蚂蚁科技

接入 iOS
使用指南

文档版本：20230727

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.接入方式介绍

本文介绍了针对不同的开发进度和场景，建议使用的接入方式。

三种接入方式

根据 iOS 开发工程的进展和使用场景，接入移动开发平台 mPaaS 的方式主要有以下三种：

- 当前无工程，从头开始创建一个全新的工程：基于 mPaaS 框架接入。
- 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。
- 目前已有工程，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 框架创建一个应用。

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。

在 iOS，mPaaS 采用的是 Objective-C 开发语言。如果您的工程采用的是 Swift 开发语言，可以通过桥接的方式引入 mPaaS 的 Objective-C 代码。

> **说明**
>
> 如果有更多接入相关问题，欢迎搜索群号 41708565 加入钉钉群进行咨询交流。该钉钉群已添加 mPaaS 公有云答疑小助手，能够快速回答常见接入问题。更多关于使用公有云答疑小助手的信息，请参见公有云答疑小助手。

2.在控制台创建应用

要使用 mPaaS，您首先需要在 mPaaS 控制台创建应用并下载配置文件。

前置条件

您需要确保拥有开发者账号。账号注册的更多信息，请参见 阿里云账号注册流程。

创建 mPaaS 应用

1. 登录 mPaaS 控制台。

   > **说明**
   >
   > 如果您在其他平台（如蚂蚁科技开放平台）使用 mPaaS，请登录对应平台的 mPaaS 控制台。

2. 单击 **创建应用** 按钮。
3. 完善应用信息。
   i. 输入应用名称。示例：mPaaS Demo。
   ii. 单击 `+` 上传应用图标。您可以忽略此步骤，此时应用将使用默认图标。
4. 单击 **创建** 按钮，完成应用创建。您可以看到应用列表，列表里包含刚才创建的应用。

下载配置文件

1. 在应用列表页，单击应用名称（如上一步中创建的应用 mPaaS Demo），您将看到以下页面：
   - 左上方展示应用名称，您可以在此切换应用。
   - 页面左侧展示 mPaaS 提供的组件服务列表。
2. 单击 iOS 代码配置，进入 **配置应用** 页面。
3. 在配置应用页面，单击 **下载配置文件** 按钮，进入 **代码配置** 页。
4. 在代码配置页，输入 **Bundle ID**，单击 **下载配置** 按钮，下载 `.config` 格式的应用配置文件到本地，为后续开发做准备。

   文件内容为 `JSON` 格式，示例如下：

```
{
 "appId":"ONEX8319839121823",
 "appKey":"ONEX8319839121823_IOS",
 "base64Code":"/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAMCAgMCAgMDAwMEAwMEBQgFBQQEBQoHBwYIDAoMDA
sKCwsNDhIQDQ4RDgsLEBYQERMUFRUVDA8XGBYUGBIUFRT/2wBDAQMEBAUEBQkFBQkUDQsNFBQUFBQUFBQUFBQ
UFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBT/wAARCAADAAMDASIAAhEBAxEB/8QAHwAAAQUBAQEB
AQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII
0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhI
WGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb
3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSEx
BhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZ
GVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2N
na4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxFobXF3FfU78U5nKhRBay4TiPPuh2kFAwABAQAAAAQAAB4AAAD
AAAAADAAAAIcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAABUAAAABAAAAAAAAAAAAAAAIAAAAAAAAAFvUTmJ3AAAAAAAA
AAAAAAAKcUJ9ig78iKP1rG1ocLPmgO/Rk8SIdQLM81Pl439j7fGla3uzZwNWXK/ExCB3qqBmfDMqBKks0ZyAtn91c
mdjbngBAwF4dWZ3AwECBgEAEQEBCAFxdHNqb3bWQSxDMewbnV9aVcN3qtd5KtBXv5coWb2f91l9bhimpuDibB7Geh
bF/pNB/JlfH8kyyoP1sUTDB2qAOuXBez479sBz+S+5vSbzsH7QvEju8CsHs6RJMjcwGe4pYkDC5zsNtqyYZWj4fXw
7QukgspHRg95zpzEVc3rPckT0FfZSqXfH+DufN5TzMn0q1tLojI70A6C1AwK0XHG+AsqubKcb/0W1k76Cxk2n/SKx
ceTMgulD3YZ1FAOv2fI8dm9IMz/s0WYijpn0XI+51P8AAMKgGAYAAAAA",
 "bundleId":"YourBundleID",
 "rootPath":"mpaas/ios/ONEX8319839121823-default",
 "workspaceId":"default",
 "syncport":"443",
 "syncserver":"msync.mpaas.cn-hangzhou.aliyuncs.com",
 "pushPort":"443",
 "pushGW":"push.mpaas.cn-hangzhou.aliyuncs.com",
 "rpcGW":"https://mgw.mpaas.cn-hangzhou.aliyuncs.com/mgw.htm",
 "logGW":"https://mdap.mpaas.cn-hangzhou.aliyuncs.com",
 "mpaasapi":"https://mpaasapi.mpaas.cn-hangzhou.aliyuncs.com/mgw.htm",
 "mrtcserver":"wss://mrtc.mpaas.cn-hangzhou.aliyuncs.com/ws",
 "mdc":"https://mdc.mpaas.cn-hangzhou.aliyuncs.com",
 "mpaasConfigVersion":"V_1.0",
 "mpaasConfigEnv":"ONEX_CLOUD",
 "mpaasConfigPluginExpired":"",
 "mpaasConfigLicense":"O4jxU1QWsWOlJYkHqAO39uvfJZCfPN4DAM8gsU/4qdxJ3KmTp0J6HebLJeSOphFr0h
5y54J7O0GAAYm2KNGFUxgaehvuR2ai8XRUuPKse0cHSRBBi3e8qQcSEHBREa3UZguFyKYe95iWqFPw3e5Rc1hEtwA
ZX4BwNbrWjEjkaI0ZwxEq+OyeLrSWJFrYi6DCLFzsLeF/l9uOHRAeba84/ruSNdg5X//qnso1J1/SXhOxWxd/uYbF
yU78k7YTeBL8wUW0Pat67QkBG0dL9+lW/050PPCDyfNIbk1JeQ8EhpwA0GStbW78LYJ21YWb4L0zYGIKvfAd5gG3Y
WDE7YxtzQ==="
}
```

3.基于 mPaaS 框架接入

本文将引导您从零开始创建一个基于 mPaaS 框架接入的全新工程。

阅读本文前，请确保已阅读 接入方式简介。

操作步骤

1. 安装开发者工具：mPaaS Xcode Extension 插件。
2. 在控制台创建应用。
3. 在 macOS 的 **应用程序** 中找到 mPaaS 独立的应用 **mPaaSPlugin**，单击直接运行。

4. 单击 **创建工程** 按钮。
5. 配置 mPaaS 新工程，输入 **项目名称** 并导入第二步中下载的 `.config` 配置文件，然后单击**下一步** 按钮。
6. 选择要创建应用的 **模版类型**，然后单击 **下一步** 按钮。
7. 选择工程使用的 **基线类型和版本**，然后单击 **下一步** 按钮。

   基线类型说明：

   - **标准基线**：公开的标准 mPaaS SDK 版本，可以在下拉列表中选择。
   - **定制基线**：目前只开放给专有云用户，在输入框中填入对应的基线 ID 后会自动进行校验。

   > **说明**
   >
   > - 校验失败会提示错误信息。
   > - 校验正确后即可使用。

8. 选择工程需要添加的模块，然后单击 **下一步** 按钮。

   > **说明**
   >
   > 您也可以先不添加模块，在创建工程之后，再参考各组件的接入文档，使用 **编辑工程** 功能添加所需的模块。

9. 选择项目保存的目录，单击 **创建** 按钮。
10. 开始创建工程。
11. 稍等片刻，创建成功之后会自动打开新建的 Xcode 工程。该工程包含与工程文件同级的 `MPaaS` 目录，与 mPaaS 相关的文件都在该目录下；更多信息，请参考 mPaaS 目录结构。
12. 直接运行此工程，进行测试。

后续步骤

您可以参考各组件的接入文档，接入并使用 mPaaS 组件。

相关链接

- mPaaS 框架介绍：了解 mPaaS iOS 框架。
- mPaaS Xcode Extension 插件：了解如何使用 mPaaS Xcode Extension 插件。

4.基于已有工程且使用 mPaaS 插件接入

假设已有基于原生 iOS 框架的工程，本文将引导您使用 mPaaS Xcode Extension 插件接入 mPaaS。

阅读本文前，请确保已阅读 接入方式简介。

操作步骤

1. 安装开发者工具：mPaaS Xcode Extension。
2. 在控制台创建应用。
3. 在 macOS 的 **应用程序** 中找到 mPaaS 独立的应用 **mPaaSPlugin**，单击直接运行。
4. 将工程文件拖拽进插件、或者单击插件中 **打开工程** 按钮后定位到工程文件即可。
5. 选择 **导入云端元数据** 菜单，单击 **选择配置文件** 按钮，选中第二步中下载的 `.config` 配置文件，单击**开始导入** 按钮，将对应的配置导入到当前工程中。

   > **说明**
   >
   > 选择配置文件之后，可以单击 **X** 按钮来关闭，只有单击 **开始导入** 之后配置才会生效。

6. 单击界面上的 **编辑模块** 菜单，工作区会展示未集成的提示页面。
7. 单击 **添加模块** 会弹出 **模块列表** 编辑页面。
   i. 进入页面会提示选择集成的 **基线类型和版本**，默认为最新基线，单击 **确认** 按钮。
   ii. 收起基线选择菜单后，下方会刷新出对应基线的模块列表，在列表中单击卡片来添加对应的模块。
   iii. 单击每个模块右侧的 **详情** 按钮可以查看该模块的具体信息，包括描述、Release Note、包含的 framework 文件详情等。
   iv. 选择好需要添加的模块后，单击右上角的 **保存** 按钮保存编辑结果，同时返回到主页面，工作区的模块列表中会展示编辑的结果。
8. 返回主页面后，单击 **开始编辑**，将选择的模块添加到当前工程中。

   > **说明**
   >
   > 只有单击 **开始编辑** 之后，结果才会生效。

9. 重新启动当前工程，您会看到工程中新增了与工程文件同级的 mPaaS 目录， mPaaS 相关的文件都在该目录下。更多信息，请参考 mPaaS目录结构。
10. 直接 build 工程，确保工程可以编译通过。至此您已成功完成 mPaaS 接入过程。

后续步骤

您可以参考各组件的接入文档，接入并使用 mPaaS 组件。

相关链接

- 接入方式简介
- mPaaS Xcode Extension 插件

- mPaaS 目录结构

5.基于已有工程且使用 CocoaPods 接入

本文介绍如何基于 CocoaPods 原生的插件扩展机制生成各项配置，进而快速接入 mPaaS。

前置条件

- 已安装 CocoaPods 1.0.0 及以上版本，并确保要接入的工程是 CocoaPods 工程。
- 已安装 Cocoapods-mPaaS 插件。如您尚未安装该插件，可使用以下命令进行安装。

```
sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPodsPlugin.sh)
```

- 已在控制台创建应用，并下载了 `.config` 配置文件。更多信息，参见 在控制台创建应用。

接入步骤

1. 将 `.config` 配置文件拷贝到工程的根目录下（与 `Podfile` 同级）。

   > **说明**
   >
   > 请确认下载的 `.config` 配置文件的文件名是以 `iOS` 结尾。如果是以 `ios` 结尾，则需要手动改为 `iOS`。

2. 在命令行执行 `pod mpaas init` 命令，自动处理 `Podfile` 文件，并添加 `plugin`、`source` 以及 `mPaaS_baseline` 配置。自动配置的代码如下：

```
plugin "cocoapods-mPaaS"
source 'https://gitee.com/mpaas/podspecs'
mPaaS_baseline 'x.x.x'
```

3. 配置 `Podfile` 文件。

   i. 指定 mPaaS 基线，修改 `mPaaS_baseline`。例如：`mPaaS_baseline '10.2.3'`，其中 `10.2.3` 为基线版本号，不同版本之间的区别，参见 发布说明。

   ii. 添加 mPaaS 组件依赖，使用 `mPaaS_pod`。例如：`mPaaS_pod "mPaaS_Nebula"`，其中 `mPaaS_Nebula` 为组件名称，更多组件名称可参考下方的组件列表。

组件配置	适用基线	说明
mPaaS_pod "mPaaS_LocalLog"	10.1.32+	本地日志
mPaaS_pod "mPaaS_Log"	10.1.32+	移动分析：行为日志、自动化日志、Crash 日志、性能日志分析。
mPaaS_pod "mPaaS_Diagnosis"	10.1.32+	诊断：客户端诊断分析。

mPaaS_pod "mPaaS_RPC"	10.1.32+	移动网关：提供下载、上传、RPC 调用等功能。
mPaaS_pod "mPaaS_Sync"	10.1.32+	移动同步：长连接服务。
mPaaS_pod "mPaaS_Push"	10.1.32+	消息推送
mPaaS_pod "mPaaS_Config"	10.1.32+	开关配置：根据 key 从服务端拉取对应的 value，可动态控制客户端逻辑。
mPaaS_pod "mPaaS_Upgrade"	10.1.32+	升级发布：提供便捷的主动检测升级的服务，可用于日常灰度发布、线上新版本更新提示。
mPaaS_pod "mPaaS_Share"	10.1.32+	分享：支持分享文本、图片到微博、钉钉、支付宝好友等知名渠道。
mPaaS_pod "mPaaS_Nebula"	10.1.32+	H5 容器与离线包：Nebula 容器，支持前端与 native 交互。
mPaaS_pod "mPaaS_UTDID"	10.1.32+	设备标识：简单快捷地获取设备 ID，以利于应用程序安全有效的找到特定设备。
mPaaS_pod "mPaaS_DataCenter"	10.1.32+	统一存储：提供安全、快速、可加密、支持多种数据类型的 KV 存储；数据库 DAO 支持等多种持久化方案。
mPaaS_pod "mPaaS_ScanCode"	10.1.32+	扫码：快速识别二维码、条形码。
mPaaS_pod "mPaaS_LBS"	10.1.32+	移动定位：移动客户端定位解决方案。
mPaaS_pod "mPaaS_CommonUI"	10.1.32+	通用 UI：通用 UI 组件库，提供各种 UI 组件。
mPaaS_pod "mPaaS_BadgeService"	10.1.32+	红点：客户端“红点”提醒组件，支持红点、数字、New 等提醒样式，自动管理树形结构的红点关系。
mPaaS_pod "mPaaS_AlipaySDK"	10.1.32+	支付宝快捷支付：支付宝快捷收银台。

mPaaS_pod "mPaaS_Multimedia"	10.1.32+	多媒体组件：多媒体组件，支持图片下载、上传、缓存等功能。
mPaaS_pod "mPaaS_MobileFramework"	10.1.32+	移动框架：客户端应用框架，子 app 管理，多 tab 类应用管理，第三方跳转管理，viewController 跳转，异常处理与上报。
mPaaS_pod "mPaaS_OpenSSL"	10.1.32+	OpenSSL
mPaaS_pod "mPaaS_TinyApp"	10.1.32+	小程序：小程序集成发布能力。
mPaaS_pod "MPBaseTest"	10.1.32+	基础测试：基础的测试模块。
mPaaS_pod "mPaaS_CDP"	10.1.32+	智能投放：智能配置各类营销广告和展示形式，动态投放到客户端。
mPaaS_pod "mPaaS_AliAccount"	10.1.60+	小程序账户通：小程序账户通发布。
mPaaS_pod "mPaaS_ARTVC"	10.1.68	音视频通话：音频、视频通话组件。支持双人、多人视频通话和在线会议。
mPaaS_pod "mPaaS_BlueShield"	10.2.3+	蓝盾加密组件：在 config 文件中添加 absBase64Code 参数，可自动生成蓝盾图片。

完整的 Podfile 示例如下：

Pods 〉 Podfile 〉 No Selection

```
# mPaaS Pods Begin
plugin "cocoapods-mPaaS", :guard_image_version => 6
source "https://gitee.com/mpaas/podspecs"
mPaaS_baseline '10.2.3' # 请将 x.x.x 替换成真实基线版本

mPaaS_version_code 12   # This line is maintained by MPaaS plugin automatically. Please don't modify.
# mPaaS Pods End

# Uncomment the next line to define a global platform for your project
platform :ios, "11.0"

target "MPRPCDemo_pod" do
  # Pods for MPRPCDemo_pod
  mPaaS_pod 'mPaaS_RPC'
  mPaaS_pod 'mPaaS_MDC'
  mPaaS_pod "mPaaS_CommonUI"
  mPaaS_pod "MPBaseTest"›
  mPaaS_pod "mPaaS_Push"
  mPaaS_pod "mPaaS_MobileFramework"
  mPaaS_pod "mPaaS_BlueShield"
end

```

4. 执行 `pod mpaas update x.x.x` ，其中 `x.x.x` 为配置的基线号，例如 `10.2.3` 。
5. 执行 `pod install` 即可完成接入。您还可以追加 `--verbose` 查看详细日志。

> **说明**
>
> 如果在执行 `pod install` 时提示不能找到从 GitHub 官网引入的库，请在 podfile 的顶部指定 GitHub 官方 Source 的源地址： `https://github.com/CocoaPods/Specs.git` 。

6. 如果您在接入后遇到了三方库冲突，可将引起冲突的三方库移除。具体操作，请参见 iOS 冲突处理。

升级指南

当 mPaaS 有新版本发布时，您可选择升级组件，或整体升级基线（即 SDK 版本）。

升级组件

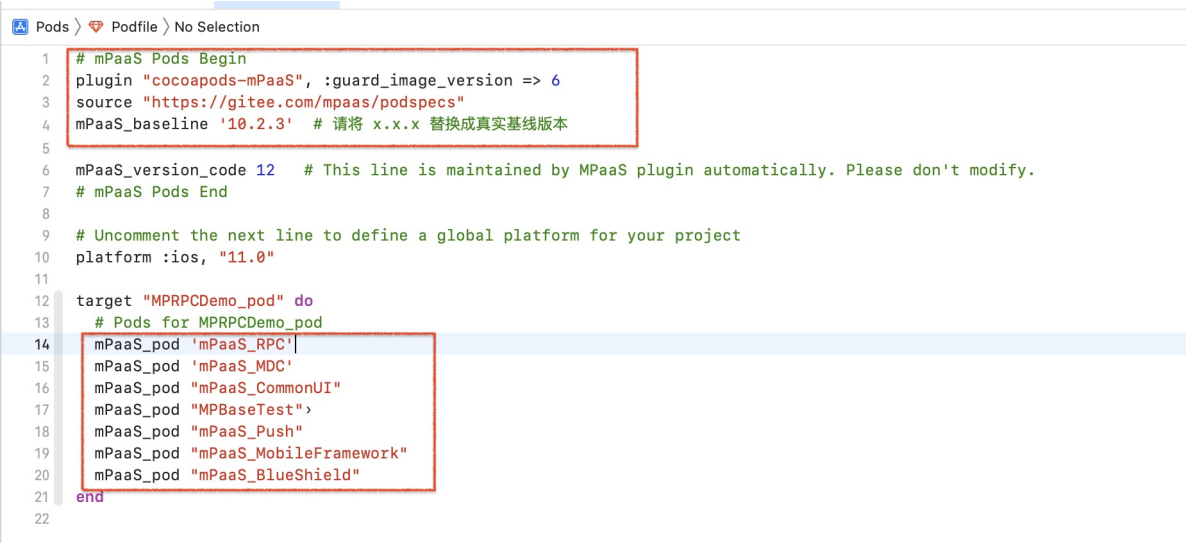
1. 在命令行执行 `pod mpaas update x.x.x` ，其中 `x.x.x` 为当前使用的基线版本号，例如 `10.2.3` 。

```
[➜  ~ pod mpaas update 10.2.3
1. update 10.2.3 baseline file ...
updateTime : 2023-06-09 10:32:15 +0800
The baseline has no change ...
The current verison is updated to 10.2.3.24
   update 10.2.3 baseline file Done

2. update mPaaS repo ...
Updating spec repo `gitee-mpaas-podspecs`
  $ /opt/local/bin/git -C
  /Users/yanjinquan/.cocoapods/repos/gitee-mpaas-podspecs fetch origin
  --progress
  remote: Enumerating objects: 29, done.
  remote: Counting objects: 100% (29/29), done.
  remote: Compressing objects: 100% (21/21), done.
  remote: Total 22 (delta 12), reused 0 (delta 0), pack-reused 0
  From https://gitee.com/mpaas/podspecs
     eed5568c..94fb3f1f  master     -> origin/master
  $ /opt/local/bin/git -C
  /Users/yanjinquan/.cocoapods/repos/gitee-mpaas-podspecs rev-parse --abbrev-ref
  HEAD
  master
  $ /opt/local/bin/git -C
  /Users/yanjinquan/.cocoapods/repos/gitee-mpaas-podspecs reset --hard
  origin/master
  HEAD is now at 94fb3f1f PodSpec Update at 2023-06-19 12:19:25
   update mPaaS repo Done
```

2. 执行 `pod install` 即可完成该基线下对应的组件的升级。

升级基线

1. 在 `podfile` 中，修改 `mPaaS_baseline` 对应的基线号（支持标准或者定制基线），例如从 `10.1.68` 修改为 `10.2.3` ，即可完成整体基线的升级。



2. 执行 `pod install` 即可完成基线升级。

mPaaS iOS podspec 地址切换

背景

mPaaS 原有使用的 podspec 保存仓库 `code.aliyun.com` 已停止服务（于 2023 年 06 月 01 日停止更新，于 2023 年 06 月 30 日停止服务）。

继续使用原有 repo 影响如下：

1. 使用 mPaaS pod plugin 进行 SDK 更新时，无法拉取到各基线的最新版本；
2. 2023.06.30 之后，使用 mPaaS pod plugin 无法拉取到任何基线版本。

当前 mPaaS 已在 `gitee.com` 上支持了全部 mPaaS 版本。

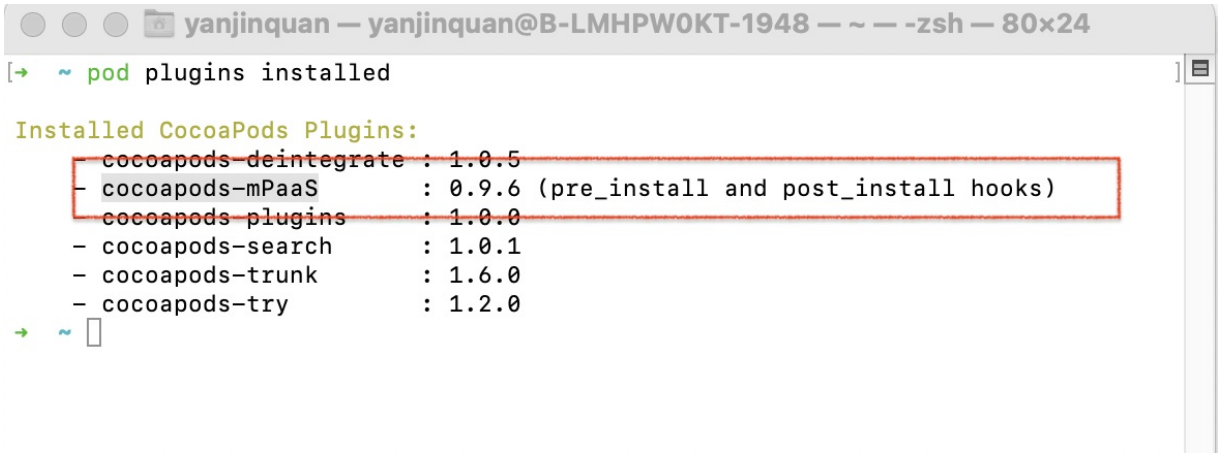
解决方案

升级 mPaaS pod plugin

执行下列命令更新到最新 mPaaS pod plugin。

```
sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPodsPlugin.sh)
```

