSuchAlgorithmException, Ke
yManagementException {
        final SSLContext sslContext = SSLContext.getInstance("TLS");
        final X509KeyManager x509KeyManagers[] = {createDefaultKeyManager()};
```

```
        final X509TrustManager x509TrustManagers[] = {new
X509TrustManagerWrapper(createDefaultTrustManager())};
        sslContext.init(x509KeyManagers, x509TrustManagers, new SecureRandom());
        return sslContext;
    }

    public static final X509KeyManager createDefaultKeyManager() throws
KeyManagementException {
        try {
            String defaultAlgorithm = KeyManagerFactory.getDefaultAlgorithm();
            KeyManagerFactory kmf = KeyManagerFactory.getInstance(defaultAlgorithm);
            kmf.init(null, null);
            return findX509KeyManager(kmf.getKeyManagers());
        } catch (NoSuchAlgorithmException e) {
            throw new KeyManagementException(e);
        } catch (UnrecoverableKeyException e) {
            throw new KeyManagementException(e);
        } catch (KeyStoreException e) {
            throw new KeyManagementException(e);
        }
    }

    private static final X509KeyManager findX509KeyManager(KeyManager[] kms) throws Key
ManagementException {
        for (KeyManager km : kms) {
            if (km instanceof X509KeyManager) {
                return (X509KeyManager)km;
            }
        }
        throw new KeyManagementException("Failed to find an X509KeyManager in "+
Arrays.toString(kms));
    }


    public static final X509TrustManager createDefaultTrustManager() throws
KeyManagementException {
        try {
            String defaultAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
            TrustManagerFactory tmf =
TrustManagerFactory.getInstance(defaultAlgorithm);
            tmf.init((KeyStore)null);
            return findX509TrustManager(tmf.getTrustManagers());
        } catch (NoSuchAlgorithmException e) {
            throw new KeyManagementException(e);
        } catch (KeyStoreException e) {
            throw new KeyManagementException(e);
        }
    }

    private static final X509TrustManager findX509TrustManager(TrustManager tms[]) thro
ws KeyManagementException {
        for (TrustManager tm : tms) {
            if (tm instanceof X509TrustManager) {
                return (X509TrustManager) tm;
            }
```

```
            }
        }
        throw new KeyManagementException("Failed to find an X509TrustManager in
"+Arrays.toString(tms));
    }

    private static final class X509TrustManagerWrapper implements X509TrustManager {

        private final X509TrustManager x509TrustManager;

        public X509TrustManagerWrapper(X509TrustManager x509TrustManager) {
            this.x509TrustManager = x509TrustManager;
        }

        @Override
        public final void checkClientTrusted(X509Certificate[] chain, String authType)
throws CertificateException {
            x509TrustManager.checkClientTrusted(chain, authType);
        }

        @Override
        public final void checkServerTrusted(X509Certificate[] chain, String authType)
throws CertificateException {
            int size = 0;
            if (chain != null) {
                size = chain.length;
            }
            try {
                x509TrustManager.checkServerTrusted(chain, authType);
            } catch (Throwable e) {
                if (e instanceof CertificateException) {
                    throw (CertificateException) e;
                } else {
                    throw new CertificateException(e);
                }
            }
        }

        @Override
        public final X509Certificate[] getAcceptedIssuers() {
            X509Certificate[] acceptedIssuers = x509TrustManager.getAcceptedIssuers();
            return acceptedIssuers;
        }