执行完成后在终端中执行 `pod plugins installed` 命令，查看 cocoapods-mPaaS 的版本，显示为 `0.9.6` 或以上即为升级成功。



修改 podfile 中 source 配置

将 podfile 中原有的 `source "https://code.aliyun.com/mpaas-public/podspecs.git"` 替换为 `source "https://gitee.com/mpaas/podspecs"` 。

API 变更

本次修改只涉及插件的变更，暂无插件命令使用的变化。

测试验证

执行完成上述升级与修改配置操作后，可继续执行 mPaaS pod plugin 相关拉取命令测试是否可以拉取到最新基线版本以及 SDK。

参数列表

您可以通过配置参数，改变插件的一些默认行为。

使用方法：

在 podfile 中的 `plugin "cocoapods-mPaaS"` 后方添加参数。示例如下：

参数	功能	适用版本
`:guard_image_version => 6`	生成 V6 保镖图片	≥ V0.9.6
`:guard_image_version => 5`	生成 V5 保镖图片	≥ V0.9.6

`:only_frameworks => true`	在某些场景（如独立 framework 工程）中，不需要自动添加 mPaaS 模板目录文件。	≥ V0.9.5.0.0.2
`:check_repo => false`	在某些场景（如使用内网代理）下，不自动检查添加默认 repo。	≥ V0.9.5.0.0.2

> **说明**
>
> 10.2.3 基线版本无需设置 `:guard_image_version`，默认生成 V6 图片。

命令列表

安装了 cocoapods-mPaaS 插件后，您可使用命令行工具辅助开发。

命令	功能
`pod mpaas init`	在 `Podfile` 中添加 `plugin`、`source` 和 `mPaaS_baseline`。
`pod mpaas update <VERSION>`	更新基线，参数 `<VERSION>` 为具体的基线号，例如 10.2.3，同时也更新下 podspec 仓库。
`pod mpaas update --all`	在正式版插件中，该命令会升级插件，重新运行安装脚本。在 beta 版插件中，该命令除了能够实现正式版中的功能外，还会更新本地基线。
`pod mpaas info`	显示完整的基线和对应的组件信息。
`pod mpaas info <NAME> <VERSION>`（其中 `<VERSION>` 为可选）	筛选某个模块名的信息。
`pod mpaas info --only-mPaaS`	显示部分缺省的基线信息，方便一键粘贴到 Podfile 中。
`pod mpaas open`	直接从命令行打开 `.xcworkspace` 文件。
`pod mpaas version`	显示当前工程所用的完整基线。

`pod mpaas version --plugin`	显示当前 Cocoapods-mPaaS 插件的版本号。

6.进阶指南

6.1. mPaaS 目录结构

基于 mPaaS iOS 框架或者系统 iOS 框架的工程导入云端数据之后，会在工程目录下添加目录结构。

> **说明**
>
> 在 10.1.32 及以上的版本中，MPaaS > Targets > mPaaSDemo 内部的目录只会保留 `APMobileFramework` 和 `mPaas` 。如果是从低版本升级而来，其中其它组件的目录和 category 不会再生成。

目录结构如下：

```
└── MPaaS
    ├── mpaas_sdk.config
    ├── Targets
    |    └── mPaaSDemo（工程 Target 名称）
    |        ├── mPaaSDemo-mPaaS-Headers.h
    |        ├── mPaaSDemo-Prefix.pch
    |        ├── APMobileFramework
    |        ├── mPaas
    |        ├── meta.config
    |        └── yw_1222.jpg
    ├── Resources
    └── Frameworks
```

其中：

- `mpaas_sdk.config` ：当前工程添加的模块信息，包括版本、添加时间、资源文件等，由 mPaaS 插件自动维护，不得手动修改。
- `mPaaSDemo-mPaaS-Headers.h` ：当前工程依赖的 mPaaS 模块的头文件，由 mPaaS 插件自动维护，不得手动修改。
- `mPaaSDemo-Prefix.pch` ：当前工程 pch 文件的引用，会自动将 `mPaaSdemo-mPaaS-Headers.h` 加入 mPaaS 模块的头文件。
- `APMobileFramework` ：mPaaS 框架的生命周期管理的 category 文件。
- `mPaas` ：MPaaSInterface 的 category 文件。
- `meta.config` ：从 mPaaS 控制台下载的云端元数据。
- `yw_1222.jpg` ：通过元数据中的 base64code 字段生成的无线保镖验签图片，在移动网关验签时使用。如不需要移动网关功能，可删除此图片。
- `Resources & Frameworks` ：mPaaS 模块的资源文件和二进制文件目录，是当前工程所有 Target 使用的 mPaaS 模块的并集，由 mPaaS 插件自动维护，不得手动修改。

> **说明**
>
> 因为所有的 Target 共用 `Resources & Frameworks`，所以不同的 Target 不可以同时使用相同模块的不同版本，不得修改这两个目录；根据每个 Target 选择模块的不同，实际添加到各自的 `Build Phase` 里的 framework 也不相同。

6.2. mPaaS 框架介绍

mPaaS iOS 框架源自支付宝客户端的开发框架，基于 Framework 的设计思想，将业务隔离成相对独立的模块，并着力追求模块与模块之间高内聚、低耦合。

mPaaS iOS 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、处理与分发 UIApplication 的代理事件、统一管理各业务模块（微应用和服务）等。

本文将对 mPaaS iOS 框架进行详细的介绍。

启动托管

通过程序 main 函数的替换，直接接管应用的生命周期，整个启动的过程如下：

```
main -> DFClientDelegate -> 打开 Launcher 应用
```

应用生命周期管理

mPaaS 框架接入之后，完全替代了 AppDelegate 的角色，整个应用的生命周期由框架进行管理，但是用户依然可以实现应用生命周期各个阶段对应的代理方法，UIApplicationDelegate 中的所有代理方法，框架都提供了等价的接入方式，只需要在 Category 中覆盖对应的方法即可。

框架提供的生命周期方法声明如下，具体内容可以查看 `DTFrameworkInterface.h` 文件。

```
/**
 *  框架有一些自己的初始化逻辑在didFinishLaunching里需要实现，但会在执行之前回调该方法。
 */
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDic
tionary *)launchOptions;

/**
 *  框架回调该方法，让接入应用可以接管自己的didFinishLaunching逻辑。
 *  并且当返回DTFrameworkCallbackResultReturnYES或DTFrameworkCallbackResultReturnNO时，直接给
系统返回，不再执行接下来的逻辑。
 *  这个方法在框架启动BootLoader前回调，应用可以通过返回DTFrameworkCallbackResultReturnYES或DTFra
meworkCallbackResultReturnNO让框架提前退出，不运行默认的BootLoader。
 *  使用框架内部的默认实现即可，通常不需要覆盖。
 *
 *  @return 是继续让框架执行，还是直接给系统返回YES或NO
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application handleDidFinishLaunch
ingWithOptions:(NSDictionary *)launchOptions;

/**
 *  框架有一些自己的初始化逻辑在didFinishLaunching里需要实现，但会在所有逻辑完成后回调该方法。
 */
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDict
ionary *)launchOptions;
```

```
ionary *)launchOptions;

/**
 *  框架会率先回调该方法，让接入应用可以预先处理通知消息。
 *  当返回DTFrameworkCallbackResultContinue时，框架会把通知消息通过UIApplicationDidReceiveRemot
eNotification广播给全局监听者。并调用completionHandler(UIBackgroundFetchResultNoData)。
 *  当返回DTFrameworkCallbackResultReturn时，表示接入应用已经完全处理完通知消息，框架中止执行之后的
逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application didReceiveRemoteNotif
ication:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult r
esult))completionHandler;

/**
 *  框架会率先回调该方法，让接入应用可以预先处理通知消息。
 *  当返回DTFrameworkCallbackResultContinue时，框架会把通知消息通过UIApplicationDidReceiveLocal
Notification广播给全局监听者。
 *  当返回DTFrameworkCallbackResultReturn时，表示接入应用已经完全处理完通知消息，框架中止执行之后的
逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application didReceiveLocalNotifi
cation:(UILocalNotification *)notification;

/**
 *  框架会率先回调该方法，让接入应用可以预先处理通知消息。
 *  当返回DTFrameworkCallbackResultContinue时，框架会把通知消息通过UIApplicationDidReceiveLocal
Notification广播给全局监听者。并调用completionHandler()。
 *  当返回DTFrameworkCallbackResultReturn时，表示接入应用已经完全处理完通知消息，框架中止执行之后的
逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application handleActionWithIdent
ifier:(NSString *)identifier forLocalNotification:(UILocalNotification *)notification compl
etionHandler:(void (^)())completionHandler;

/**
 *  框架会率先回调该方法，让接入应用可以拿到deviceToken。
 *  当返回DTFrameworkCallbackResultContinue时，框架会把deviceToken通过UIApplicationDidRegister
ForRemoteNotifications广播给全局监听者。
 *  当返回DTFrameworkCallbackResultReturn时，表示接入应用已经完全处理完，框架中止执行之后的逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application didRegisterForRemoteN
otificationsWithDeviceToken:(NSData *)deviceToken;

/**
 *  当取deviceToken失败时，框架率先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，目前无其它逻辑。
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application didFailToRegisterForR
emoteNotificationsWithError:(NSError *)error;

/**
 *  框架会先给分享组件（如果有，并且shouldAutoactivateShareKit返回YES）通知，如果分享组件处理不了，
再回调该方法，由接入应用处理openURL。
 *  当返回DTFrameworkCallbackResultReturnYES或DTFrameworkCallbackResultReturnNO时，框架直接给
```

```
系统返回，不再执行接下来的逻辑。
 *  当返回DTFrameworkCallbackResultContinue时，继续由框架处理URL，并分发给SchemeHandler等类来处理
。
 *
 *  这个方法相比系统方法，多了一个newURL参数，允许应用在处理后，返回一个不同的url。如果函数整体返回DTF
rameworkCallbackResultContinue，并且给newURL赋值，框架会使用新的URL来做后续处理。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application openURL:(NSURL *)url
newURL:(NSURL **)newURL sourceApplication:(NSString *)sourceApplication annotation:(id)anno
tation;

/**
 *  框架率先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，目前无其它逻辑。
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)applicationWillResignActive:(UIApplication *)application;

/**
 *  框架率先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，目前无其它逻辑。
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)applicationDidEnterBackground:(UIApplication *)application;

/**
 *  框架率先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，目前无其它逻辑。
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)applicationWillEnterForeground:(UIApplication *)application;

/**
 *  框架先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，给分享组件事件（如果有，并且shouldAut
oactivateShareKit返回YES）。并且当整个应用没被加载时，调用BootLoader
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)applicationDidBecomeActive:(UIApplication *)application;

/**
 *  框架率先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，目前无其它逻辑。
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)applicationWillTerminate:(UIApplication *)application;

/**
 *  框架率先回调该方法。
 *  当返回DTFrameworkCallbackResultContinue时，框架继续执行，目前无其它逻辑。
 *  当返回DTFrameworkCallbackResultReturn时，框架中止之后的逻辑，目前无其它逻辑。
 */
- (DTFrameworkCallbackResult)applicationDidReceiveMemoryWarning:(UIApplication *)applicatio
n;
```

```
/**
 *  框架率先回调该方法，接入应用可以先行处理Watch的消息。
 *  当返回DTFrameworkCallbackResultContinue时，框架会把Watch消息通过UIApplicationWatchKitExtensionRequestNotifications广播给全局监听者。
 *  当返回DTFrameworkCallbackResultReturn时，表示接入应用已经完全处理完通知消息，框架中止执行之后的逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application handleWatchKitExtensionRequest:(NSDictionary *)userInfo reply:(void(^)(NSDictionary *replyInfo))reply;

/**
 *  框架率先回调该方法，接入应用可以先行处理消息。
 *  当返回DTFrameworkCallbackResultContinue时，框架会把消息通过UIApplicationUserActivityNotifications广播给全局监听者，并最后给系统返回NO。
 *  当返回DTFrameworkCallbackResultReturnYES或DTFrameworkCallbackResultReturnNO时，框架直接给系统返回，不再执行接下来的逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application continueUserActivity:(NSUserActivity *)userActivity restorationHandler:(void(^)(NSArray *restorableObjects))restorationHandler;

/**
 *  框架率先回调该方法，接入应用可以先行处理3D Touch快捷入口的消息。
 *  当返回DTFrameworkCallbackResultContinue时，框架会处理shortcutItem带过来的URL，并最后调用completionHandler()返回是否已经处理。
 *  当返回DTFrameworkCallbackResultReturn时，框架直接给系统返回，不再执行接下来的逻辑。
 */
- (DTFrameworkCallbackResult)application:(UIApplication *)application performActionForShortcutItem:(UIApplicationShortcutItem *)shortcutItem completionHandler:(void (^)(BOOL))completionHandler;

/**
 *  Background Fetch 机制回调
 *  必须在30s内回调completionHandler，否则进程将被terminate
 *  若要启用此机制，需要先配置Background Modes的fetch选项。其次在didFinishLaunching中调用下面的方法。更多信息参考文档。
 *  [application setMinimumBackgroundFetchInterval:UIApplicationBackgroundFetchIntervalMinimum];
 *  默认实现为空，需要接入方自己处理。
 */
- (void)application:(UIApplication *)application performFetchWithCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler;
```

应用模块划分

mPaaS 框架内定义了微应用和服务的概念来进行模块间的划分。其中，以是否有 UI 界面作为标准，Framework 将不同的模块划分为 微应用 和 服务，通过 框架上下文 进行微应用与服务的生命周期管理。

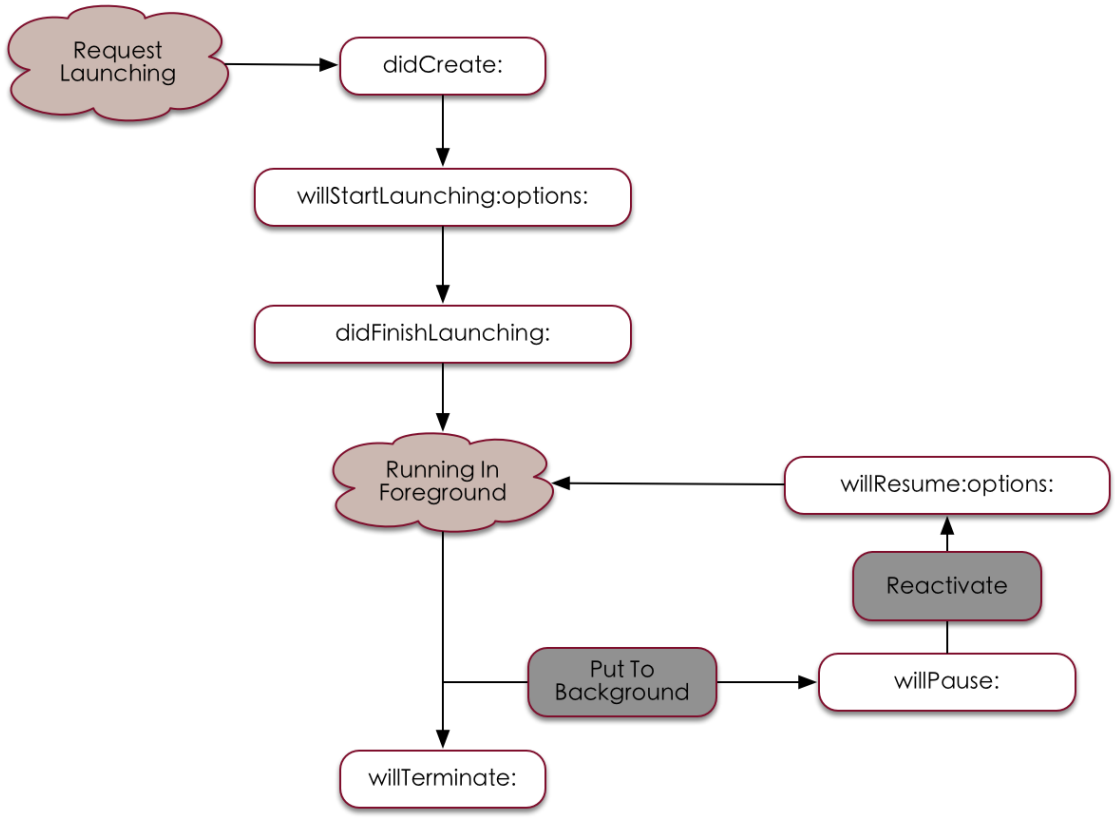
中文	英文	解释
微应用	MicroApplication	客户端运行期带有用户界面的微应用
服务	Service	客户端运行期提供的轻量级抽象服务
框架上下文	Context	客户端微组件运行期上下文

本文主要介绍微应用、服务、框架上下文的概念。有关具体的使用方法，查看 创建微应用。

微应用

在基于 mPaaS iOS 框架开发应用的过程中，一般会将带有 UI 界面的独立业务设置为一个微应用（如支付宝中的转账、手机充值等），与其他的业务隔离开，实现各个微应用之间高度独立，不相互依赖。

微应用也有自己的生命周期，整个过程如下：



微应用整个生命周期的回调方法，具体内容参考 `DTMicroApplicationDelegate.h` 文件。

```
@required
/**
 * 请求应用对象的代理返回根视图控制器。
 *
 * @param application 应用对象。
```

```
 *
 * @return 应用的根视图控制器。
 */
- (UIViewController *)rootControllerInApplication:(DTMicroApplication *)application;

@optional

/**
 * 通知应用代理，应用对象已经对经被实例化。
 *
 * @param application 应用对象。
 */
- (void)applicationDidCreate:(DTMicroApplication *)application;

/**
 * 通知应用代理，应用将要启动。
 *
 * @param application 启动的应用对象。
 * @param options 应用运行参数。
 */
- (void)application:(DTMicroApplication *)application willStartLaunchingWithOptions:(NSDict
ionary *)options;

/**
 * 通知应用代理，应用已启动。
 *
 * @param application 启动的应用对象。
 */
- (void)applicationDidFinishLaunching:(DTMicroApplication *)application;

/**
 * 通知应用代理，应用即将暂停进入后台运行。
 *
 * @param application 启动的应用对象。
 */
- (void)applicationWillPause:(DTMicroApplication *)application;

/**
 * 通知应用代理，应用将被重新激活。
 *
 * @param application 要激活的应用对象。
 */
- (void)application:(DTMicroApplication *)application willResumeWithOptions:(NSDictionary *
)options;

/**
 * 通知应用代理，应用已经被激活。
 *
 * @param application 要激活的应用对象。
 */
- (void)applicationDidResume:(DTMicroApplication *)application;

/**
 * 通知应用代理，应用已经被激活。
```

```
 *
 * @param application 要激活的应用对象，带上参数的版本。
 */
- (void)application:(DTMicroApplication *)application didResumeWithOptions:(NSDictionary *)
options;

/**
 * 通知应用的代理，应用将要退出。
 *
 * @param application 应用对象。
 */
- (void)applicationWillTerminate:(DTMicroApplication *)application;

/**
 * 通知应用的代理，应用将要退出。
 *
 * @param application 应用对象。
 * @param animated 是否以动画方式退出。
 */
- (void)applicationWillTerminate:(DTMicroApplication *)application animated:(BOOL)animated;

/**
 * 询问应用的代理，应用是否可以退出。
 * 注意：只有特殊情况才返回 NO；如果默认是 YES，则可以退出。
 *
 * @param application 应用对象。
 *
 * @return 是否可以退出。
 */
- (BOOL)applicationShouldTerminate:(DTMicroApplication *)application;
```

服务

mPaaS iOS 框架将没有 UI 界面的 Framework 称为服务，其与微应用的区别如下：

- 微应用是独立的业务流程，服务则用来提供通用服务。
- 服务有状态，一旦启动后，其在整个客户端的生命周期中一直存在，任何时候都可以被获取；微应用在退出后即被销毁。

服务管理相关的接口，具体内容参考 `DTService.h` 文件。

```
@required

/**
 * 启动一个服务。
 * 注意：
 * 框架在完成初始化操作后，会调用该方法。
 * 在一个服务里面，要先调用该方法，之后才能去启动应用。
 */
- (void)start;

@optional

/**
 * 创建服务完成。
 */
- (void)didCreate;

/**
 * 服务将要销毁。
 */
- (void)willDestroy;
```

框架上下文（Context）

框架上下文（Context）是整个客户端框架的控制中心，统一管理各个微应用和服务之间的交互与跳转，主要负责：

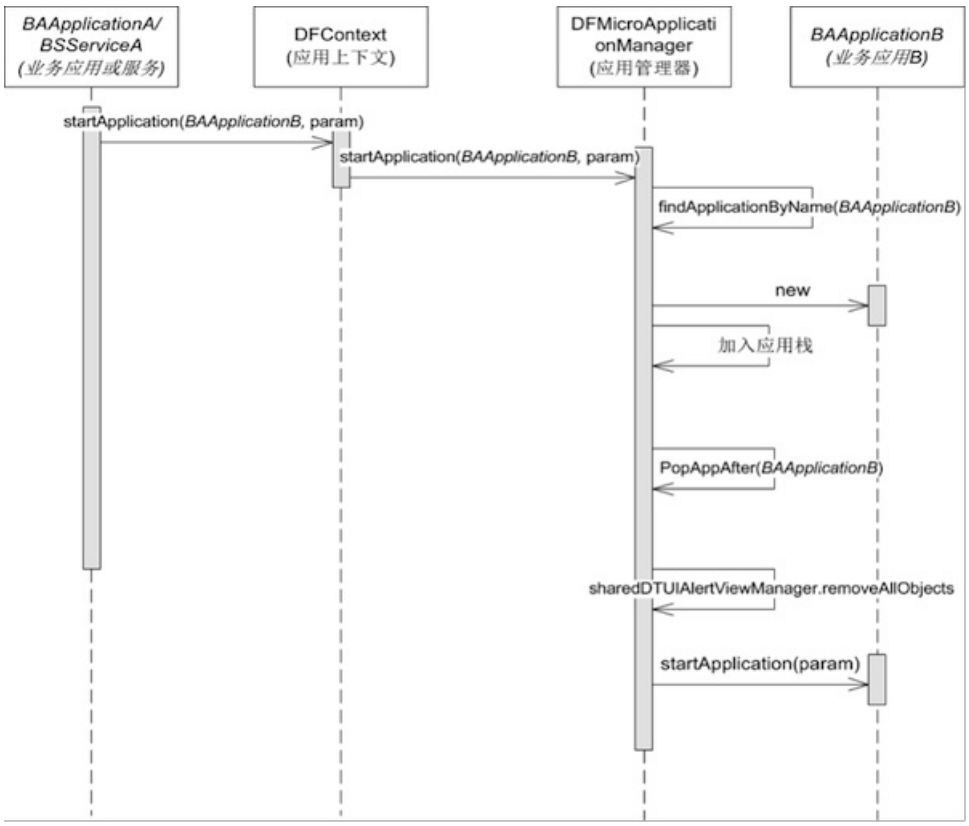
- 提供启动微应用的接口，可通过名字快速查找、关闭、管理微应用的跳转等。
- 提供启动服务的接口，管理服务的注册、发现和反注册。