    }
```

2.2. 接入 iOS

移动调度中心在 10.1.68 及以上版本基线中提供支持。本文介绍移动调度中心接入 iOS 的方法。

接入 SDK

根据您采用的接入方式，请选择相应的接入方法。

接入方法

接入移动开发平台 mPaaS 的方式主要有以下 3 种：

- 无工程，从头开始创建一个全新的当前已有工程，所有 SDK 直接链接到此工程中：基于已有工程且使用 mPaaS 插件接入。
- 当前已有工程，所有 SDK 直接链接到此工程中采用 CocoaPods 管理 SDK：基于已有工程且使用 mPaaS 插件接入。
- 当前已有工程，采用 CocoaPods 管理 SDK：基于已有工程且使用 CocoaPods 接入。

基于 mPaaS 框架接入

mPaaS iOS 框架是源自支付宝客户端的开发框架。该框架直接接管应用的生命周期，负责整个应用 启动托管、应用生命周期管理。同时基于 Framework 的设计思想，将业务隔离成相对独立的模块，着力于追求模块之间的高内聚、低耦合。参考 mPaaS 框架介绍 获取更多信息。

您可以快速搭建一个基于 mPaaS 框架的全新应用，具体步骤参考 基于 mPaaS 框架接入。

基于已有工程且使用 mPaaS 插件接入

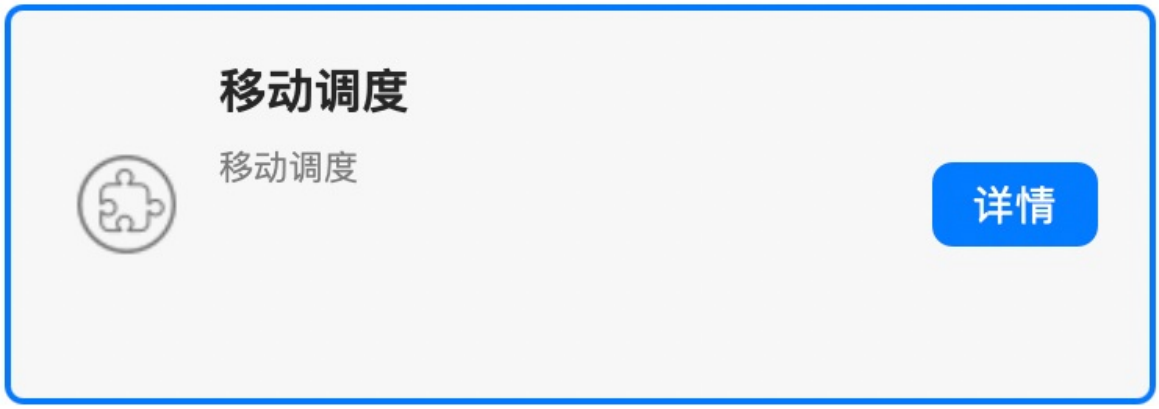
若当前已有工程为一个单工程，所有 SDK 直接链接到此工程中，推荐您使用 mPaaS 插件接入，具体步骤参考 基于已有工程且使用 mPaaS 插件接入。

基于已有工程且使用 CocoaPods 接入

若当前已有工程通过 Cocoapods 来管理 SDK 的依赖，推荐您使用 Cocoapods 接入，具体步骤参考 基于已有工程且使用 CocoaPods 接入。

组件接入

- 使用 mPaaSPlugin 接入 mPaaS。
  i. 点击 Xcode 菜单项 **Editor** > **mPaaS** > **编辑工程**，打开编辑工程页面。
  ii. 选择 **移动调度**，保存后点击 **开始编辑**，即可在模块列表里添加移动调度模块。



- 使用 Cocopods 接入 mPaaS。
  i. 在 Podfile 文件中添加 `mPaaS_pod 'mPaaS_MDC'` 。
  ii. 在终端（terminal）执行 `pod install` 即可完成接入。

使用 SDK

参数配置

重写 `MPNetworkInterface` 的 `category` 配置。

- Port 开关：是否使用移动调度中心下发的 Port，默认关闭。
- 备用 IP：移动调度中心请求地址的备用 IP，默认 nil。

- 域名分组：控制台配置的域名分组，默认值为 nil。
- 域名列表：控制台配置的域名列表，默认值为 nil。