微应用管理

- 微应用管理相关接口，具体内容参考 `DTContext.h` 文件。

```
/**
 * 根据指定的名称启动一个应用。
 *
 * @param name 要启动的应用名。
 * @param params 启动应用时，需要转递给另一个应用的参数。
 * @param animated 指定启动应用时，是否显示动画。
 *
 * @return 应用启动成功返回 YES，否则返回 NO。
 */
- (BOOL)startApplication:(NSString *)name params:(NSDictionary *)params animated:(BOOL)anim
ated;

/**
 * 根据指定的名称启动一个应用。
 *
 * @param name 要启动的应用名。
 * @param params 启动应用时，需要转递给另一个应用的参数。
 * @param launchMode 指定 App 的启动方式。
 *
 * @return 应用启动成功返回 YES，否则返回 NO。
 */
- (BOOL)startApplication:(NSString *)name params:(NSDictionary *)params launchMode:(DTMicro
ApplicationLaunchMode)launchMode;

/**
 * 查找一下指定的应用。
 *
 * @param name 要查找的应用名。
 *
 * @return 如果指定的应用已在应用栈中，则返回对应的应用对象。否则返回 nil。
 */
- (DTMicroApplication *)findApplicationByName:(NSString *)name;

/**
 * 返回当前在栈顶的应用，即对用户可见的应用。
 *
 * @return 当前可见的应用。
 */
- (DTMicroApplication *)currentApplication;
```

- 微应用启动过程：

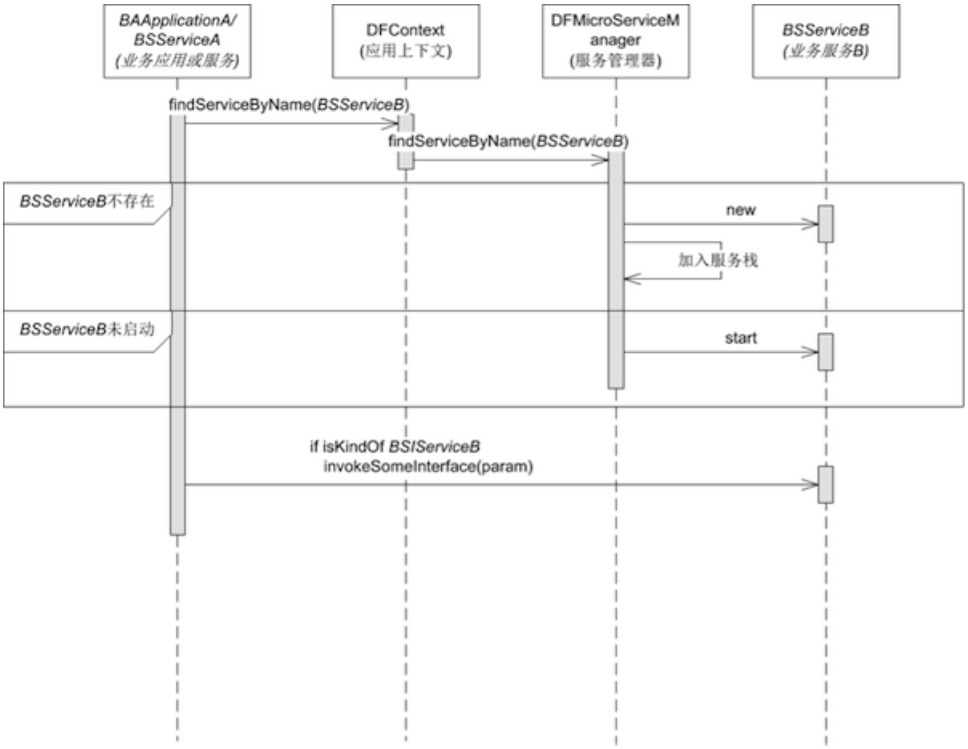


服务管理

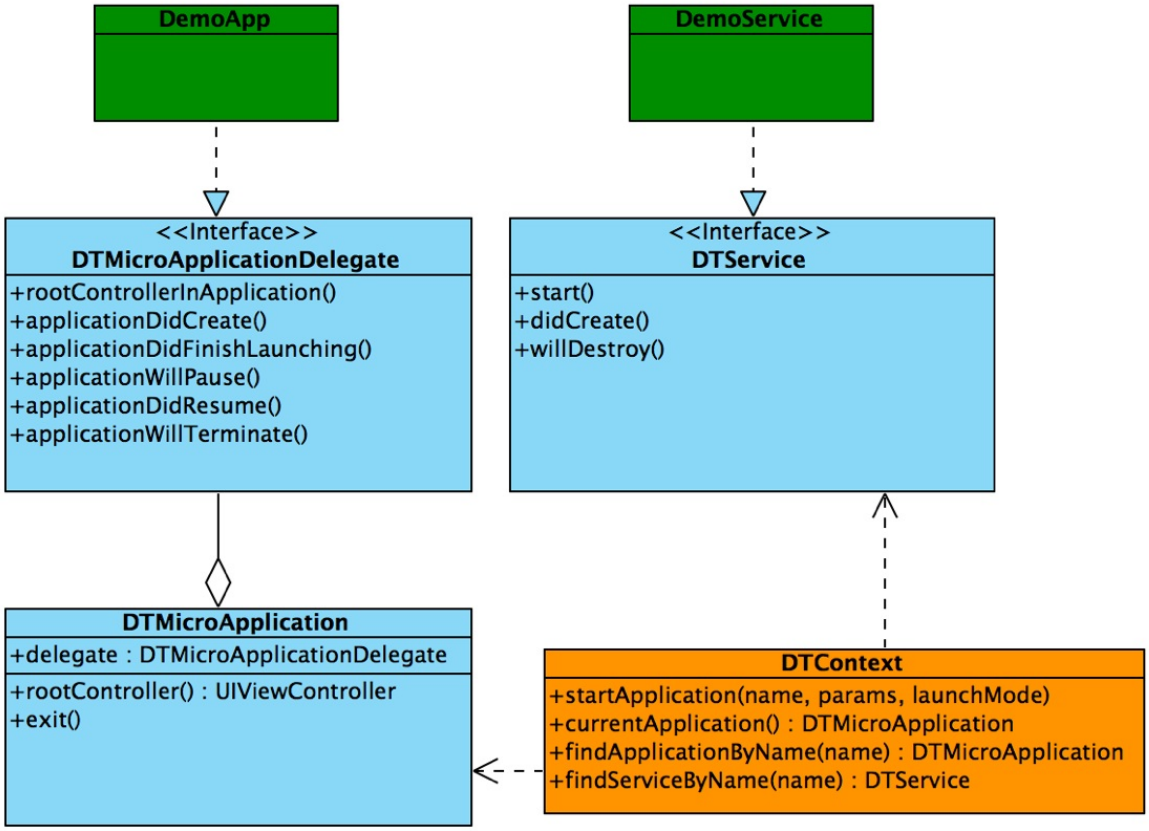
- 服务管理相关接口，具体内容参考 `DTContext.h` 文件。

```
/**
 * 根据指定的名称查找服务。
 *
 * @param name 服务名
 *
 * @return 如果找到指定名称的服务，则返回一个服务对象，否则返回空。
 */
- (id)findServiceByName:(NSString *)name;

/**
 * 注册一个服务。
 *
 * @param name 服务名
 */
- (BOOL)registerService:(id)service forName:(NSString *)name;

/**
 * 反注册一个已存在的服务。
 *
 * @param name 服务名。
 */
- (void)unregisterServiceForName:(NSString *)name;
```

- 服务启动过程：



框架上下文管理微应用与服务的 UML 类图如下：



6.3. 微应用与服务

6.3.1. 创建微应用

在基于 mPaaS iOS 框架开发应用的过程中，一般会将带有 UI 界面的独立业务设置为一个微应用（如支付宝中的转账、手机充值等），与其他的业务隔离开，在微应用内实现自身业务逻辑。要添加一个微应用，您需要添加微应用模板代码，并注册微应用。

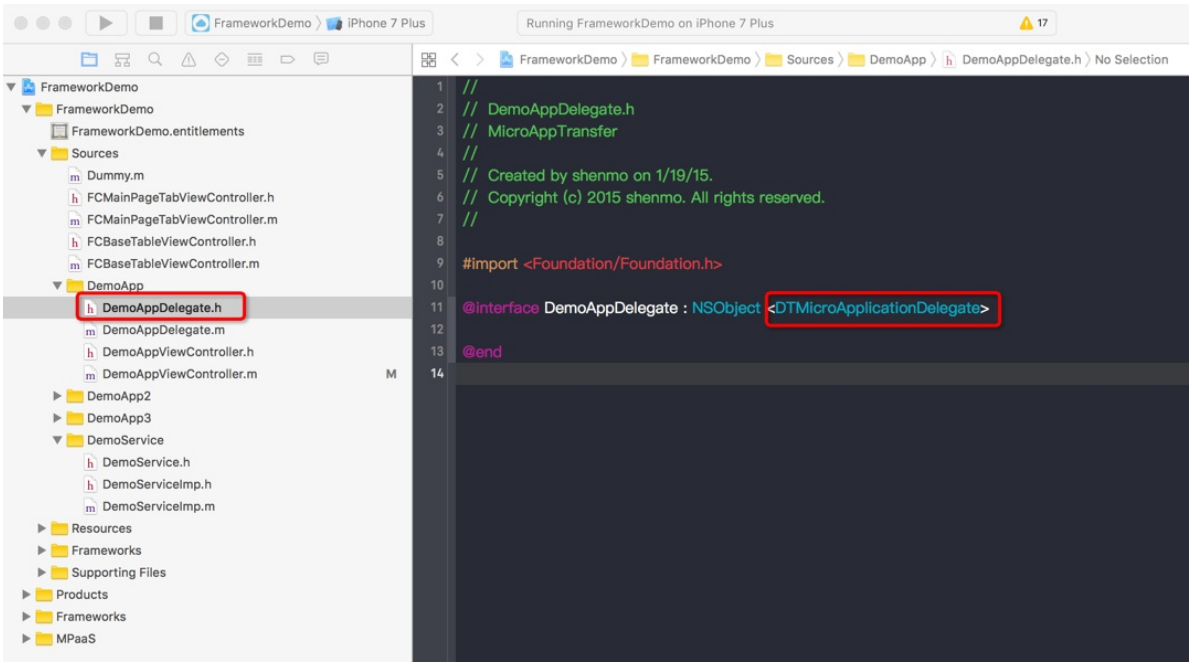
代码示例

点击 iOS framework-demo 下载 iOS 移动框架示例代码。

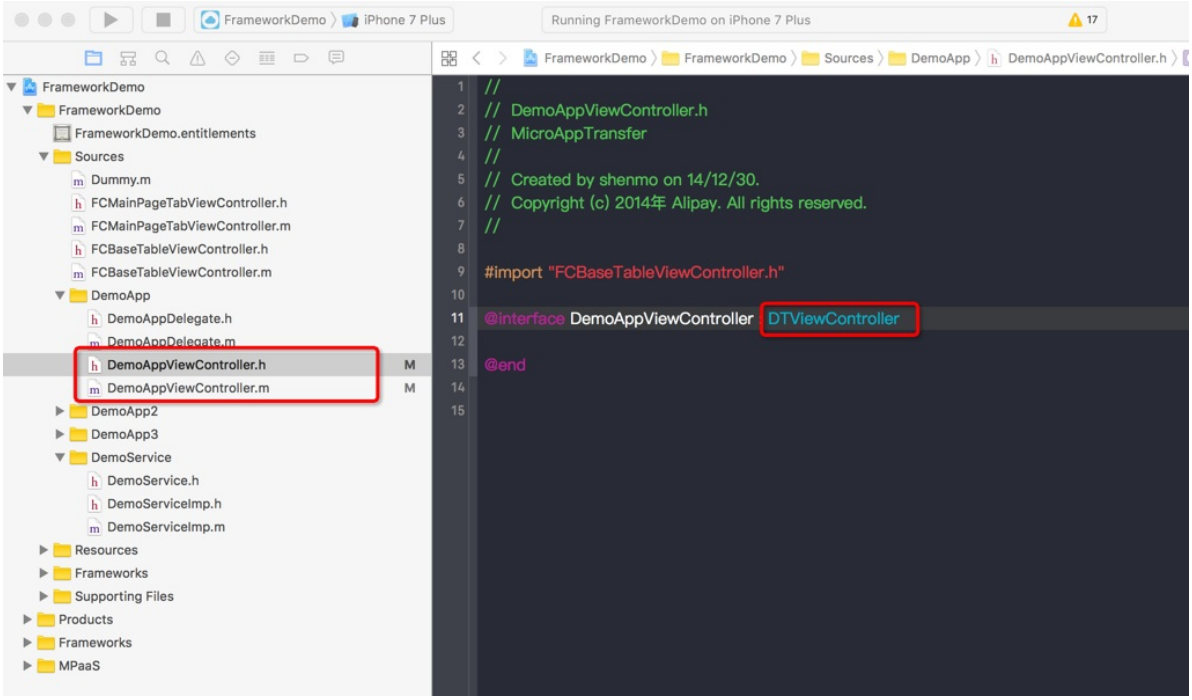
操作步骤

1. 添加微应用模板代码

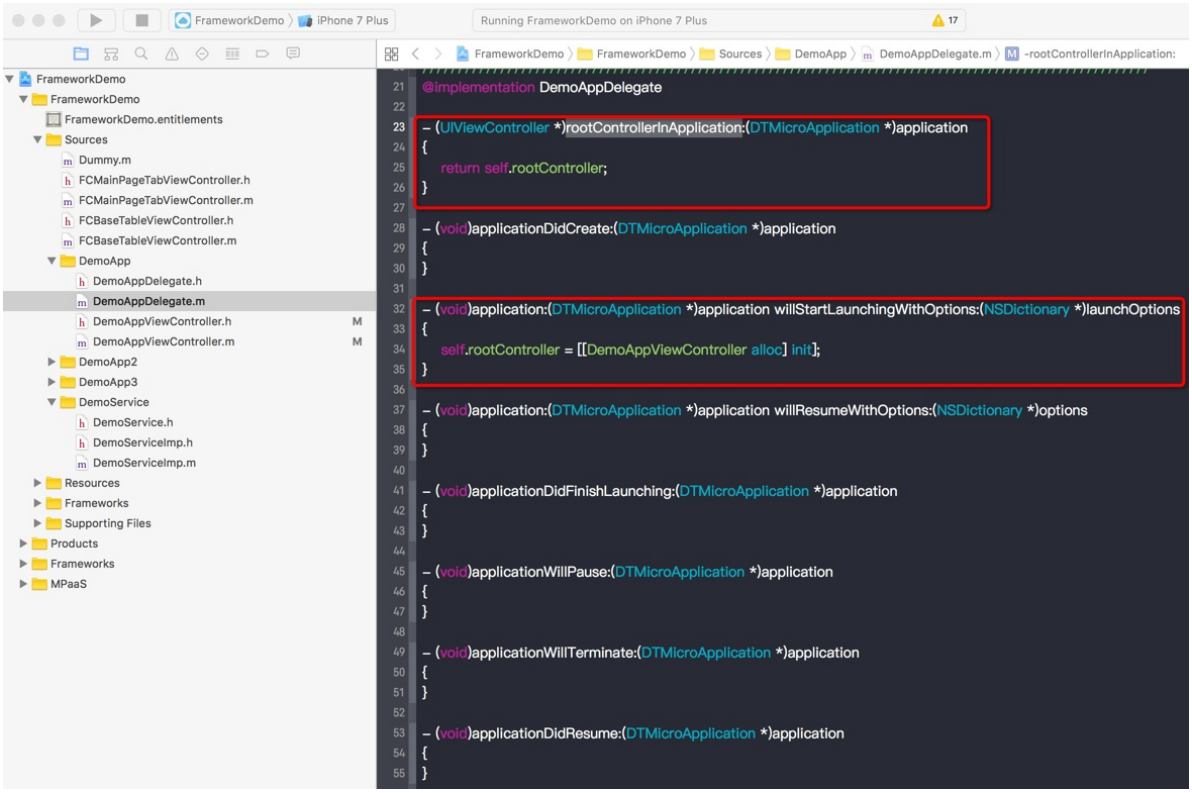
1. 创建微应用的代理类并实现 mPaaS iOS 框架的微应用管理器 `DTMicroApplicationDelegate` 的代理方法。



2. 创建微应用的 `rootViewcontroller`，可继承 mPaaS iOS 框架提供的基类 `DTViewController`。

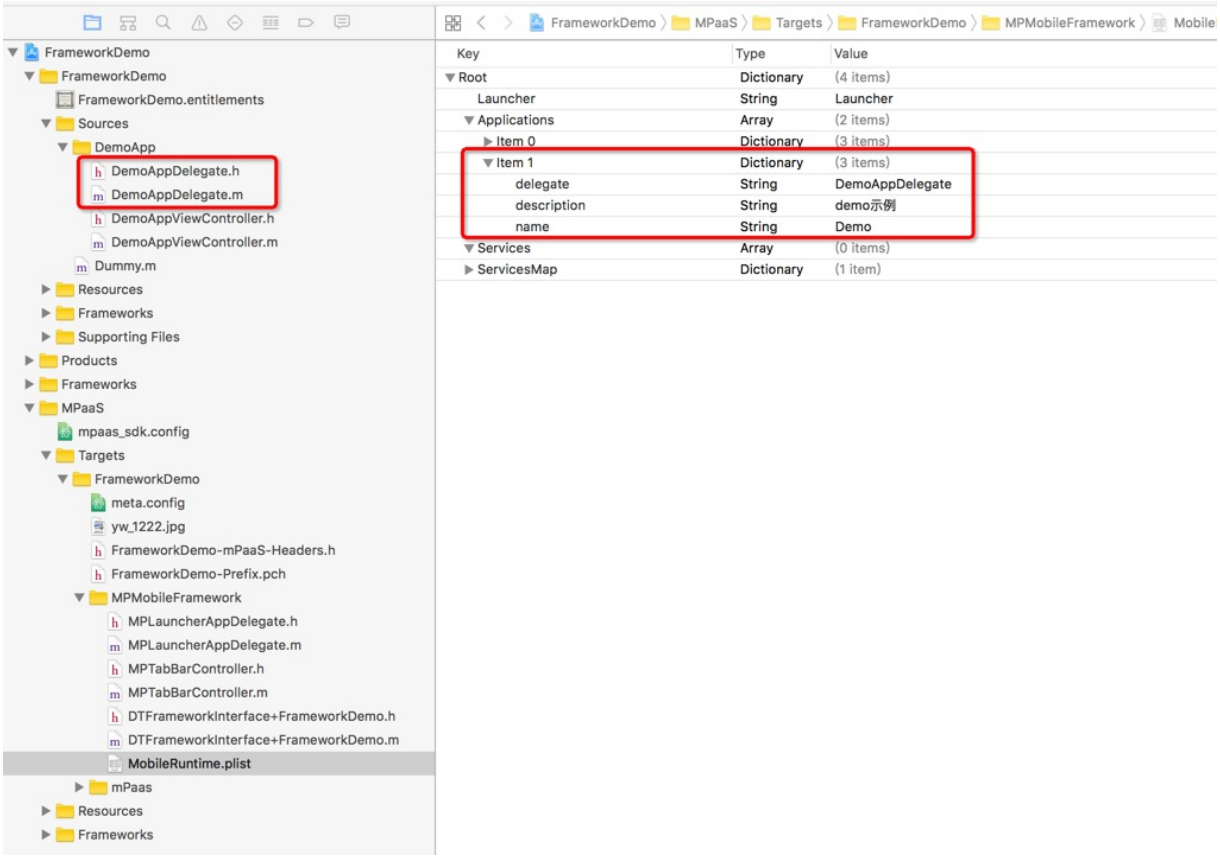


3. 指定微应用的 `rootViewcontroller`。可通过微应用的代理方法，在微应用的生命周期中进行相关的业务处理。



2. 注册微应用

创建后的微应用只有在 `MobileRuntime.plist` 中注册后，才能通过框架进行统一管理。



字段	说明
Delegate	应用实现 `DTMicroApplicationDelegate` 的类名。
Description	应用的描述。
Name	应用的名字，框架上下文通过 name 来查找应用。

6.3.2. 创建服务

在基于 mPaaS iOS 框架开发应用的过程中，没有 UI 界面且通用的功能，可以设置为服务（如登录服务），在整个 App 运行期可以方便地被其他微应用或服务获取。添加一个服务，您需要添加服务模板代码，并注册服务。

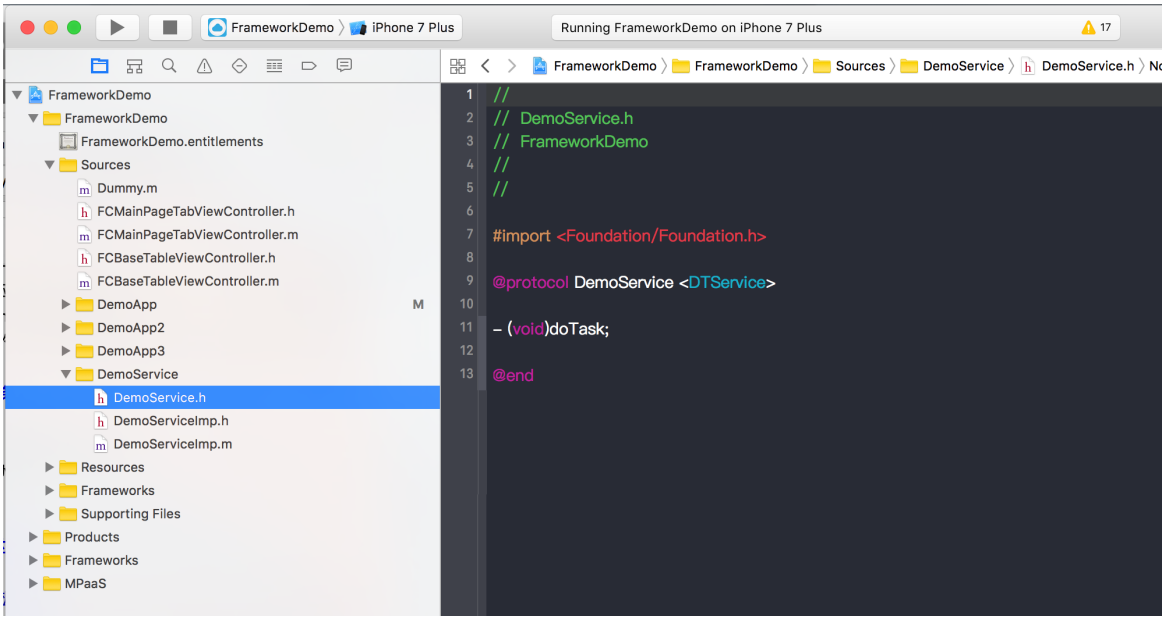
代码示例

请参考 获取代码示例 下载并查看 iOS framework-demo。

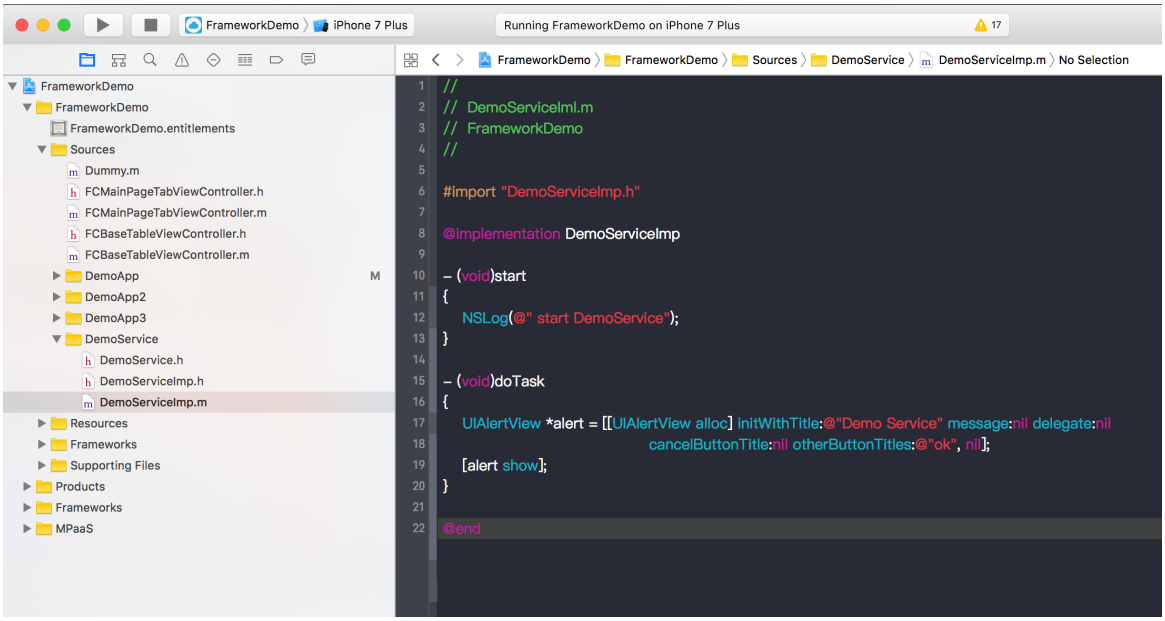
操作步骤

1. 添加服务模板代码。

i. 定义服务的协议（Protocol）并公开对外的接口方法。

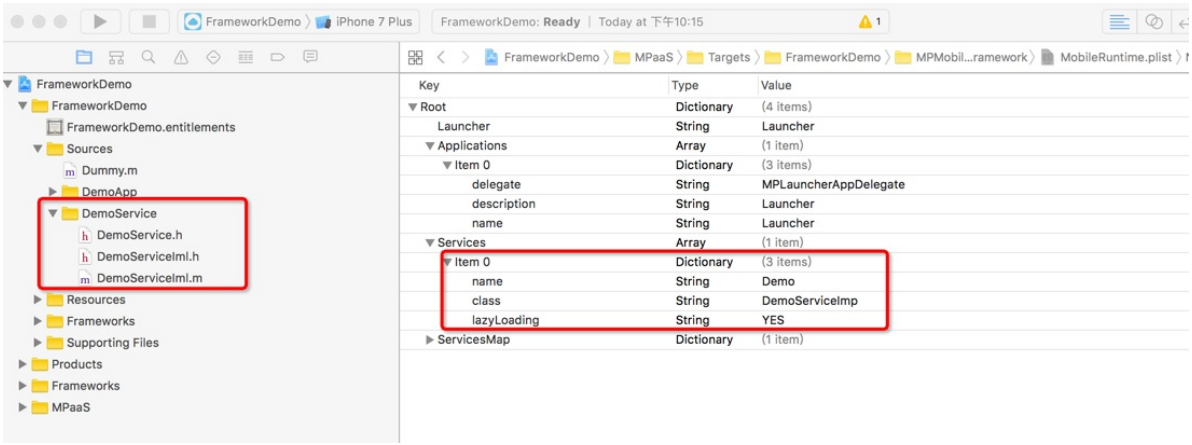


ii. 定义实现服务接口方法的类。



2. 注册服务。

同微应用一样，新创建的服务也只有在 `MobileRuntime.plist` 中注册后，才能通过框架进行统一管理。



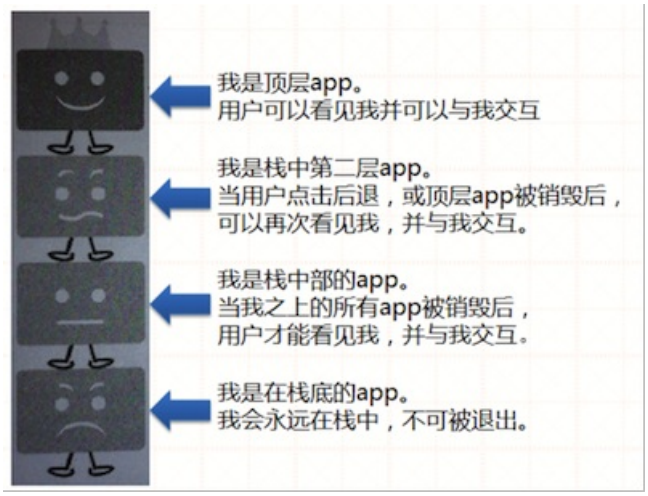
字段	说明
name	服务的唯一标识。
class	服务的实现类，框架在创建该服务时，会利用运行时的反射机制，创建服务实现类的实例。
lazyLoading	是否延迟加载。如果是延迟加载，在框架启动时，该服务不会被实例化，只有在用到时才会实例化并启动。如果是非延迟加载，在框架启动时会实例化并启动该服务。默认为 `NO` 。

6.3.3. 管理微应用与服务

将业务分割为微应用和服务后，不仅实现了不同模块之间的低耦合、高内聚，同时可以通过 mPaaS iOS 框架提供的框架上下文，进行微应用与服务的管理，包括微应用与微应用、服务与服务、微应用与服务之间的跳转和数据传递等。

管理微应用

框架上下文通过堆栈对所有微应用的跳转进行统一管理，遵循如下规则：



- 基于 mPaaS iOS 框架，可以根据微应用的 `name`，快速查找到此微应用，并在当前微应用中启动另一个微应用：

```
- (void)pushSubApp2
{

    [DTContextGet() startApplication:@"20000002" params:@{} launchMode:kDTMicroApplicat
ionLaunchModePushWithAnimation];
}
```

- 微应用堆栈中上层的微应用，可以快速跳转到堆栈底部的根应用：

```
- (void)exitToLauncher
{
    // 因为 Launcher 在下层，所以再启动 Launcher 实际是退出上层所有的 App，回到 Launcher
    [DTContextGet() startApplication:@"Launcher" params:nil animated:kDTMicroApplicatio
nLaunchModePushNoAnimation];
}
```

- 快速退出当前微应用：

```
- (void)exitSelf
{
    [[DTContextGet() currentApplication] exitAnimated:YES];
}
```

- 快速退出已启动的应用：

```
- (void)exitApp2
{
    // 当前顶层应用是 app3，但是可以强行把 app2 和它的窗口都退出。
    [[DTContextGet() findApplicationByName:@"20000002"] forceExit];
}
```

管理服务

基于 mPaaS iOS 框架，可以快速在当前微应用中启动另一个服务。

```
- (void)findService
{
    id<DemoService> service = [DTContextGet() findServiceByName:@"DemoService"];
    [service doTask];
}
```

6.3.4. 微应用层级代码示例

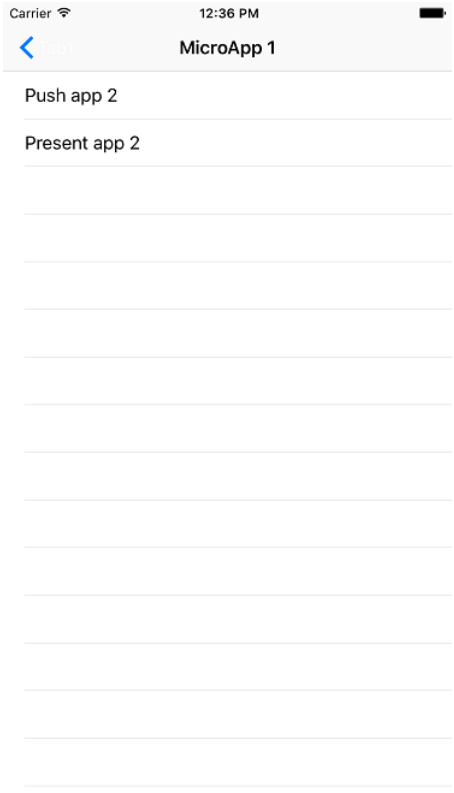
本代码示例介绍 mPaaS 微应用之间的层级关系。有关 iOS 框架的详细介绍，查看 mPaaS 框架介绍。

下载代码

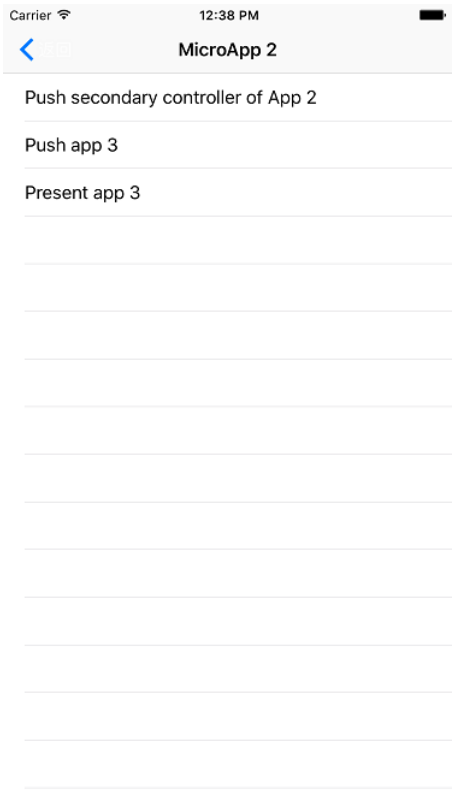
请参考 获取代码示例 下载示例代码。微应用层级演示工程为 `mpaas_demo_ios/FrameworkDemo` 。

微应用层级演示

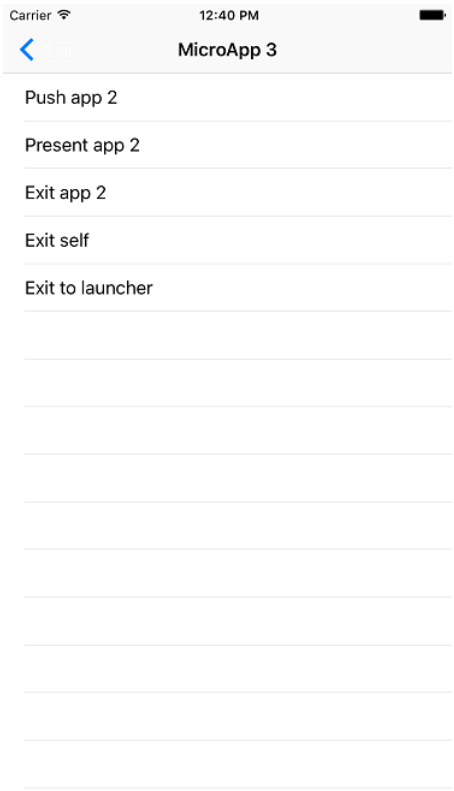
- 点击 Push app，启动 MicroApp 1；



- 点击 Push app 2，启动 MicroApp 2；

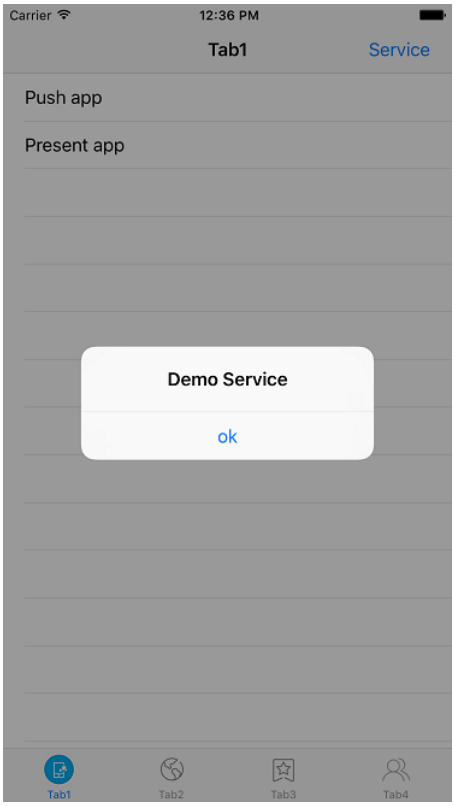


- 点击 **Push app 3**，启动 MicroApp 3；



- 当前顶层应用是 app 3，可以通过 **Exit app 2** 强行把 app 2 和它的窗口都退出；
- 当前顶层应用是 app 3，可以通过 **Exit self** 将 app 3 自身退出，返回到 app 2；

- 当前顶层应用是 app 3，可以通过 **Exit to launcher** 直接退回到根应用，强行将 app 1、app 2、app 3 都退出；
- 同时，可以在 **Launcher** 的微应用中，启动一个服务。



6.4. iOS 语言设置

本文介绍了在将 mPaaS 接入 iOS 客户端过程中设置语言的实现方法。

在接入 iOS 过程中，您可对 iOS 应用进行语言设置。

默认跟随系统语言

1. 您可在工程中添加 Languages.bundle.zip 来设置当前 App 支持的语言。
2. 在应用启动完成时，初始化多语言框架：

```
//#import <mPaas/APLanguage.h>
[APLanguageSetting sharedSetting];
```

获取 App 当前语言

您可通过以下方式获取 App 当前语言：

```
NSString *currentLanguage = [APLanguageSetting currentLanguage].name;
```

修改 App 当前语言

在工程的 `Languages.bundle` 中，您可查看当前 App 支持的语言，您可通过以下方式修改 App 当前语言：

```
[APLanguageSetting setCurrentLanguageWithName:@"en"];
```

文案支持多语言

1. 添加多语言 bundle 文件。
   i. 根据当前 App 支持的语言，添加对应的 strings 文件。
   ii. 设置多语言文件的路径：

   ```
   [[APLanguageBundleLoader sharedLoader] setCustomLanguagesBundlePath:@""];
   ```

2. 实现 strings 文件。

   strings 文件的实现原则如下：

   - strings 文件中每一个文案格式如下，等号左侧标识文案的 key，等号右侧字符串标识文案在此语言下的展示内容：

     `"BeeCityPicker : 城市选择"="城市选择"`

   - 对于同一文案，在所有 strings 文件中的 key 必须一致。key 的定义，建议以 bundle 名与文案中文内容拼接而成，如 `"BeeCityPicker: 城市选择"` 。

3. 设置文案。

   对需要支持多语言的文案，请勿写死，可使用 `__Text` 宏进行复制，如下所示：

   ```
   self.navigationItem.title = __TEXT(@"BeeCityPicker",@"BeeCityPicker:城市选择", @"城市选择"
   );
   ```

   - `@"BeeCityPicker"` ：为文本在字符串表所在 `bundle` 名，通常为模块资源 bundle 名称。
   - `@"BeeCityPicker : 城市选择"` ：为文本在字符串表中的 key。
   - `@"城市选择"` ：为当在对应字符串表中找不到 key 对应的文本内容时，默认返回的内容。

6.5. 自定义选择城市

本文介绍了在将 mPaaS 接入 iOS 客户端过程中自定义选择城市的实现方法。

在接入 iOS 过程中，您可以自定义选择城市。

> **说明**
>
> 此功能仅在基线版本 ≥ 10.1.68.27 时生效。

自定义城市列表文件

所有城市

1. 新建城市列表文件，文件以 `.txt` 结尾，文件内容格式如下：
   - 字段 1：adcode。
   - 字段 2：城市名。
   - 字段 3：城市名的拼音，用于配置右侧首字母列表。

2. 设置自定义城市列表路径。文件路径为相对于 bundle 的相对路径，如 `BeeCityPicker.bundle/citiesWithCounty.text`，SDK 内部会自动读取此文件名：

```
[BeeCityPickerAdapter sharedInstance].customHotCityTextFile = @"BeeCityPicker.bundle/citi
esWithCounty.text";
```

热门城市

1. 新建城市列表文件，热门城市列表文件同 所有城市。
2. 设置自定义热门城市列表路径。

```
[BeeCityPickerAdapter sharedInstance].customHotCityTextFile = @"BeeCityPicker.bundle/hotC
ities.text";
```

小程序中自定义城市列表

关于如何在小程序中自定义城市列表，请参见 选择城市。

7.开发者工具

7.1. 关于开发者工具

mPaaS 提供了以下开发者工具以帮助开发者能够快速接入 mPaaS 并辅助进行开发工作。

- mPaaS Xcode Extension：是基于 XcodeKit 构建的官方插件，它可以在 Xcode 的 Editor 菜单中增加额外的命令。安装简单，不需要去除 Xcode 签名。通过全新的图形化界面，帮助您更快速地接入 mPaaS。mPaaS Xcode Extension 支持新建 mPaaS 工程、编辑工程（导入云端配置文件，添加删除 mPaaS 组件，升级 mPaaS 基线，生成无线保镖图片）、基础工具（工程打包工具等）。
- 开发小助手：DevHelper 提供了常用的辅助开发工具和 mPaaS 专用辅助工具，该工具集通过 mPaaS 插件提供给开发者。在将开发小助手接入开发工程后，您就可以使用开发小助手进行调试、帮助开发。
- 公有云答疑小助手：公有云答疑小助手是 mPaaS 研发团队提供的钉钉群答疑机器人，它不仅可以帮助开发者快速了解、接入 mPaaS 框架，还可以协助排查使用 mPaaS 框架过程中出现的常见问题，提高开发效率。
- mPaaS 插件：是一个具有图形化界面的插件工具。功能包括新建 mPaaS 工程、增删 mPaaS 组件依赖、工程打包等。因为 mPaaS 插件需要和命令行协同工作，因此在安装 mPaaS 插件时，会同时安装命令行工具。

> **说明**
>
> - mPaaS Xcode Extension 和 mPaaS 插件两者因为功能冲突，不能同时安装。
> - 命令行工具是一套 mPaaS 相关的终端命令集，在安装 mPaaS 插件时一起安装。通过命令行工具，您可以在终端执行命令查看 SDK 版本、增删 mPaaS 组件依赖、去除和恢复 Xcode 签名等。
> - 自 2020 年 9 月 1 日起，mPaaS 已停止维护 Xcode mPaaS 插件，请您选用 mPaaS Xcode Extension 插件。

7.2. mPaaS Xcode Extension

7.2.1. 关于 mPaaS Xcode Extension

mPaaS Xcode Extension 是基于 XcodeKit 构建的官方插件。该插件安装简单，且不需要去除 Xcode 签名，安全可靠。同时也提供了图形化界面，帮助您更便捷地接入 mPaaS。mPaaS Xcode Extension 支持新建 mPaaS 工程、编辑工程包括导入云端配置文件、添加删除 mPaaS 组件、升级 mPaaS 基线、生成无线保镖图片，基础工具（工程打包工具等）。除上述基础功能外，mPaaS Xcode Extension 还具备以下特性，使其能够更大程度地支持开发者进行开发工作。

> **重要**
>
> mPaaS Xcode Extension 和 mPaaS 插件两者因为功能冲突，不能同时安装。

mPaaS Xcode Extension 特性

贴近 macOS 原生的 UI 设计

对 mPaaS Xcode Extension 的 UI 进行了全局考量和整体设计，视觉和交互更加友好，更贴近 macOS 生态。对深色模式的适配更加出色，实现对系统主题实时切换的适配。

稳定优异的性能

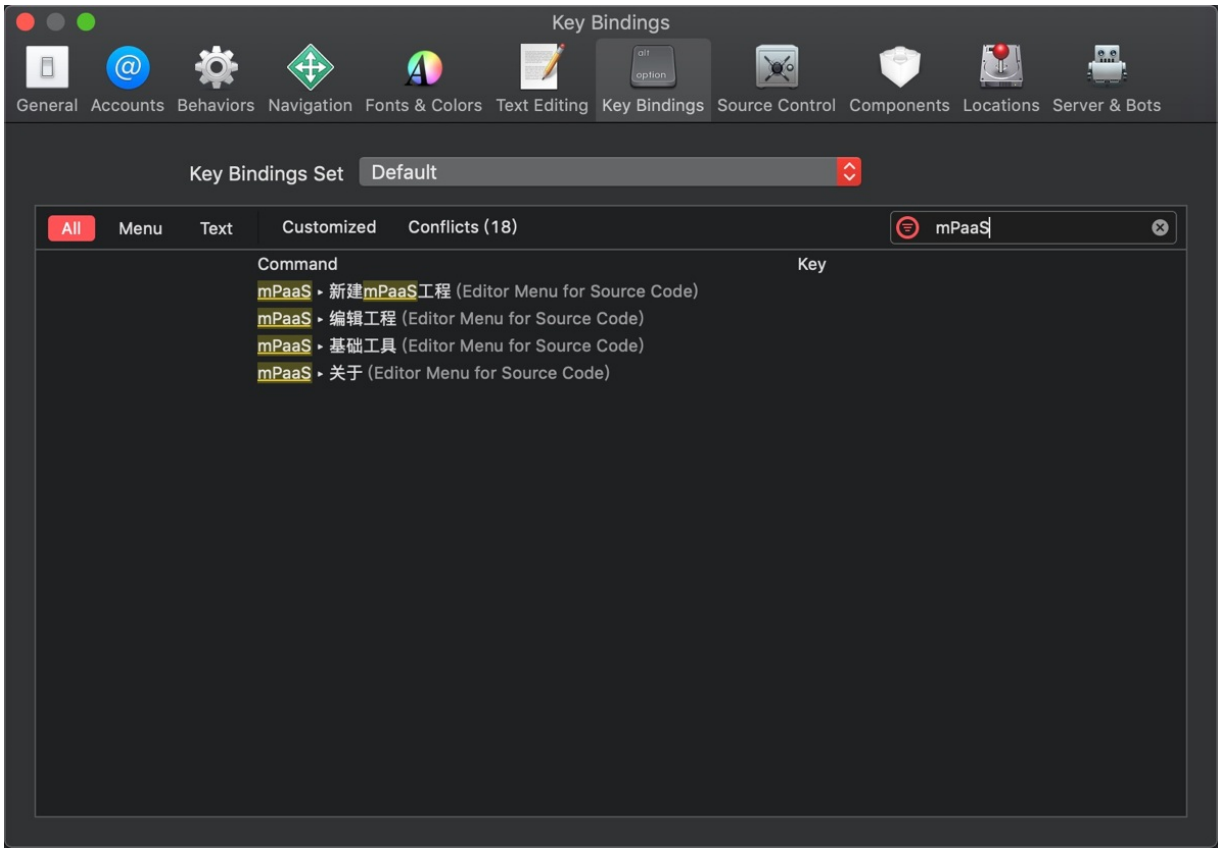
mPaaS Xcode Extension 独立于 Xcode 进程单独运行，不再影响 Xcode 的原生功能。从此告别 Xcode 启动卡死、操作无响应、系统内存飙高等困扰。

启动方式

在macOS的 **应用程序** 中找到 mPaaS 独立的应用 **mPaaSPlugin**，直接运行即可。

支持自定义快捷键

mPaaS Xcode Extension 支持用户为 mPaaS 菜单添加自定义快捷键，方便快捷并且个性十足。打开 Xcode 偏好设置，选择 Key Bindings，过滤 mPaaS 即可设置，如下图所示。



7.2.2. 安装 mPaaS Xcode Extension

本文将向您介绍安装 mPaaS Xcode Extension 的具体操作。

重要

- 如果您已经安装了旧版本 mPaaS 插件（去签名版本），需要先将 Xcode 签名恢复，或者直接卸载旧版插件，卸载方法请参考文档 卸载 mPaaS 插件。
- mPaaS Xcode Extension 在 Xcode 升级之后依然生效，您无需因升级 Xcode 重新安装 mPaaS Xcode Extension。