```
// 移动调度中心 Port开关
- (BOOL)useMdcPort {
    return NO;
}

// 备用IP，默认值为nil
- (NSString *)amdcBackupIp
{
    return @"192.168.1.1";
}

// 控制台配置的域名分组。默认值为nil
- (NSString *)amdcDomainGroup
{
    return @"test";
}

// 控制台配置的域名列表，默认值为nil
- (NSArray *)amdcDomainList
{
    return @[@"www.test.com"];
}
```

接口文档

初始化

初始化移动调度中心，并刷新数据。

- 初始化操作只应执行一次，若在控制台修改了域名 IP 信息，或者在客户端修改了配置，请调用刷新数据的方法 `initServiceWithDelegate` 。
- 若未初始化移动调度中心，则其他接口无法使用。

```
/// 初始化
+ (void)initServiceWithDelegate:(id<MPDNSNetworkDelegate>)delegate;
```

`MPDNSNetworkDelegate` 回调

```
// code为RPC响应码，1000表示成功
- (void)didDNSNetworkInfoRefresh:(int)code;
```

刷新数据

刷新域名IP信息

```
/// 刷新数据
+ (void)refresh;
```

获取全部 IP 信息

获取移动调度中心返回的全部域名 IP 信息。

```
/// 获取全部IP信息
+ (NSArray<MPDNSNetworkInfo *> *)getNetworkInfoList;
```

获取单个域名的 IP 信息

返回单个域名对应的 IP 信息。

```
/// 获取单个域名的IP信息
/// @param host 域名
+ (MPDNSNetworkInfo *)getNetworkInfoByHost:(NSString *)host;
```

返回结果说明

`MPDNSNetworkInfo`

```
@interface MPDNSNetworkInfo : NSObject

/// 域名
@property (nonatomic, copy) NSString *host;
/// cname
@property (nonatomic, copy) NSString *cname;
/// IP信息列表
@property (nonatomic, copy) NSArray<MPDNSNetworkIP *> *ipList;

@end
```

`MPDNSNetworkIP`

```
@interface MPDNSNetworkIP : NSObject

/// IP地址
@property (nonatomic, copy) NSString *ip;
/// IP端口
@property (nonatomic, assign) NSInteger port;
/// IP类型(4:IPv4 6:IPv6)
@property (nonatomic, assign) NSInteger ipType;

@end
```

`MPMDCNetworkInfo`

```
// IP地址
@property (nonatomic, copy) NSString *ip;
// IP端口
@property (nonatomic, copy) NSNumber *port;
// 移动调度中心时间戳
@property (nonatomic, copy) NSString *timestamp;
```

代码示例

```
#import <MPMDC/MPMDC.h>

// 初始化移动调度中心，只会执行一次
[MPDNSNetworkService initServiceWithDelegate:nil];

// 刷新移动调度中心，若用户在控制台修改IP或者在客户端修改配置，需手动执行刷新操作
[MPDNSNetworkService refresh];
```

RPC接入

RPC 接入移动调度中心需要配置开关，通过重写 `DTRpcInterface` 的 `category` 来配置

- 移动调度中心开关：是否启用移动调度中心，开启该配置后仍然需要手动调用移动调度中心初始化方法。默认关闭。
- IPv6 开关：是否使用移动调度中心返回的 IPv6 地址发起请求，开启该配置后，优先使用 IPv6 地址发起请求。默认关闭。
- Port 开关：是否使用移动调度中心下发的 Port。