前置条件

- 安装 Xcode。目前 mPaaS Xcode Extension 仅支持 9.0及以上版本的 Xcode。
- macOS 版本 ≥ 10.13。
- 未安装过 mPaaS Xcode Extension（如需升级安装，建议使用 Extension 插件内的升级功能）。

操作步骤

1. 在终端运行以下安装命令。

```
curl -sSL https://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/mpaaskit/Xcode-extension/install.sh | sh
```

2. 安装完成后，会打开引导页面。
   i. 如果您需要创建新工程，请关闭引导页，在启动台中找到 mPaaSPlugin，单击打开。更多信息，请参见 新建工程。
   ii. 如果您需要编辑已有工程，在引导页中单击 **开始使用**，会打开 mPaaSPlugin 插件。在插件菜单中选择 **编辑工程**，即可通过拖拽工程文件或单击 **打开工程** 按钮打开已有工程。更多信息，请参见 编辑工程。

后续操作

进行系统授权。由于 mPaaS Xcode Extension 需要对 Xcode 工程进行操作，因此在第一次使用“编辑工程”功能的时候，系统需要用户进行授权。请务必允许，否则插件无法按预期工作。

7.2.3. 使用 mPaaS Xcode Extension

本文将对 mPaaS Xcode Extension 的使用进行详细介绍，主要包括以下方面：

- 启动 mPaaS Xcode Extension
- 新建工程
- 编辑工程
- 基础工具
- 常见问题
- Preferences 设置
- 评分及评论

启动 mPaaS Xcode Extension

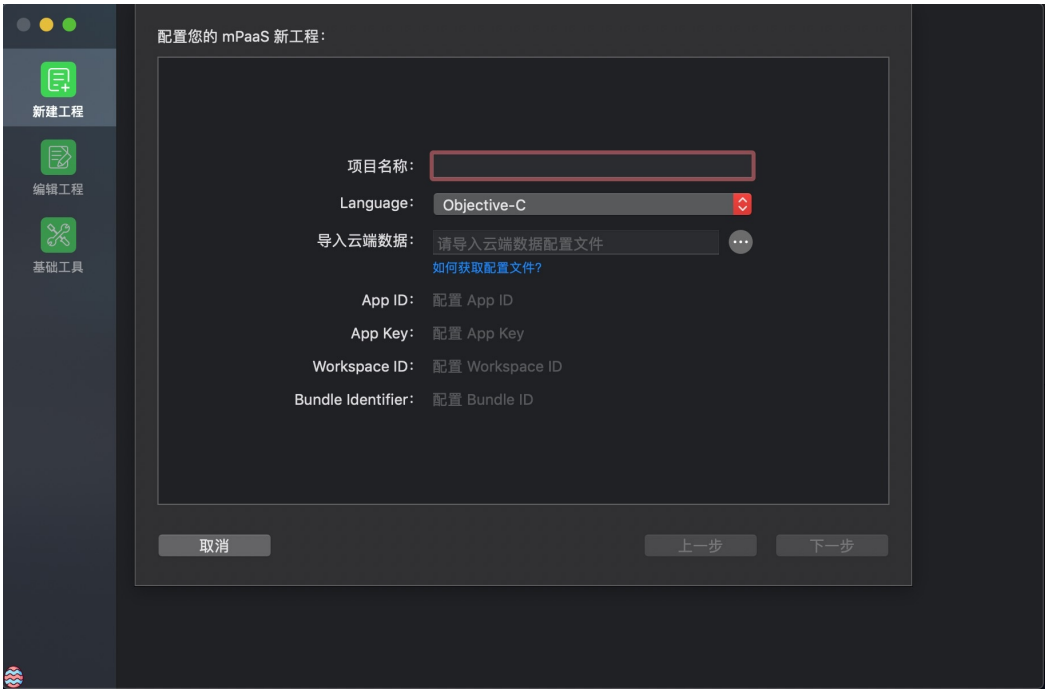
在 macOS 的 **应用程序** 中找到 mPaaS 独立的应用 **mPaaSPlugin**，单击直接运行即可。

新建工程

创建一个全新的、基于 mPaaS 框架的工程，所需步骤如下。

1. 单击 **mPaaS > 创建工程**。

2. 配置 mPaaS 新工程，输入 项目名称 并导入下载的 `.config` 配置文件，然后单击 下一步 按钮。



3. 选择要创建应用的 模板类型，然后单击 下一步 按钮。

4. 选择工程使用的 基线类型和版本，然后单击 下一步 按钮。

基线类型说明：

- **标准基线**：公开的标准 mPaaS SDK 版本，可以在下拉列表中选择。

- **定制基线**：目前只开放给专有云用户，在输入框中填入对应的基线 ID 后会自动进行校验。校验失败会提示错误信息。校验正确后即可使用。

5. 选择工程需要添加的模块，然后单击 下一步 按钮。

> **说明**
>
> 您也可以先不添加模块，在创建工程之后，再参考各组件的接入文档，使用 编辑工程 功能添加所需的模块。

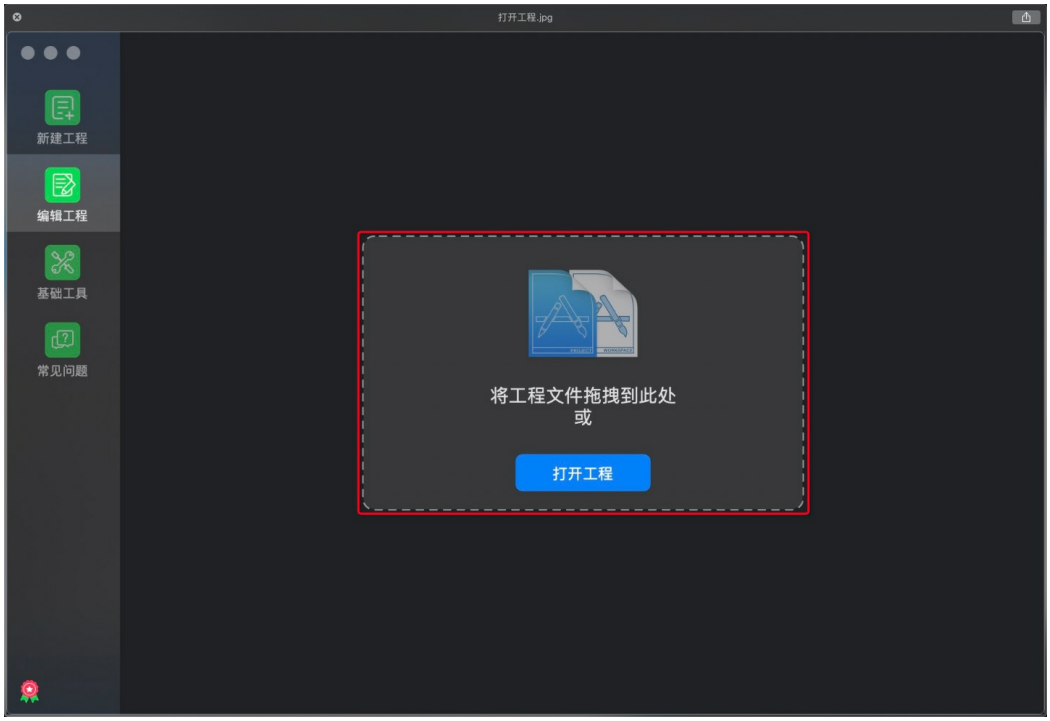
6. 选择项目保存的目录，单击 创建 按钮。

7. 开始创建工程，创建成功之后会自动打开新建的 Xcode 工程。

编辑工程

打开工程

如果您已经有 Xcode 工程，那您也可以通过插件直接打开现有工程。将工程文件拖拽进插件、或者单击插件后定位到工程文件即可。



工程面板介绍

单击 mPaaS > 编辑工程，打开编辑工程页面，页面分为若干区域，如下图所示。

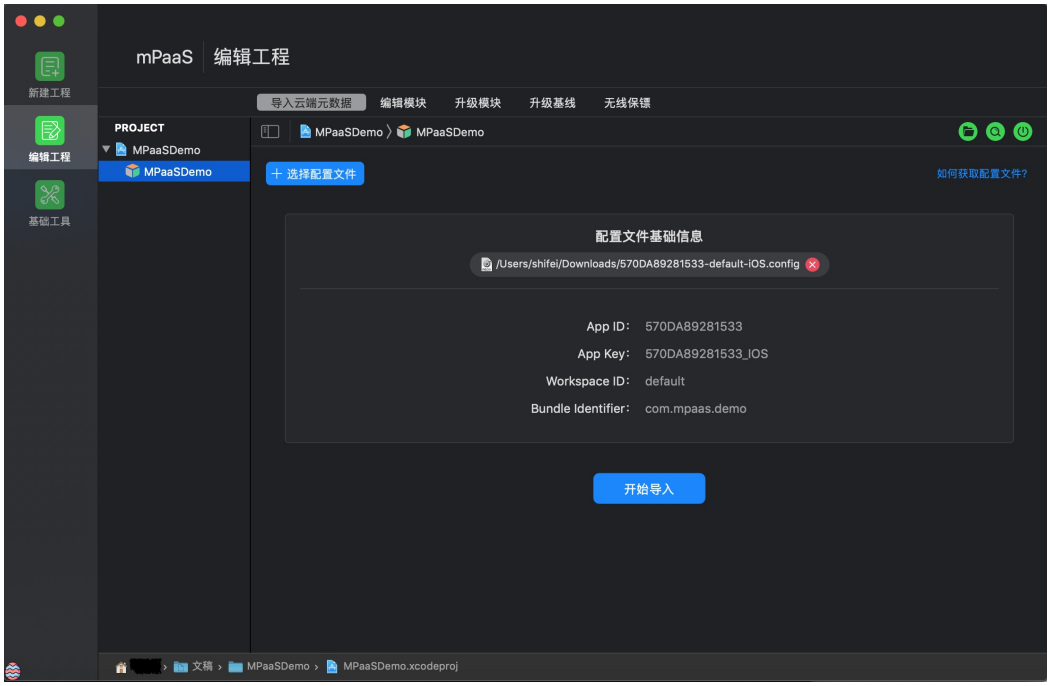
- **菜单栏**：展示所有工程的操作菜单。
- **工具栏**：展示常用的工具和当前操作的 target 信息。

- **状态栏**：展示当前编辑的工程路径。
- **工程结构栏**：展示工程的结构，包括 workspace、project、target 的结构。
- **工作区**：主要的操作区域，根据选择不同的编辑菜单展示不同。

导入云端元数据

单击 **导入云端元数据** 菜单，单击 **选择配置文件** 按钮，选择下载的云端数据配置文件，页面中可以预览该配置文件的基础信息，单击 **开始导入** 按钮，将对应的配置导入到当前工程中。

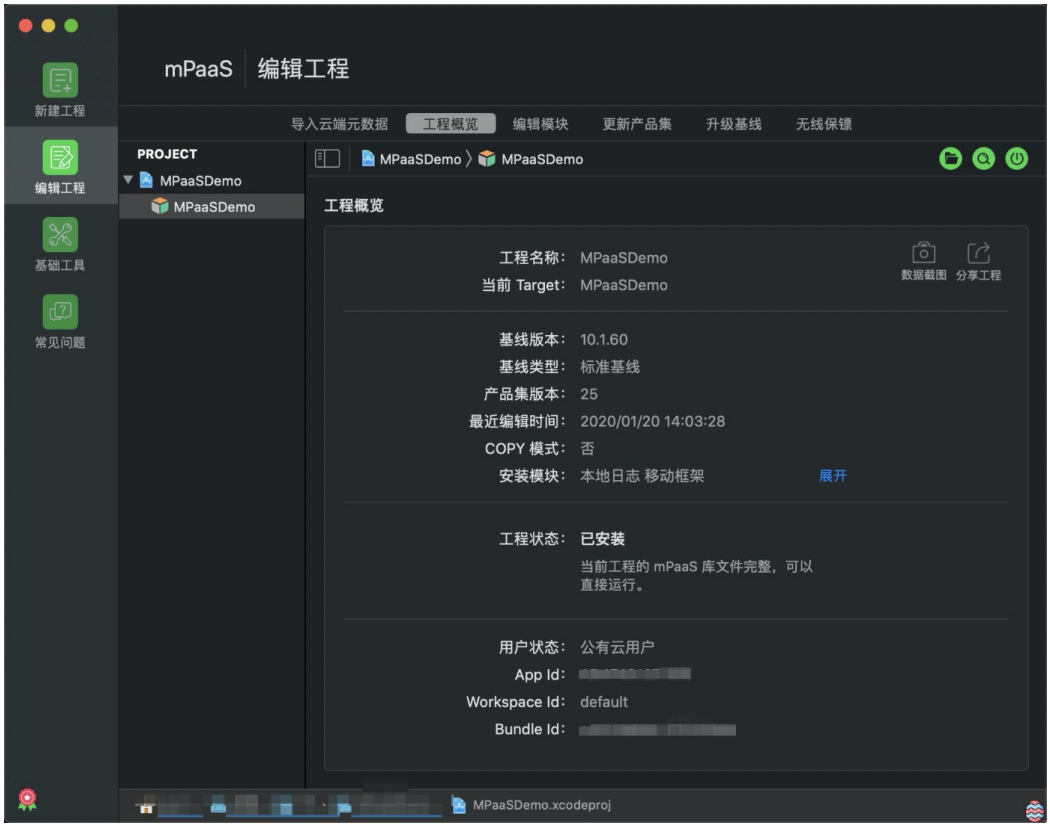
> **说明**
>
> 选择配置文件之后，可以单击 “X” 按钮来关闭，只有单击 **开始导入** 之后配置才会生效。



工程概览

单击 **工程概览** 菜单，工作区会解析工程配置，展示工程的详细信息，包括工程基础信息，mPaaS 基线信息，工程状态，应用信息。相关操作包括：

- 模块查看：单击 **展开** 按钮，弹框展示集成的所有模块。
- 工程状态：
  - 未知：表示未集成 mPaaS 或者解析信息失败。
  - 未安装：表示 mPaaS 库文件缺失，不可正常运行，单击 **安装** 按钮可以完成安装操作。
  - 已安装：表示 mPaaS 库文件已经正确安装，工程可以直接运行。



编辑模块

单击 **编辑模块** 菜单，工作区会根据工程属性展示不同的页面。

已经集成 mPaaS 的工程

1. 编辑已经集成 mPaaS 的工程，工作区会展示工程中已经集成的模块列表。

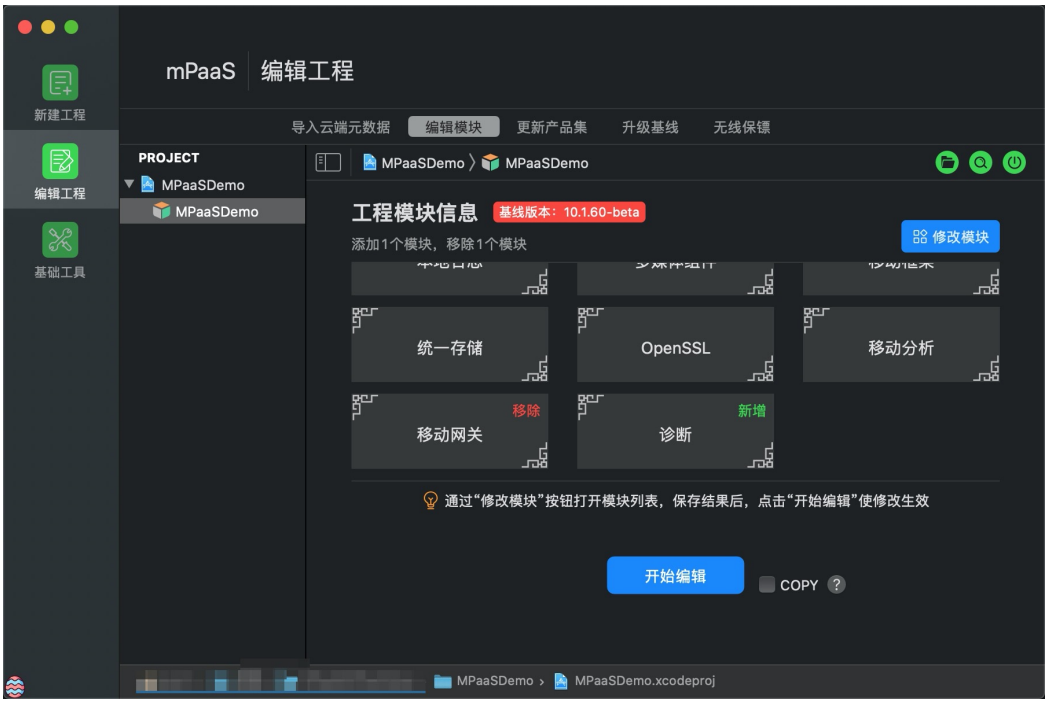
2. 单击 **修改模块** 会弹出 **模块列表** 编辑页面。

- 在列表中单击模块卡片来添加或取消模块。

- 单击每个模块右侧的 **详情** 按钮可以查看该模块的具体信息，包括描述、Release Note、包含的 framework 文件详情等。

- 单击左上角的 **X** 可以关闭模块列表，返回到主页面，选择的结果不会生效。

- 单击右上角的 **保存** 按钮保存编辑结果，同时返回到主页面，工作区的模块列表中会展示编辑的结果。

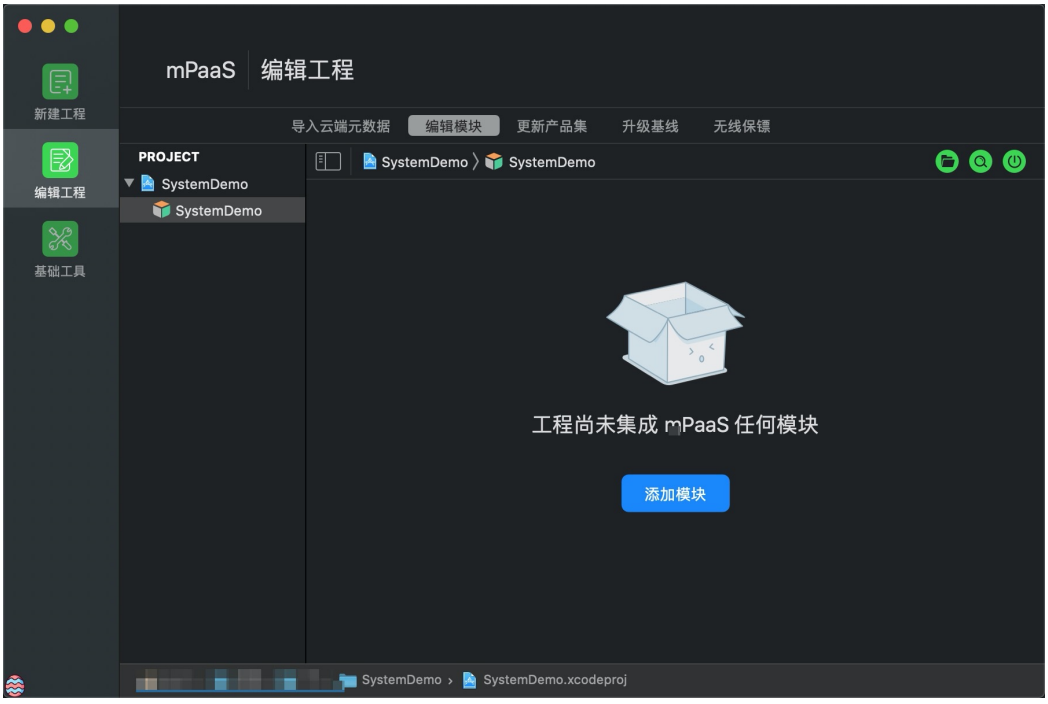


3. 返回主页面后，单击 **开始编辑**，将修改的模块添加到当前工程中或从当前工程中移除。

> **说明**
>
> 只有单击开始编辑之后，结果才会生效。

原生系统工程

1. 编辑原生的系统工程，工作区会展示未集成的提示页面。



2. 单击 **添加模块** 会弹出 **模块列表** 编辑页面。

   i. 进入页面会提示选择集成的 **基线类型和版本**，选择好基线后，单击 **确认** 按钮会收起基线选择菜单，下方刷新出对应基线的模块列表。

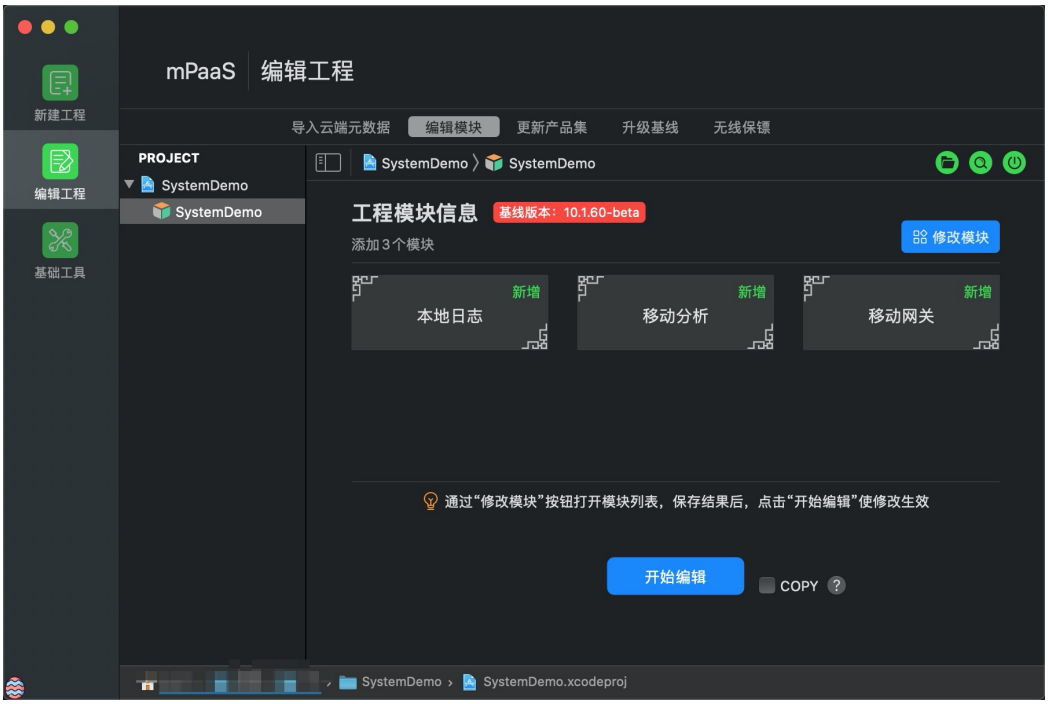
   ii. 进入模块列表后，单击右上方的 **选择基线** 下拉菜单可以重新选取基线版本，切换基线后，之前添加的结果会丢失。

- 在列表中单击模块卡片来添加模块。

- 单击每个模块右侧的 **详情** 按钮可以查看该模块的具体信息，包括描述、Release Note、包含的 framework 文件详情等。

- 单击左上角的 **X** 可以关闭模块列表，返回到主页面，选择的结果不会生效。

- 单击右上角的 **保存** 按钮保存编辑结果，同时返回到主页面，工作区的模块列表中会展示编辑的结果。



3. 返回主页面后，单击 **开始编辑**，将修改的模块添加到当前工程中或从当前工程中移除。

> **说明**
>
> 只有单击开始编辑之后，结果才会生效。

编辑模块的相关操作及说明

- **选择基线：**
  - **标准基线：** 公开的标准 mPaaS SDK 版本，可以在下拉列表中选择。

  - **定制基线：** 目前只开放给专有云用户，在输入框中填入对应的基线 ID 后会自动进行校验。校验失败会提示错误信息。校验正确后即可使用。

- **修改模块：** 打开模块列表，进行模块的添加或删除操作。
- **COPY：** 建议启用该项。选中该项后，依赖的 SDK 文件会自动拷贝到工程目录下。
- **版本：** 在工作区会展示当前工程集成 mPaaS SDK 的版本信息。

- **开始编辑**：执行模块编辑操作。

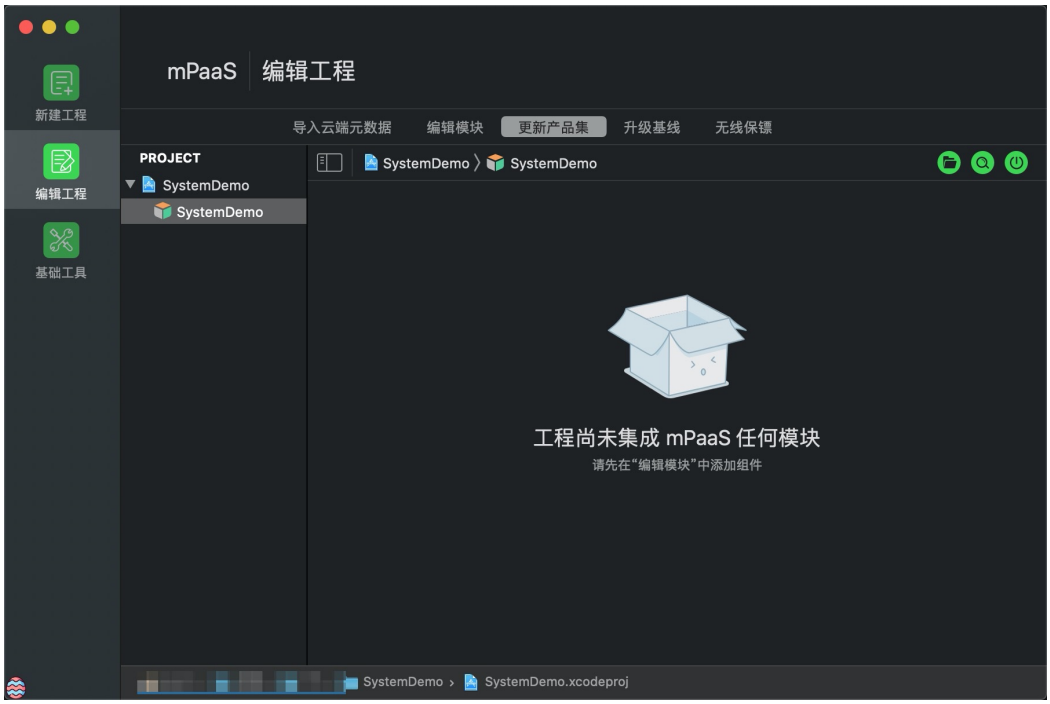
更新产品集

选择 **更新产品集** 后，单击 **开始更新** 按钮执行升级，集成的 mPaaS SDK 产品集会更新到最新的版本。

在工作区会展示当前工程的产品集信息和基线所在的最新产品集信息，单击 **查看模块更新详情** 按钮，会弹出模块列表页面，可以查看每个模块的更新状态。

单击每个模块右侧的 **详情** 按钮，可以查看该模块的具体信息，包括描述、Release Note、包含的 framework 文件详情等。

如果当前工程未集成 mPaaS，展示提示页面，建议先在 **编辑模块** 进行集成。



更新产品集的相关操作及解释如下：

- **当前产品集版本**：已集成产品集会展示当前的产品集版本，如果工程未集成产品集，右上角展示当前的基线版本。
- **最新产品集版本**：展示最新产品集的版本号，版本号结构为“基线版本+产品集版本”，如 10.1.60-beta (20)。
- **模块更新数量提示**：提示当前工程集成的模块在最新产品集中更新的数量。

- **查看模块更新详情**：弹出模块列表页面，可以查看每个模块的更新状态。
- **查看 Release Note**：打开文档“SDK 发布说明”。
- **产品集更新时间**：展示最新产品集的发布时间。
- **开始升级**：执行升级操作。

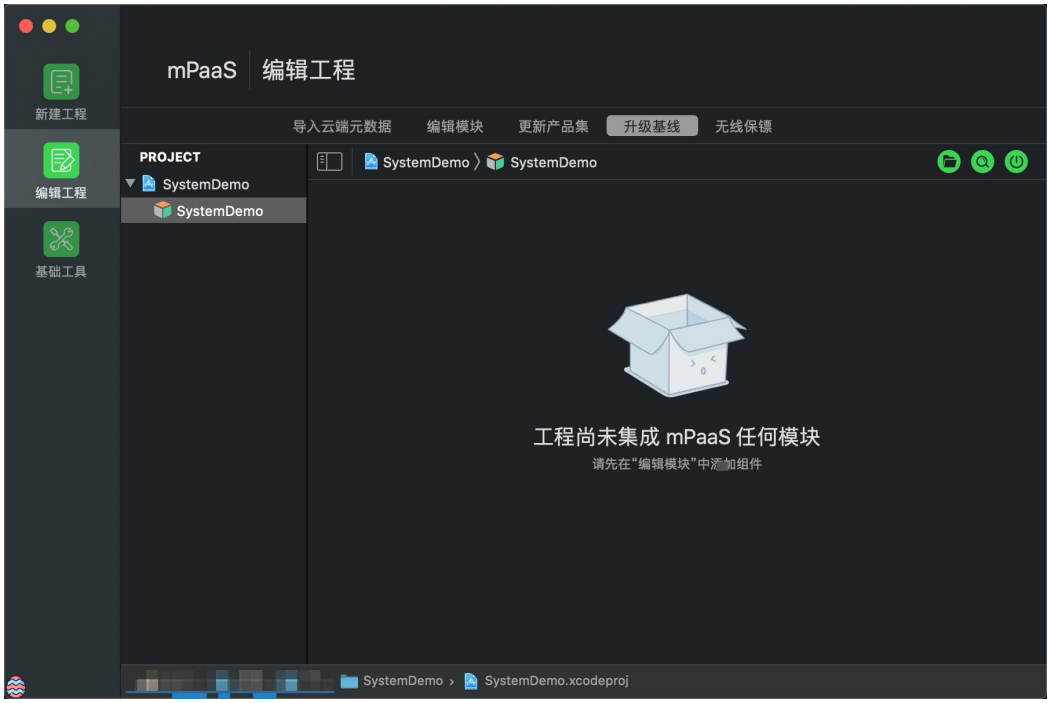
升级基线

单击 升级基线 菜单，单击 确认升级 按钮可进行升级。

在基线选择框内选择基线的类型和版本，下方会出现对应基线的预览框，单击预览中的 查看包含的模块详情 按钮，会弹出模块列表页面，展示当前所选基线的模块列表和模块的安装情况。

单击每个模块右侧的 详情 按钮，可以查看该模块的具体信息，包括描述、Release Note、包含的 framework 文件详情等。

如果当前工程未集成 mPaaS，展示提示页面，建议先在 “编辑模块” 进行集成。



升级基线的相关操作及解释如下：

- 选择基线：

- **定制基线**：目前只开放给专有云用户，在输入框中填入对应的基线 ID 后会自动进行校验。校验失败会提示错误信息。校验正确后即可使用。

- **标准基线**：公开的标准 mPaaS SDK 版本，可以在下拉列表中选择。

- **预览**：展示基线的基线类型和版本，单击“查看包含的模块详情”可以查看对应的模块列表信息。
- **确认升级**：执行升级基线操作。

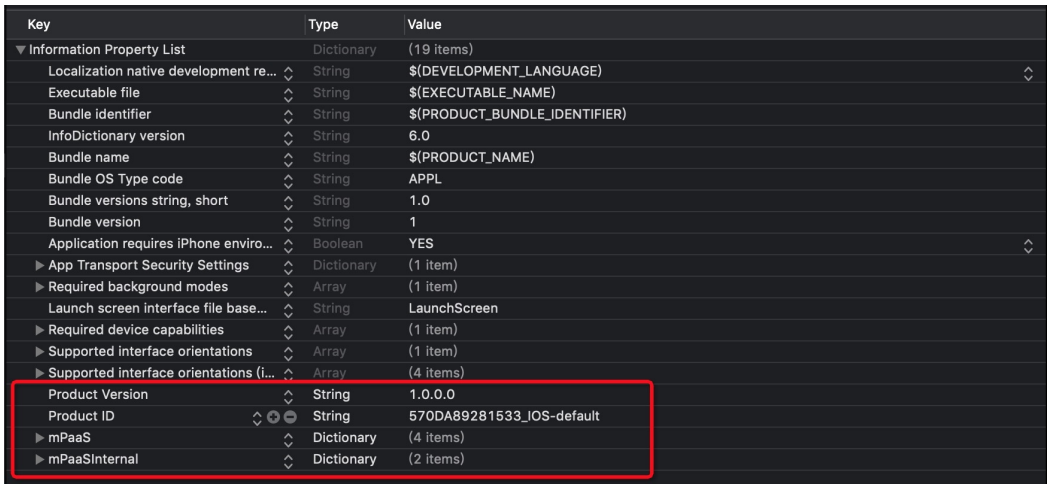
升级基线之后工程中的 mPaaS 信息将会全部更新，当从 低于 10.1.32 的版本升级到 10.1.32 及以上版本时，工程中的 `MPaaS` 目录结构和 `Info.plist` 中的内容会发生一些变化，具体内容如下。

目录结构变化

升级前工程中 MPaaS 目录下存在的组件 category 目录和文件，在升级之后只会保留 APMobileFramework 和 mPaas，其它目录将会移除自动移除，比如 MPHotpatchSDK、APRemoteLogging 等。如果这些目录下有存放自定义的文件，请提前备份保存。详细目录结构参见 mPaaS 目录结构。

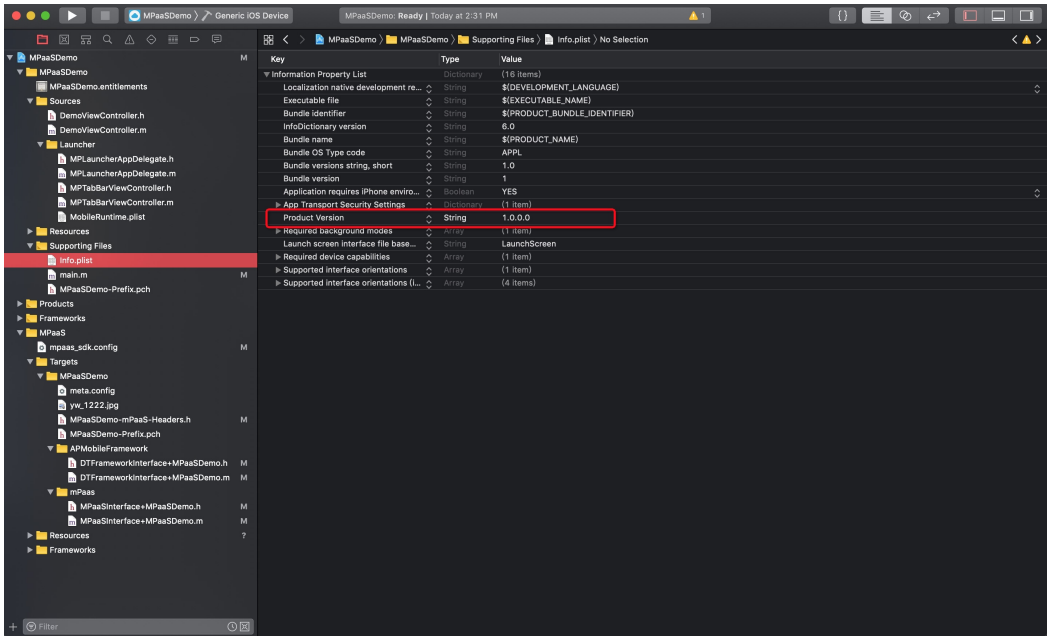
Info.plist 变化

升级前工程中 `Info.plist` 中插入的 mPaaS 相关字段内容如下图所示。



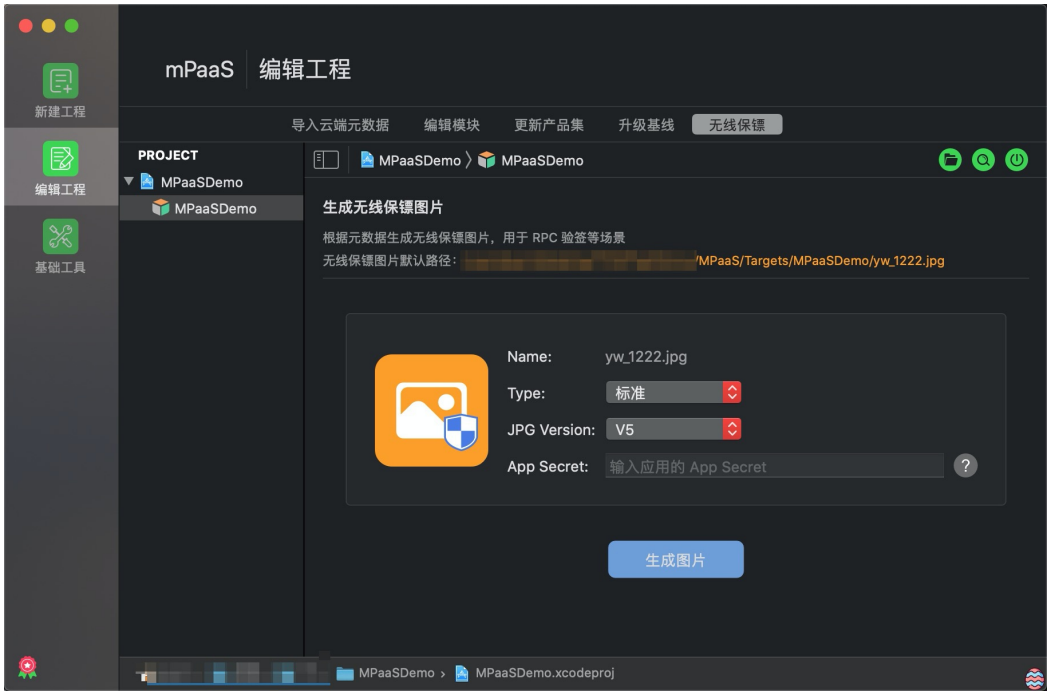
由于 10.1.32 及以上的版本只需保留 Product Version, 不再需要 Product ID、mPaaS、mPaaSInternal 字段，所以升级基线后，插件会自动将这些字段移除。如果发现未成功移除，需手动删除这些内容。升级后的内容如下图所示：

> **说明**
>
> 手动删除时，请勿删除 `Product Version` 。



无线保镖

单击 **无线保镖** 菜单，在页面中输入 AppSecret 等信息，单击 **生成图片** 按钮可生成 RPC 验签和离线包等解密需要的 `yw_1222.jpg` 图片。

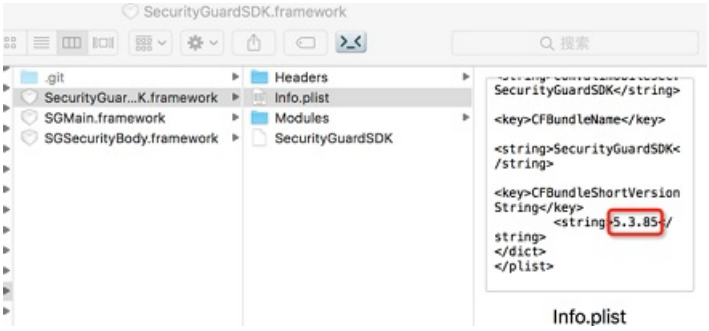


无线保镖的相关操作及解释如下：

- **默认路径**：该路径展示为工程中无线保镖图片的默认存放位置。

- **App Secret**：无线保镖验签的私钥。公有云用户可以在 mPaaS 控制台查询；专有云用户请向服务端开发人员咨询。单击右侧帮助按钮可以查看更多信息。
- **Type**：无线保镖图片 yw_1222.jpg 的类型。
  - 标准：生成普通的无线保镖图片（默认为此类型）。
  - 账户通：生成小程序账户通专用的图片。

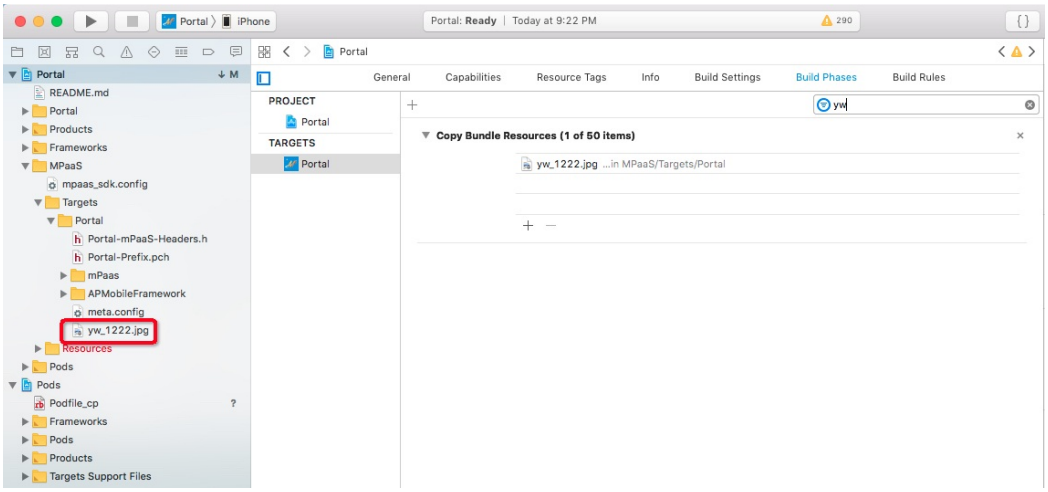
    注意：使用该类型的无线保镖图片后，注意对账户通和网关相关功能进行回归测试）。
- **JPG Version**：无线保镖图片 `yw_1222.jpg` 的版本号。根据客户端 `SecurityGuardSDK.framework` 的版本号区分：
  - 若版本号低于 5.3.85，则 `yw_1222.jpg` 版本号为 V4。
  - 若版本号高于或等于 5.3.85，则 `yw_1222.jpg` 版本号为 V5。



Info.plist

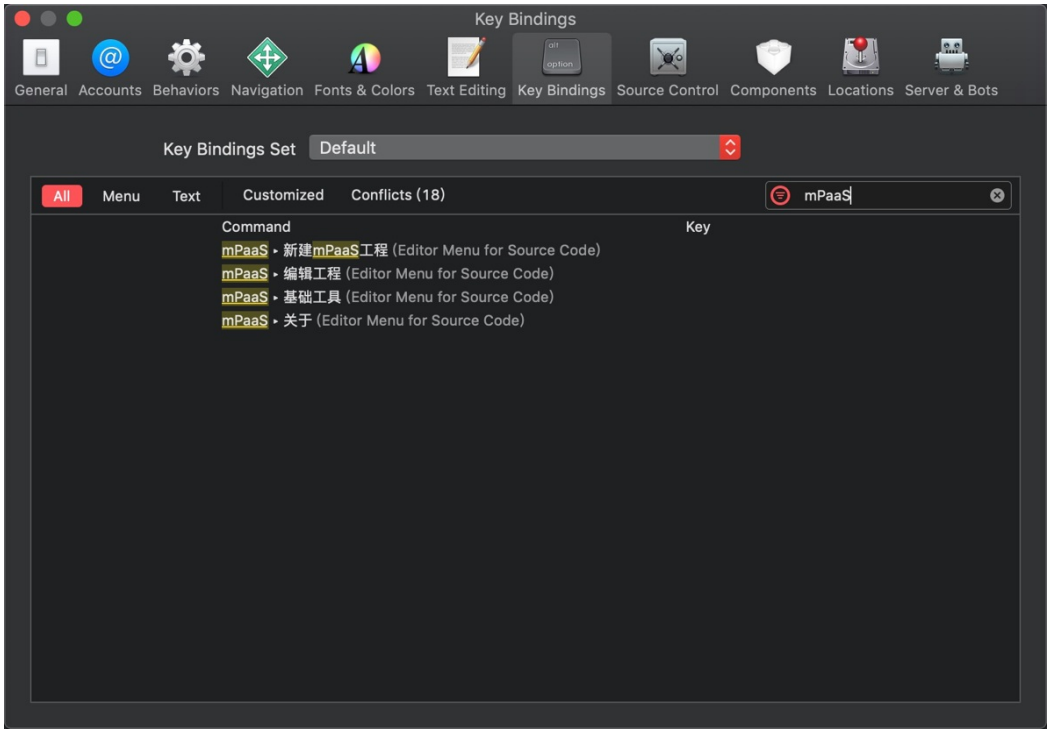
- **生成图片**：单击按钮会提示选择图片保存的路径，自动打开默认路径，选择路径后会执行生成无线保镖图片操作。

> **说明**
>
> - 公有云用户导入的 `meta.config` 文件中已生成了此图片，如无特殊需求，一般不用重新生成。
> - 专有云用户请使用此方法生成 `yw_1222.jpg` 图片，并添加到工程中。



自定义快捷键

mPaaS Xcode Extension 支持用户为 mPaaS 菜单添加自定义快捷键，方便快捷并且个性十足。打开 Xcode 偏好设置，选择 Key Binding，过滤 mPaaS 即可设置，如下图所示。



基础工具

单击 mPaaS > 基础工具，打开基础工具页面，如下图所示。mPaaS 提供了一些基础工具供开发者使用，包括自助服务、热修复工具、和打包构建工具。

自助服务-日志诊断工具

使用日志诊断工具，可以方便自助排查 mPaaS SDK 使用中的问题，输入问题日志，该工具会自动扫描匹配到的问题原因，并提供解决方案。

单击 **日志诊断工具** 图标，打开诊断工具页面。在输入框中输入问题日志，单击 **提交** 按钮可以自动对日志进行分析。

分析到结果之后会弹出诊断报告页面，查看匹配到的问题详情。

日志诊断工具的相关操作和说明

- 日志输入框：输入待扫描的原始日志（目前支持客户端运行时的日志）。
- 重置：提交任务后，如需再次查询，可以单击 重置 在输入框内重新提交日志。
- 查看完整诊断报告：打开诊断报告窗口，展示当前日志的扫描结果。

热修复生成资源包

使用热修复功能时，为了保障安全性，本地测试通过后的 `.js` 脚本文件，需要进行打包加密后才能提交到发布平台。

单击 **生成资源包** 图标，打开生成热修复资源包页面，选择本地测试通过的 `.js` 脚本和 RSA 密钥文件，填入应用的 App Secret，单击 **确定** 按钮可以生成加密后的热修复资源包文件。

生成资源包的相关操作及解释如下：

- **脚本文件**：本地测试验证通过后的 `.js` 脚本。
- **App Secret**：应用对应的 App Secret。单击右侧帮助按钮查看更多信息。
- **私钥文件**：已有的 RSA 私钥文件，通过 openssl 获取，与添加到工程中的公钥文件配对的私钥文件。
- **生成新的 RSA 密钥**：如果您还没有密钥文件或者想使用新的密钥文件，可以选中该项，插件会自动帮你生成新的 RSA 密钥对，保存在输出的目录中。
- **查看产物说明**：单击可以展开具体的说明信息，如图中所示。