```
// 移动调度中心开关
- (BOOL)isHTTPDNS {
    return YES;
}

// IPv6 开关
- (BOOL)ipv6Enabled {
    return NO;
}

// 移动调度中心 Port 开关
- (BOOL)useMdcPort {
    return NO;
}
```

> **说明**
>
> - RPC 内部不会自动初始化移动调度中心，需要用户手动调用移动调度中心的初始化方法才可以正常使用。
> - RPC 单次请求内部不会重试，需要用户重新请求才会切换到下一个 IP。

3.使用控制台

移动调度中心控制台提供管理域名分组、管理域名策略、管理用户组和管理系统配置的功能。客户端发送请求时通过域名组进行解析，最大限度地节省流量。通过配置域名调度策略，将域名策略用户组中的用户调度到指定的接入点。

3.1. 管理域名分组

域名组指具有相同属性的域名的集合，客户端可通过域名组向移动调度中心发起请求，一次性获取该域名组全量域名的解析结果，极大地提升请求效率。本文介绍域名分组的新增、修改、删除操作步骤。

添加域名分组

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤添加域名分组。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名分组** 标签页点击 **创建分组** 按钮，进入 **创建分组** 页签。
3. 在 **创建分组** 页签，输入分组名称和分组描述。
4. 点击 **提交**，即可生成域名分组。

修改域名分组信息

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤修改域名分组信息。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名分组** 标签页下，点击域名分组操作列的 **编辑**，进入 **编辑域名分组** 页面。
3. 点击右上角的 **编辑**，可选择修改域名组的名称、描述和状态。
4. 点击 **保存** 完成域名组信息修改。

删除域名分组

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤删除域名分组。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名分组** 标签页下，点击域名分组操作列的 **删除**，二次确认后删除域名组。

添加域名

进入**移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤在域名分组中添加域名。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名分组** 标签页下，点击域名分组操作列的 **编辑**，进入 **编辑域名分组** 页面。
3. 点击 **对应域名** 后的 **添加**，打开 **添加域名** 页签。
4. 点击 **+ 添加域名**，输入域名。可添加多个域名。
5. 单击 **提交**，域名组添加域名成功。

删除域名

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤在域名分组中添加域名。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名分组** 标签页下，点击域名分组操作列的 **编辑**，进入 **编辑域名分组** 页面。
3. 点击域名对应操作列的 **删除**，二次确认后删除域名。

3.2. 管理域名策略

本文介绍域名调度策略的新增、修改、删除的操作步骤。

添加域名调度策略

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤添加域名调度策略。

新增域名

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名策略** 标签页点击 **创建域名** 按钮，进入 **创建域名** 页签。
3. 在 **创建域名** 页签，输入域名和描述。
4. 点击 **提交**，即可完成添加域名。

配置域名调度策略

1. 在 **域名策略** 标签页下，点击域名操作列的 **编辑**，进入 **编辑域名策略** 页面。
2. 单击 **域名调度策略** 后的 **添加**，打开 **添加域名调度策略** 页签。
3. 您可根据业务需求配置各字段。其中，优先级和权重用于移动调度中心完成自身流量的容灾调度，移动调度中心的容灾策略分为主备和多活，具体配置如下：

   主备： 同一域名、区域、运营商、用户组条件下，主机优先级为 0，备机优先级为 1，权重均为 100。

   多活： 同一域名、区域、运营商、用户组条件下，优先级均为 0，权重按比例分配，多中心的权重和为 100。

   优先级数字越低在返回的 IP 列表里越靠前，权重数字越大返回的概率越大，总体上会按照优先级 > 权重的过滤顺序来返回 IP 列表。

   例如：在 **记录值** 中输入 `1.1.1.X:1`，即代表该用户组的用户将被分配到 `1.1.1.X:1`。

   > **重要**
   > - 同一域名、区域、运营商、用户组条件下，不允许同时添加 IP 和 CNAME。
   > - 同一域名、区域、运营商、用户组条件下，只允许同时添加一条 CNAME 数据。
   > - 若打开扩展能力开关，开发者可以给相应的 IP 或 CNAME 记录添加自定义扩展参数（形式为 K、V 键值对），扩展参数随记录一同下发到客户端。
   > - MDC 内置了部分扩展能力通过 CNAME 记录添加 BACKUP_DOMAIN 标签，取值为 T；通过 MDC 可支持主备域名（CNAME 为备用域名）自动切换。

4. 类似地，您可为同一域名配置不同域名调度策略。完成必要字段填写后，单击 **提交** 即可保存域名调度策略配置。完成提交后，您可在域名调度策略列表中查看新增的策略，其中 **记录类型** 支持筛选。

域名调度策略各维度配置说明

- 优先级数字越低最终在返回的 IP 列表里越靠前。
- 两个优先级一样的 IP 会根据权重配置加权随机返回，数字越大代表权重越大，返回的概率更高；随机指的是加权随机，权重越高返回的概率高，但不代表始终会返回高权重。
- 优先级不一致，权重不论怎么配置，都是优先级高的在前。
- 总体上会按照 **用户白名单筛选** > **地域筛选** > **运营商筛选** > **优先级排序** > **加权随机选一个** 这样的过滤顺序返回 IP 列表。

修改域名调度策略

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤修改域名策略信息。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **域名策略** 标签页下，点击域名操作列的 **编辑**，进入 **编辑域名策略** 页面。

3. 点击右上角的 **编辑**，可选择修改域名策略的域名、描述和状态。
4. 点击 **保存** 完成域名策略信息修改。

删除域名调度策略

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤删除域名策略。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 点击域名操作列的 **删除**，二次确认后删除域名策略。

3.3. 管理用户组

本文介绍添加、配置和删除用户组的操作步骤。

添加用户组

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤添加用户组。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **用户组** 标签页点击 **创建用户组** 按钮，进入 **创建用户组** 页签。
3. 在 **创建用户组** 页签，输入用户组名和用户组描述。
4. 点击 **提交**，即可生成用户组。

添加用户 ID

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤添加用户 ID。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **用户组** 标签页下，点击用户组名操作列的 **编辑**，进入 **编辑用户组** 页面。
3. 在 **用户 ID 列表** 模块下，您可选择从文件添加或手动添加用户 ID。
4. 按照提示操作后，点击 **提交** 完成添加用户 ID。

修改用户组信息

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤修改用户组信息。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **用户组** 标签页下，点击用户组名操作列的 **编辑**，进入 **编辑用户组** 页面。
3. 点击右上角的 **编辑**，可选择修改用户组名和用户组描述。
4. 点击 **保存** 完成用户组信息修改。

删除用户组

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤删除用户组。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 点击用户组名操作列的 **删除**，二次确认后删除用户组。

3.4. 管理系统配置

系统配置用于管控组件域名是否使用移动调度中心获取解析结果，您可通过开关配置开启。

进入 **移动开发平台 mPaaS** 控制台，选择目标应用，完成以下步骤管理系统配置。

1. 在左侧导航栏点击 **后台服务** > **移动调度**，进入 **移动调度** 页面。