热修复提取公钥签名

在热修复脚本的验签和加解密这个过程中，涉及到一对 RSA 非对称密钥和一个 AES 对称加密密钥，每个接入应用需要使用自己的密钥，保证脚本下发的安全性。其中验证 RSA 的密钥过程需要在 `main.m` 中验证公钥自身的签名，以确保公钥文件未被替换。

单击 **提取公钥签名** 图标，打开提取热修复公钥签名页面，选择生成热修复包使用的公钥和私钥文件，单击 **确定** 按钮在页面上可以看到提取出来的签名数组。将打印出来的签名数组拷贝到 `main.m` 方法中即可。

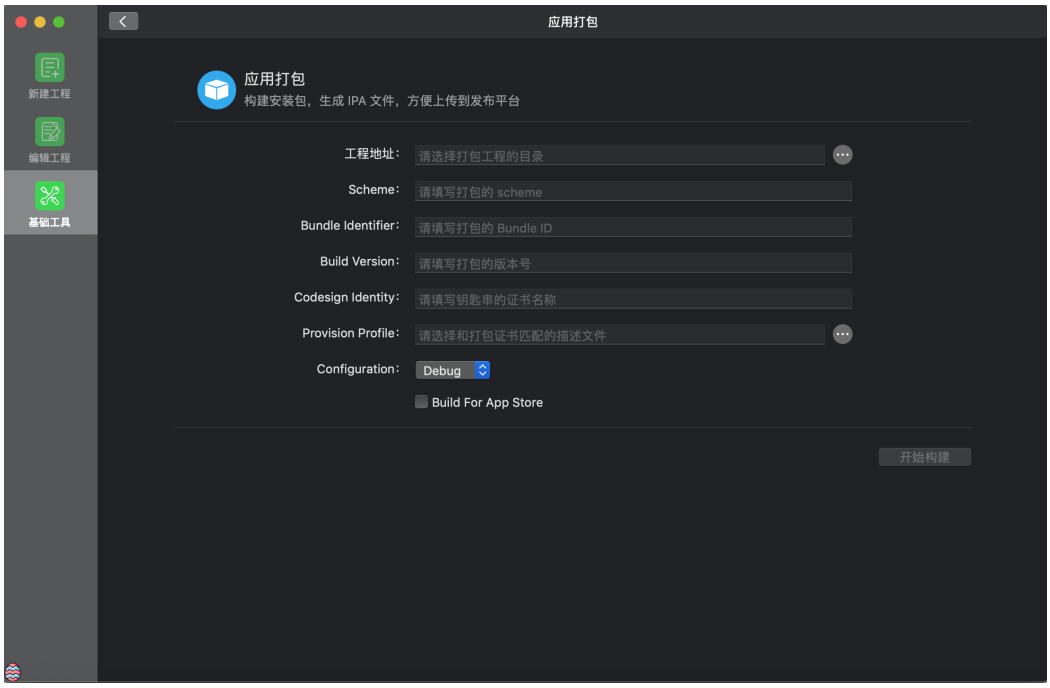
提取公钥签名的相关操作及解释如下：

- **公钥文件**：生成热修复资源包使用的 RSA 公钥文件。
- **私钥文件**：生成热修复资源包使用的 RSA 私钥文件。
- **签名数组**：展示提取出来的公钥签名。

应用打包

mPaaS 插件提供一键打包功能，输入 bundleID 、bundle version、签名参数等信息，可生成一个 `.ipa` 的安装包，使用 mPaaS 发布管理进行 App 的升级，提醒用户安装新版本。

单击 **应用打包** 图标，打开应用打包页面，填写页面全部信息，单击 **开始构建** 按钮开始打包。



应用打包的相关操作及解释如下：

- **工程地址**：选择打包工程的目录地址，即工程文件（`.xcodeproj` 或 `.xcworkspace`）所在的目录。
- **Scheme**：构建的 target 名称。
- **Bundle Identifier**：当前工程的 Bundle Identifier 。如果添加了移动网关服务，必须保证与云端元数据中的 bundleId 字段保持一致，否则移动网关验签失败。
- **Build Version**：当前工程的版本号。一般与工程里 `info.plist` 文件中的 Bundle Version 保持一致。
- **CodeSign Identity**：签名的证书名称。可在 Keychain 中导入给当前工程签名的证书，找到证书的常用名称即可。
- **Provisioning Profile**：与 Bundle Identifier 和签名证书相匹配的 provision 配置文件；
- **Configuration**：打包的配置，Debug 或 Release。
- **Build For App Store**：是否为 App store 安装包，勾选后表示生成的 `.ipa` 为上传 App Store 的包；未勾选表示生成的 `.ipa` 为开发包。
- **开始构建**：单击按钮会提示选择打包产物生成的目录，选择后会开始打包构建操作，打包成功后会自动打开生成的目录文件。

重签名

mPaaS 插件提供对原有 `.ipa` 安装包重签名的功能，可生成一个重签名后的 `.ipa` 安装包，可在更多的设备上安装使用，便于测试验证。

单击 **重签名** 图标，打开重签名页面，填写页面全部信息，单击 **确定** 按钮开始重签名操作。

重签名的相关操作及解释如下：

- IPA 安装包：已有的 `.ipa` 安装包文件。
- 证书名称：新证书的名称，可使用 `security find-identity -p codesigning -v ~/Library/Keychains/login.keychain` 查看。
- 证书 ID：新证书的 ID，可查看 keychain 中相应证书的用户 ID。
- Provision File：更新证书相匹配的 provision 配置文件。
- 确定：单击按钮会提示选择重签名文件的保存目录，选择后会开始重签名操作，重签名成功后会自动打开生成的目录文件。

常见问题

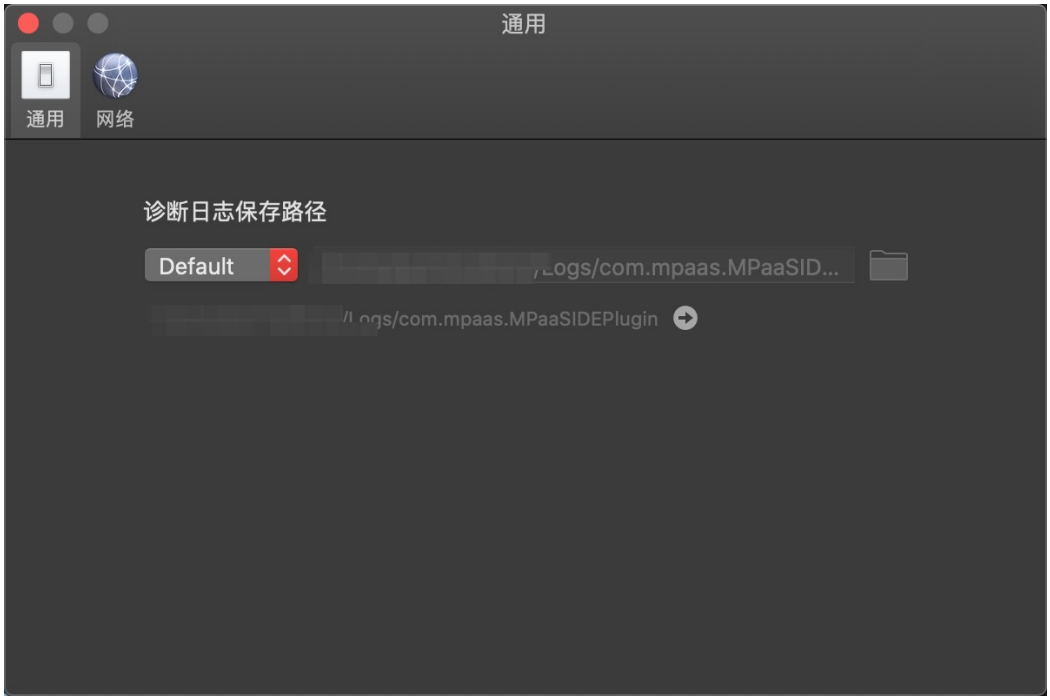
在常见问题页面，单击 **开始浏览**，即可打开详情页面，可以根据查询的具体组件，查看相关的常见问题。

Preferences 设置

打开 mPaaS Xcode Extension 之后，可以单击菜单 **mPaaSPlugin** > **Preferences**，或通过快捷键“⌘ + ,” ，打开设置窗口。

通用

配置诊断日志保存路径：使用诊断功能时，生成日志报告的保存路径，下拉菜单中选择 Custom 选项，可以自定义日志存储目录。



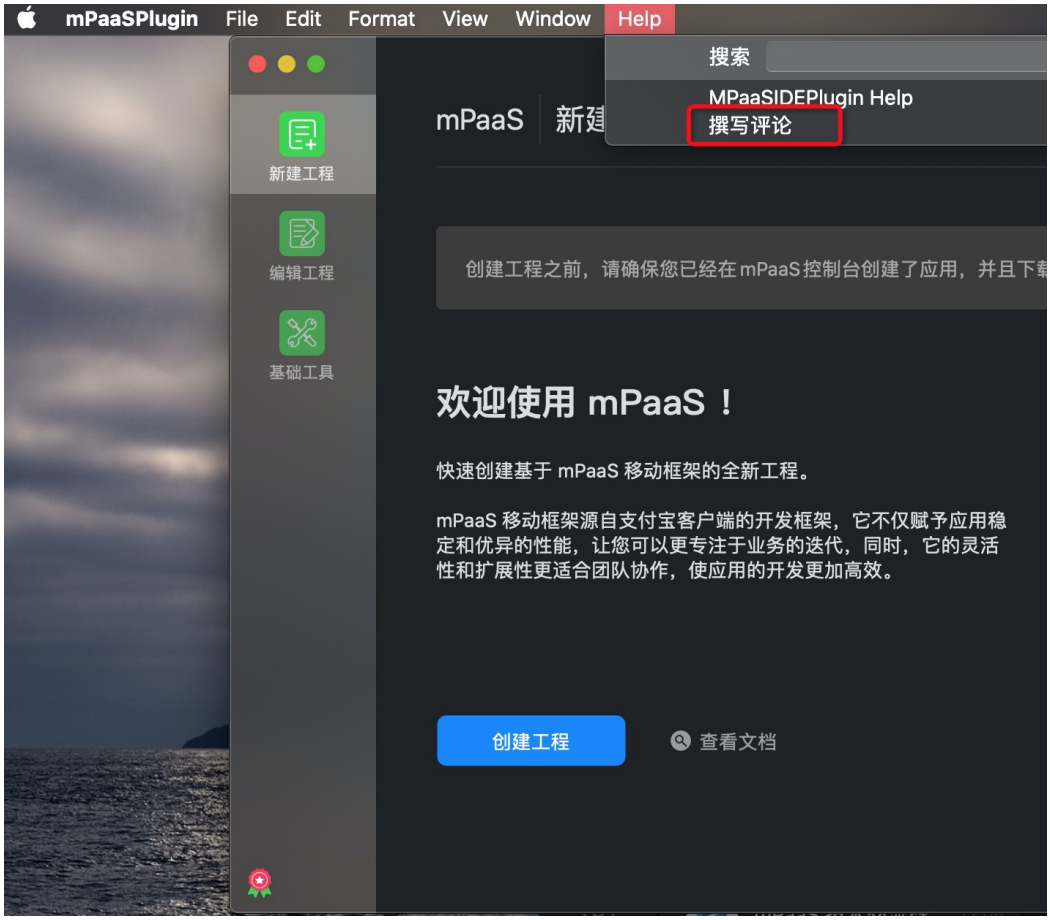
网络

配置 SOCKS 代理：勾选 启用 SOCKS 代理 来启用代理，同时在下方输入框输入代理服务器的地址和端口。

评分及评论

打开 mPaaS Xcode Extension 之后，可以单击首页左下角的 **徽章** 图标打开 **评分及评论** 窗口。

如果是直接在 **应用程序** 中启动的 mPaaS Xcode Extension，还可以通过菜单 **Help > 撰写评论** 打开 **评分及评论** 窗口。



提交评分及评论

在对话框中单击星形进行评分，填写评论标题和内容，单击 **提交** 按钮提交评分及评论内容。

> **说明**
>
> 已经提交的评论和评分可以再次修改、重新提交。

查看全部评论

在对话框中单击 **查看全部评论** 按钮，打开新窗口，展示全部用户的评论数据。

7.2.4. 更新 mPaaS Xcode Extension

mPaaS Xcode Extension 会自安装后的第一次启动起，每两个小时进行一次检查更新。如果有更新版本发布，mPaaS Xcode Extension 会自动提示版本更新。此时，您可以根据提示更新 mPaaS 插件。除此之外，您也可以手动检查更新。

在自动提示时更新插件

在新版本 mPaaS Xcode Extension 插件发布后，mPaaS 插件会自动提示版本更新。

您可以选择 **跳过这个版本**、**稍后提示我** 或 **安装更新**。

- 跳过这个版本：mPaaS Xcode Extension 插件不会再提示更新该版本。
- 稍后提示我：mPaaS Xcode Extension 插件会在两个小时之后再次弹出提示。
- 安装更新：mPaaS Xcode Extension 插件会自动下载更新。下载完成后点击 **安装并重启应用** 来完成新版本的安装。

手动更新插件

单击 **mPaaS** > **检查更新**，mPaaS Xcode Extension 会进行版本检查。

如果当前有更新版本，会提示更新版本已经发布，按照 **在自动提示时更新插件** 操作，即可完成更新。如果没有更新版本，则会提示当前版本已经是最新版。

> **说明**
>
> 如果是独立启动应用的方式打开，可以在 mPaaSPlugin 菜单中选择 Check For Updates 执行更新。

7.2.5. 卸载 mPaaS Xcode Extension

由于 mPaaS Xcode Extension 在 macOS 中的存在形式为独立的应用，因此可以采用在 macOS 中常规的卸载应用方式进行卸载。

> **说明**
>
> 在卸载 mPaaS Xcode Extension 前一定要先退出 Xcode 和 mPaaS Xcode Extension 的所有进程。

7.2.6. mPaaS Xcode Extension 常见问题

查看与 mPaaS Xcode Extension 相关的常见问题列表，点击具体的问题，查看解答：

- 安装 mPaaS Xcode Extension 时，执行安装命令后未打开引导页面
- 安装 mPaaS Xcode Extension 成功，但启动 Xcode 后未看到 mPaaS 菜单项
- 安装 mPaaS Xcode Extension 后第一次确定，在系统授权时误点了“不允许”
- 安装了多个 Xcode，可以同时使用 mPaaS Xcode Extension 吗

安装 mPaaS Xcode Extension 时，执行安装命令后未打开引导页面

- 对于首次安装 mPaaS Xcode Extension 情况：检查终端的安装日志是否安装成功，如果有报错查看对应错误内容，如果显示成功，可以尝试手动打开应用来启动引导页面。
- 对于非首次安装 mPaaS Xcode Extension 情况：如果此次安装的目的是升级插件，那么请使用插件内部的升级功能进行升级；如果想重新安装，请确保执行安装命令之前关闭 Xcode 和 mPaaSPlugin 所有进程，否则会导致安装失败。

安装 mPaaS Xcode Extension 成功，但启动 Xcode 后未看到 mPaaS 菜单项

确保终端展示“安装成功”提示，并且成功打开了引导页面点击“开始使用”，如果未打开引导页，请先手动打开一次，然后按照下述步骤逐项排查。

1. 重启Xcode。
2. 检查 Xcode 是否不再处于去签名状态
3. 检查系统扩展设置，是否启用 mPaaS，查看位置 **系统偏好设置 > 扩展 > Xcode Source Editor**，是否已将 mPaaS 勾选，如果未勾选，手动勾选即可。
4. 如果系统偏好设置中未展示 mPaaS 选项，则判断是 Extension 加载失效，使用如下命令查看：

   ```
   pluginkit -m -p com.apple.dt.Xcode.extension.source-editor
   ```

   如果结果为空，可以尝试使用如下命令进行修复：

   ```
    /System/Library/Frameworks/CoreServices.framework/Frameworks/LaunchServices.framework/Su
   pport/lsregister -f     /Applications/Xcode.app
   ```

   执行之后再次使用第一条命令查看，如果展示结果中包括 mPaaS 插件，表示修复成功，重启 Xcode 即可使用；如果依然无效，则需要重新安装 Xcode 来解决。

安装 mPaaS Xcode Extension 后第一次确定，在系统授权时误点了“不允许”

可以前往系统偏好设置中，手动授权。查看 **系统偏好设置 > 安全性与隐私 > 隐私 > 自动化**，将其中 mPaaSPlugin 下面的 **Xcode** 和 **System** 选项都选中即可。

安装了多个 Xcode，可以同时使用 mPaaS Xcode Extension 吗

由于 Xcode Extension 的加载机制决定只能使用一个 Xcode 路径，一般会使用默认的 Xcode 路径进行加载，即 `/Applications/Xcode.app` 。

7.2.7. 常见错误和错误码说明

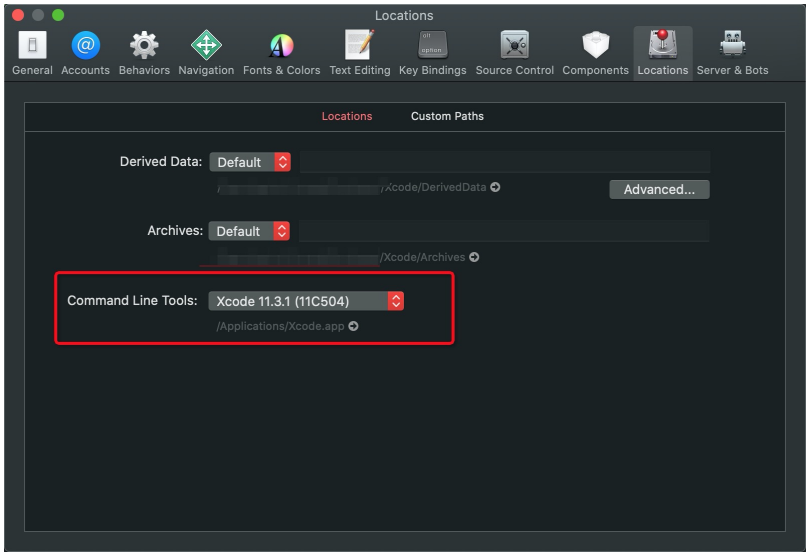
本文将向您介绍使用 mPaaS Xcode Extension 中的常见错误，并说明出现错误的原因和解决办法。同时也提供了错误码说明。

常见错误

[!] 执行 Shell 命令失败: 无法解析工程 Build Settings [Errno: 12] (Mpaas::CommandExecError)

问题原因：Xcode 命令行工具安装或配置不正确。

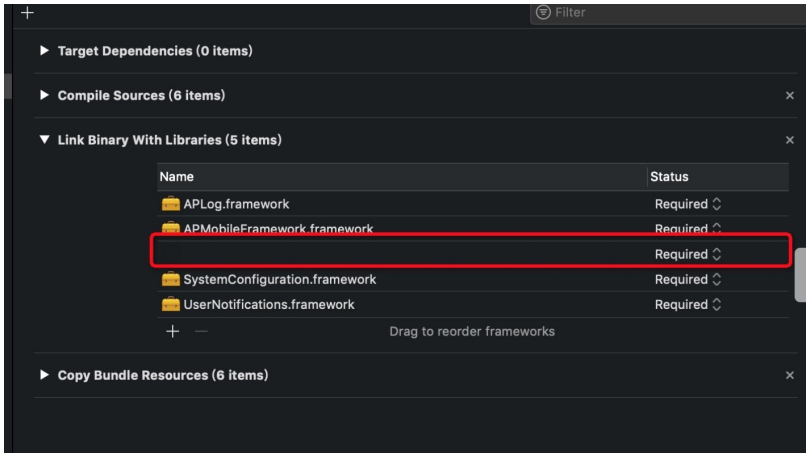
解决方案：重新配置或安装 Xcode 命令行工具，可查看 Xcode 的系统设置，正确的配置如下图所示。



[!] undefined method `display_name'' for nil:NilClass [Errno 1] (NoMethodError)

问题原因：Xcode 工程的 Build Phases 中存在空引用。

解决方案：仔细检查 Build Phases 的每个 Section，移除空引用即可。空引用示例如下图所示。



错误码参考

错误码	描述	错误原因
1	普通错误	需要结合具体的错误内容判断

错误码	描述	错误原因
10	mpaas 运行错误	插件版本异常或安装不正确
12	shell 命令执行失败	需要结合具体的错误内容判断
14	网络异常	网络环境不稳定导致
71	无线保镖图片生成失败	服务异常
72	热修复签名生成失败	服务异常
73	热修复资源包生成失败	服务异常
74	ipa 重签名失败	配置参数错误

7.3. mPaaS 插件（停止维护）

7.3.1. 安装 mPaaS 插件

本文将引导您安装开发者工具，同时也向您介绍安装后加载 mPaaS 插件和重装 mPaaS 插件的相关操作。

mPaaS 插件失效声明

由于苹果系统安全策略的升级，从 macOS 10.15.1 版本开始加入了应用签名的强校验机制，去签名容易导致 Xcode 无法使用，甚至造成系统卡死。经过验证，在 macOS 10.15.3 及以上的系统使用 Xcode 11.3.1 及以上的版本时，安装去签名版本 mPaaS 插件会大概率出现系统卡死，恢复 Xcode 签名之后问题消失。基于以上原因，我们决定逐渐停止更新 iOS mPaaS 插件（去签名版本），并于 2020 年 9 月停止维护。

建议您尽快安装并使用新版本插件 mPaaS Xcode Extension。Extension 目前已经涵盖 mPaaS 插件的全部功能，并且增加了很多新特性，欢迎大家使用。

前置条件

- 安装并打开了 Xcode，并完成了Xcode引导的相关基础工具的安装。
- ruby 版本 ≥ 2.3.0。
- 拥有sudo 权限。

操作步骤

安装 mPaaS 插件

1. 完全退出 Xcode，然后在终端运行以下安装命令：

```
sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/mpaaskit/install.sh)
```

2. 按照屏幕提示输入密码。
3. 去除 Xcode 签名。

> 说明
>
> Xcode 8 及以后版本对 mPaaS 插件的安装有限制，需要去除 Xcode 签名才能顺利安装 mPaaS 插件。步骤如下：
>
> i. 按照屏幕提示，输入 `Y` 同意去除 Xcode 签名。
>
> ii. 选中希望去除签名的 Xcode 按下回车键继续。如果安装了多个 Xcode，请上下移动箭头进行选择。该操作同时会去掉 xcodebuild 的签名。

如果在安装过程中遇到问题，请参考 开发工具常见问题。如果需要恢复 Xcode 签名，或者想了解更多操作命令，请参考 命令行工具。

加载 mPaaS 插件

完成以下步骤以加载 mPaaS 插件：

1. 安装完开发者工具后，重启 Xcode。
2. 在是否加载插件的弹框中，点击 **Load Bundles**：

> 说明
>
> 提示：如果意外点击了 Skip Bundles，您需要执行恢复命令，然后重启 Xcode，再选择 Load Bundles。恢复命令如下，您需要将 8.2.1 替换成实际的 Xcode 版本号： defaults delete com.apple.dt.Xcode DVTPlugInManagerNonApplePlugIns-Xcode-8.2.1