2. 在 **系统配置** 标签页，MGS 移动网关开关默认开启，您可单击右侧按钮关闭。

4.常见问题

移动调度中心高可用

Q：移动调度中心实现高可用的策略是什么？

A：移动调度中心实现高可用的策略如下：

- 移动调度中心已预先配置好自身域名和 IP，这样 App 可以拉取到 MDGC 自身的域名调度配置。
- 移动调度中心已预先配置好域名和 IP 到公网 DNS，主要用于 App 首次安装启动后通过公网域名拉取自身 IP。
- 移动调度中心 SDK 内置有移动调度中心域名的 IP 硬编码，防止 App 首次安装后运营商的 DNS 不可用。

Q：移动调度中心域名获取IP的策略有哪些？

A：移动调度中心域名获取自身域名 IP 的策略如下：

- 从移动调度中心后台配置中获取，移动调度中心会下发移动调度中心 IP list。
- 若移动调度中心 IP list 中没有 IP，则需进行 httpdns 解析。
- 若 httpdns 解析失败，则由客户端进行 local DNS 解析获取 IP。

IP 列表拉取

Q：客户端调用服务端接口时，拉取域名对应的 IP 列表的时机有哪些？

A：可以在以下情形下拉取域名对应 IP 列表：

- 客户端程序首次启动时拉取 IP 列表。
- 前台网络切换时拉取 IP 列表。
- IP TTL 超时的情况下拉取 IP 列表。

MGS RPC 链路

Q：对应的 MGS RPC 链路是否必须为 H2？

A：不是的。在使用移动调度中心 SDK 将移动网关接入移动调度中心时，不需要使用 H2 链路。

IPv6 降级

Q：IPv6 是如何降级的？

A：移动调度中心服务端配置域名对应的 IPv4 和 IPv6 地址，客户端发起请求后，将全量提供给客户端进行业务选择。当 MDS 上最终开启客户端 H2 加 IPv6 开关时，如果 IPv6 不可用，客户端会有自动降级处理，连续 4 次 RPC 失败则自动降级。IPv6 失败降级后，3 小时内不恢复，这时即使开关可以获取到 IPv6 为 `true` 的开关，也不会通过 IPv6 建连来规避这部分用户使用 IPv6 的异常问题。

接入常见问题

Q：项目中使用自建应用的配置文件，却没有配置移动调度中心域名的分组信息，为什么调用移动调度中心配置接口时能获取到相关 IP 信息？

A：移动调度中心返回的信息包括内置应用和自建应用两部分，内置应用中会提前创建域名 IP （不建议执行此操作），自建应用中用户可以配置自己的域名 IP 信息。客户端使用自建应用的配置信息，即使没有在自建应用中配置，也会默认下发内置应用中的域名IP信息。

Q: 必须要使用内置应用的配置文件吗？

A: 内置应用用来配置移动调度中心以及 MGS 等 mPaaS 组件域名信息，移动调度中心每次处理域名请求都会默认下发内置应用中配置的域名信息。移动调度中心自身的域名必须配置在内置应用内，MGS 可以不配置，不配置则使用 Local DNS 解析。

Q：同一个域名可以添加多个 IPv4、IPv6 地址吗？

A：可以。

Q：添加多个 IP 后，服务端一次返回多个 IP 还是多个 IP 轮询返回？如果返回多个 IP 到客户端，那么客户端有轮询能力吗？

A：服务端会一次返回全部 IP 信息，客户端会轮询IP信息，如果轮询信息IP失败，会使用 LocalDNS 解析 IP。

客户端问题

Q：IPv4/IPv6 访问逻辑是什么样的？

A：若 RPC 的 IPv6 开关开启并且当前网络环境支持 IPv6，则优先使用 IPv6 地址发起请求。否则优先使用 IPv4 地址。

Q：对移动调度中心服务访问失败的处理机制是什么？

A：若访问移动调度中心服务失败，表示无法拉取到 IP 列表，则发起 RPC 请求时仍需使用域名。

Q：用当前 App 缓存中的 IP 访问失败后的详细处理机制是什么？

A：使用 IP 发起 RPC 请求失败后，到下一次发起请求时，会切换到下一个 IP，如果发现 IP 列表已遍历完，则使用域名请求。

Q：是否存在决定客户端何时获取 IP 列表的 SDK？

A：RPC中提供了可以开启/关闭移动调度中心的开关，开关默认开启。如果关闭移动调度中心，则使用域名发起请求。

TTL+TTD 逻辑判断

Q：TTL+TTD 的逻辑判断流程是什么样的？

A：判断逻辑如下：

1. 客户端获取到域名对应的 IP 信息后，判断 TTL 是否超时，若超时执行步骤 2，否则执行步骤5。
2. TTL 超时，保留 IP 信息并异步请求新的 IP 信息，然后执行步骤 3。
3. 判断 TTD 是否超时，若超时执行步骤 4，否则执行步骤 5。
4. TTD 超时，弃用 IP 信息，返回空，流程结束。
5. 返回 IP 信息列表，流程结束。

> **重要**
>
> - TTL 超时，一定会发起请求更新 IP 信息，跟 TTD 是否超时无关。
> - TTL 超时，TTD 未必超时；TTD 超时，TTL 必然超时。
> - TTD 超时，返回空。如用户下次发起请求时，新的 IP 信息还未返回，仍然会返回空。