3. 打开任意 Xcode 工程，单击 **Edit > mPaaS > Main**。

> 说明
>
> - 需先打开 Xcode 工程，再打开 **Edit > mPaaS > Main**，否则会因插件读取不到工程信息而报错。
> - 也可通过打开 **File > New > 新建 mPaaS 工程**，查看 mPaaS 插件加载是否成功。

更新 Xcode 后重装 mPaaS 插件

如果您已经安装了开发者工具，那么更新 Xcode 之后，您可以按照如下步骤重装 mPaaS 的 Xcode 插件：

1. 执行命令 `mpaas --version` 查看开发者工具的版本；若执行失败请尝试 `mpaas version` 命令。
2. 若为 `5.0.0` 及以上版本，则执行命令 `mpaas xcode plugins install` 重装 mPaaS 插件；否则请参照上文重新安装开发者工具。

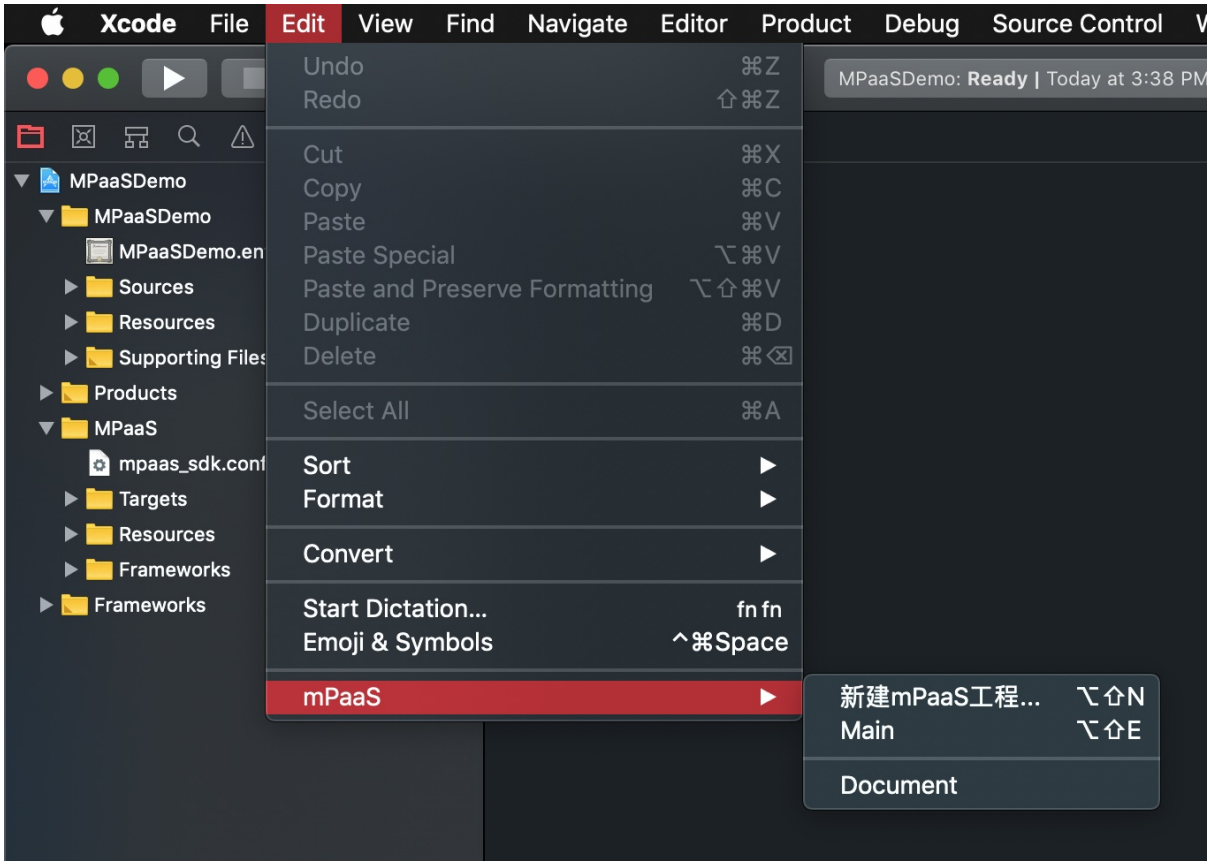
相关链接

- mPaaS 插件：查看 mPaaS 插件的具体使用说明。
- 命令行工具：查看命令行工具的具体使用说明。

7.3.2. 使用 mPaaS 插件

mPaaS 插件 提供图形化界面，帮您快速接入 mPaaS。功能包括：增删和升级 mPaaS 组件、工程打包、重签名等。

点击 **Edit > mPaaS > Main**，您可以进入 mPaaS 插件主界面。如下图所示：



本文将对 **mPaaS 插件** 以下功能进行详细说明：

- 导入云端数据
- 编辑模块
- 产品集更新
- 升级基线
- 生成 HotPatch 资源包
- 打包 mPaaS
- 重签名
- 生成无线保镖图片
- 查看 mPaaS 插件版本

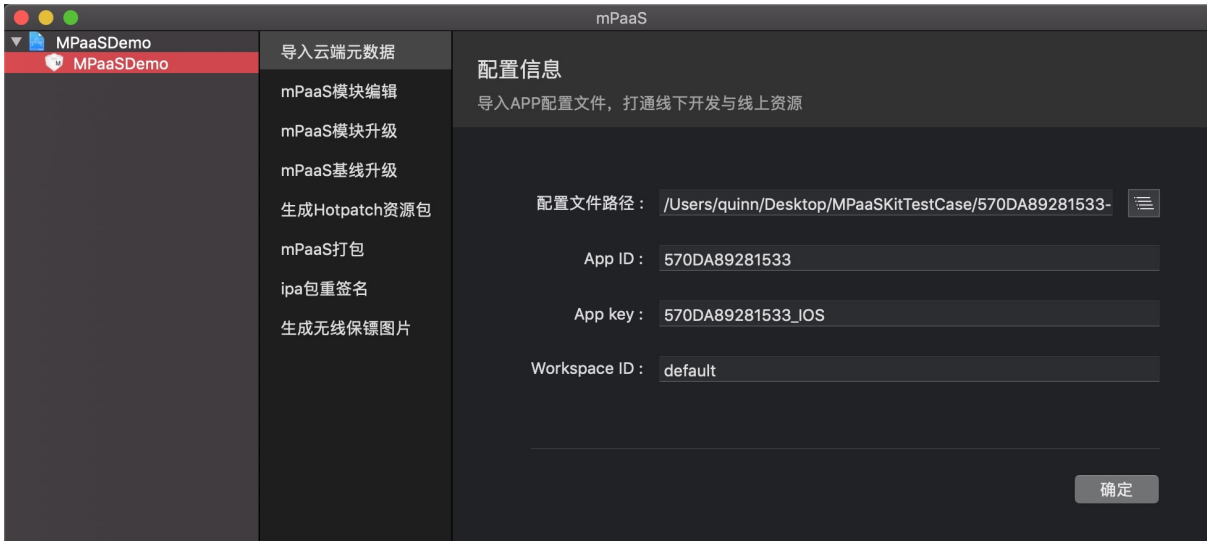
mPaaS 插件失效声明

由于苹果系统安全策略的升级，从 macOS 10.15.1 版本开始加入了应用签名的强校验机制，去签名容易导致 Xcode 无法使用，甚至造成系统卡死。经过验证，在 macOS 10.15.3 及以上的系统使用 Xcode 11.3.1 及以上的版本时，安装去签名版本 mPaaS 插件会大概率出现系统卡死，恢复 Xcode 签名之后问题消失。基于以上原因，我们决定逐渐停止更新 iOS mPaaS 插件（去签名版本），并预计于 2020 年 9 月停止维护。

建议您尽快安装并使用新版本插件 mPaaS Xcode Extension。Extension 目前已经涵盖 mPaaS 插件的全部功能，并且增加了很多新特性，欢迎大家使用。

导入云端数据

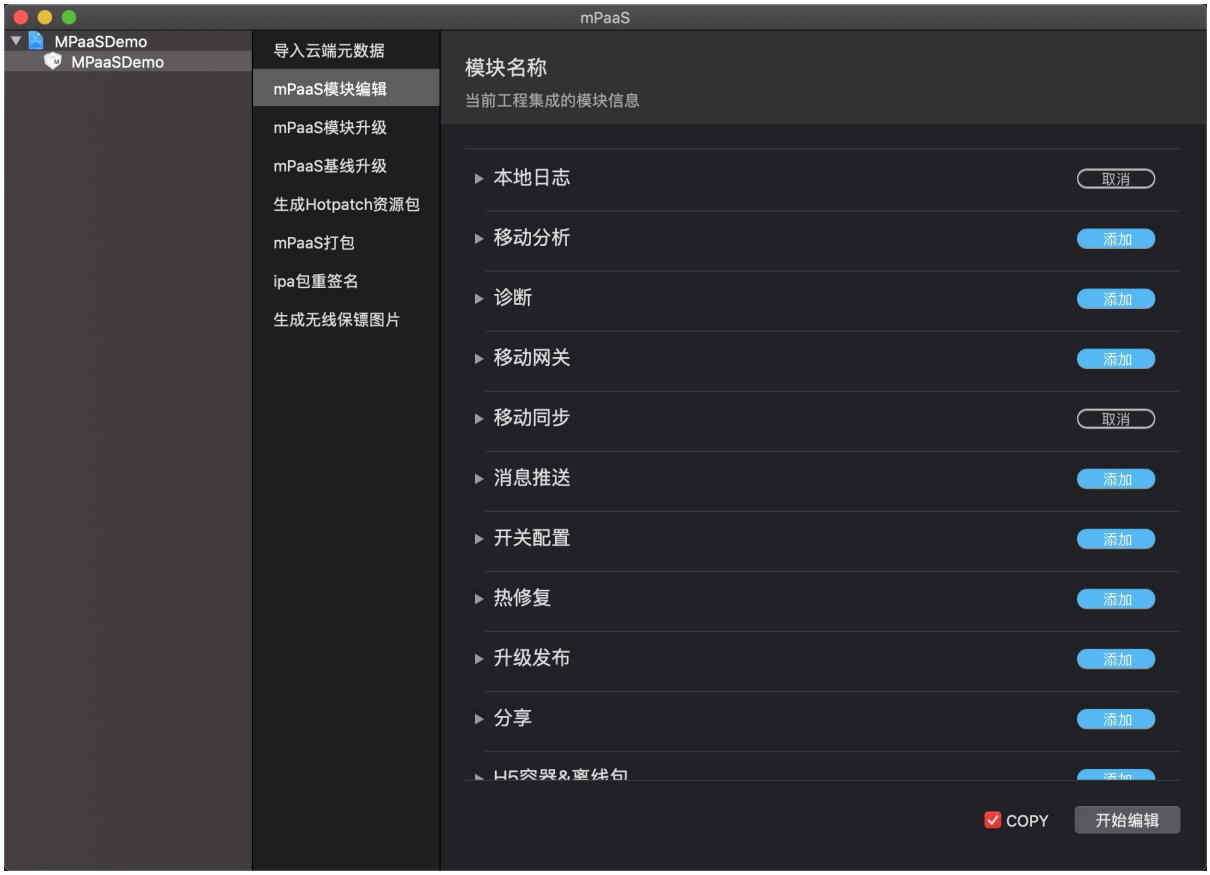
点击 导入云端元数据 选项，选择下载的云端数据配置文件，点击 确定 按钮，将对应的配置导入到当前工程中。



> 说明　在 mPaaS 控制台 > 代码管理 > 代码配置 页面下载云端元数据。更多信息，请参见 在控制台创建应用 > 下载配置文件。

编辑模块

点击 mPaaS 模块编辑 选项，选择需要添加的模块，点击 开始编辑 按钮，将对应的模块添加到当前工程中。



- **取消**：表示此模块已经添加，可以点击按钮从当前工程中删除。

  > **说明** 如果是使用基于 mPaaS 框架创建的工程，在编辑模块时，请勿取消移动客户端框架模块。

- **添加**：表示此模块还未添加，可以点击按钮，添加此模块到当前工程。
- **COPY**：工程依赖的 SDK 会统一安装到用户机器的公共目录，并由插件自动维护。
  - **不勾选**：会使用公共目录下的 `framework` 集合。这样可以减小工程的大小，用户只需要提交自己的代码。
  - **勾选**：相应模块的 `framework` 会拷贝到当前工程的 `MPaaS` 目录下，您可以将这些模块包一起提交。

添加的模块会同步到工程的 `mPaaS` 目录下，具体结构说明详见 mPaaS 目录结构。

产品集更新

点击 **mPaaS 产品集更新** 选项，可查看产品集信息和状态。点击右下方的 **开始更新** 后，产品集会升级到最新版本，包括所有添加的模块。

- **当前产品集版本**：表示当前工程集成的 mPaaS 产品集版本号。
- **最新产品集版本**：表示 mPaaS 最新发布的产品集版本号。
- **查看最新产品集 Release Note**：查看 mPaaS SDK 的 release note。
- **模块名称**：表示最新产品集中包含的模块名。
- **状态**：表示此模块在当前工程中的状态：未安装、已安装最新版本、可升级到最新版本。
- **开始更新**：自动将产品集中所有可升级的模块进行升级。

升级成功后，当前工程目录下的 `mpaas_sdk.config` 文件也会同步更新产品集及各模块的版本信息。

升级基线

点击 **mPaaS 基线升级** 选项，在右侧下拉框中选择要升级到的基线版本（即 SDK 版本），点击 **确认升级** 按钮可进行升级。

- **当前基线版本**：表示当前 target 已经集成的 mPaaS 基线版本号。
- **要升级的基线版本**：表示要升级到的基线版本号。
- 模块列表：要升级到的基线中包含的所有模块。包括以下信息：
  - **模块名称**：模块的名称。
  - **描述**：模块的描述信息，右侧 info 按钮可以查看具体的 release note。
  - **状态**：表示该模块是否已经集成安装到工程中。

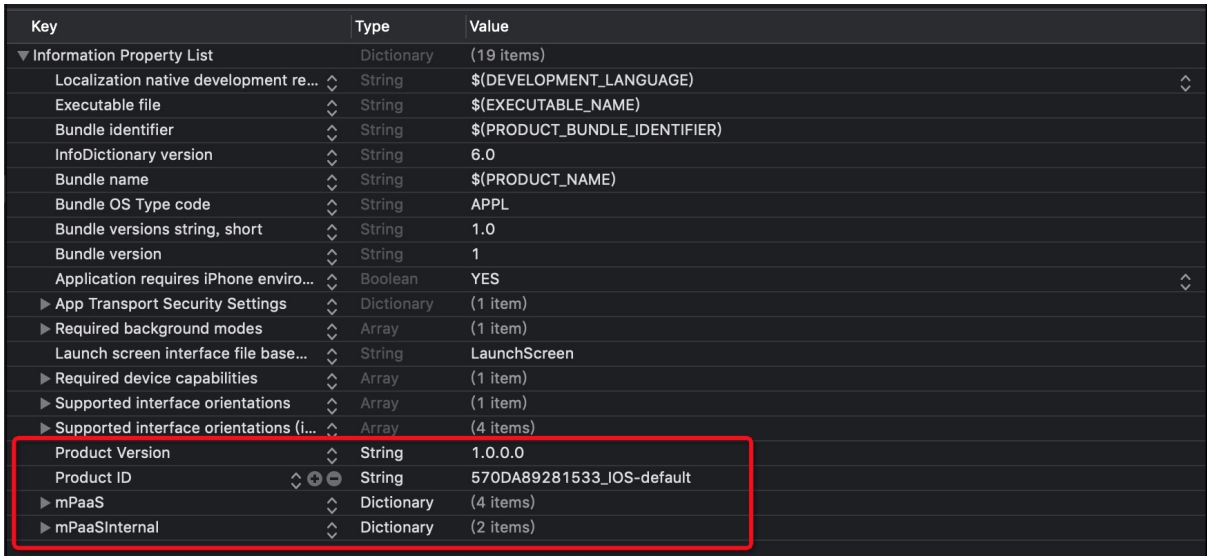
升级基线之后工程中的 mPaaS 信息将会全部更新，当从 低于 10.1.32 的版本升级到 10.1.32 及以上版本时，工程中的 `MPaaS` 目录结构和 `Info.plist` 中的内容会发生一些变化，具体内容如下。

目录结构变化

升级前工程中 `MPaaS` 目录下存在的组件 category 目录和文件，在升级之后只会保留 `APMobileFramework` 和 `mPaas`，其它目录将会移除自动移除，比如 `MPHotpatchSDK`、`APRemoteLogging` 等。如果这些目录下有存放自定义的文件，请提前备份保存。详细目录结构参见文档 mPaaS 目录结构。

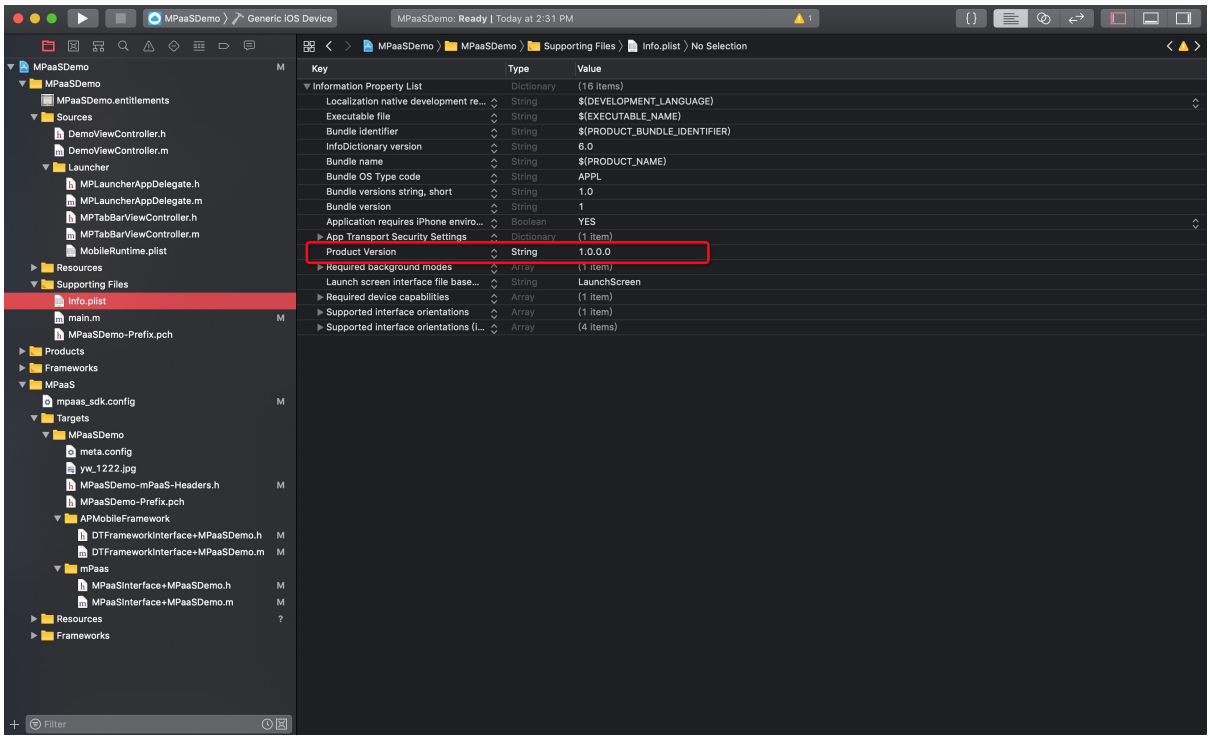
Info.plist 变化

升级前工程中 `Info.plist` 中插入的 mPaaS 相关字段内容如下图所示。



由于 10.1.32 及以上的版本只需保留 `Product Version`，不再需要 `Product ID`、`mPaaS`、`mPaaSInternal` 字段，所以升级基线后，插件会自动将这些字段移除，如果发现未成功移除，需手动删除这些内容。升级后的内容入下图所示：

> **说明** 手动删除时，请勿删除 `Product Version`。

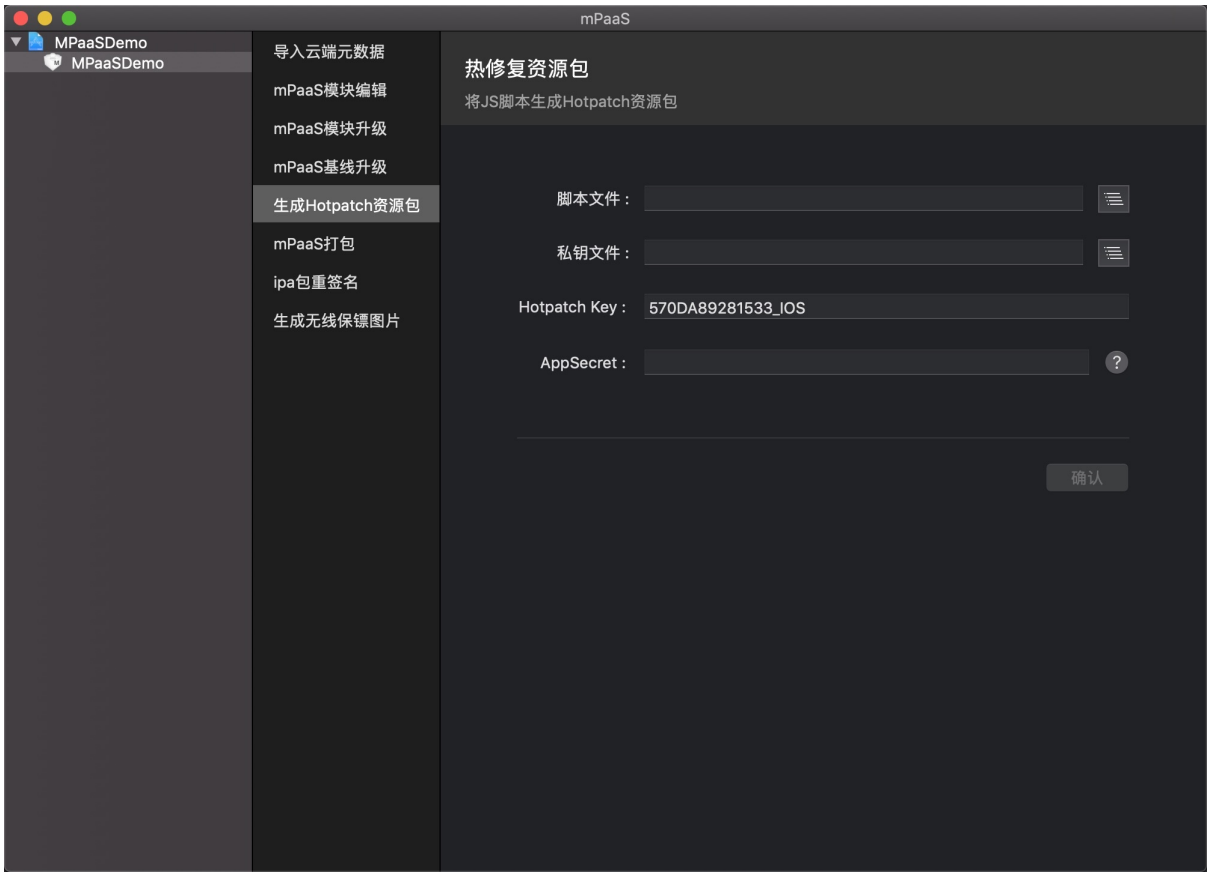


生成 HotPatch 资源包

使用热修复功能时，为了保障安全性，本地测试通过后的 `.js` 脚本文件，需要进行打包加密后才能提交到发布平台。

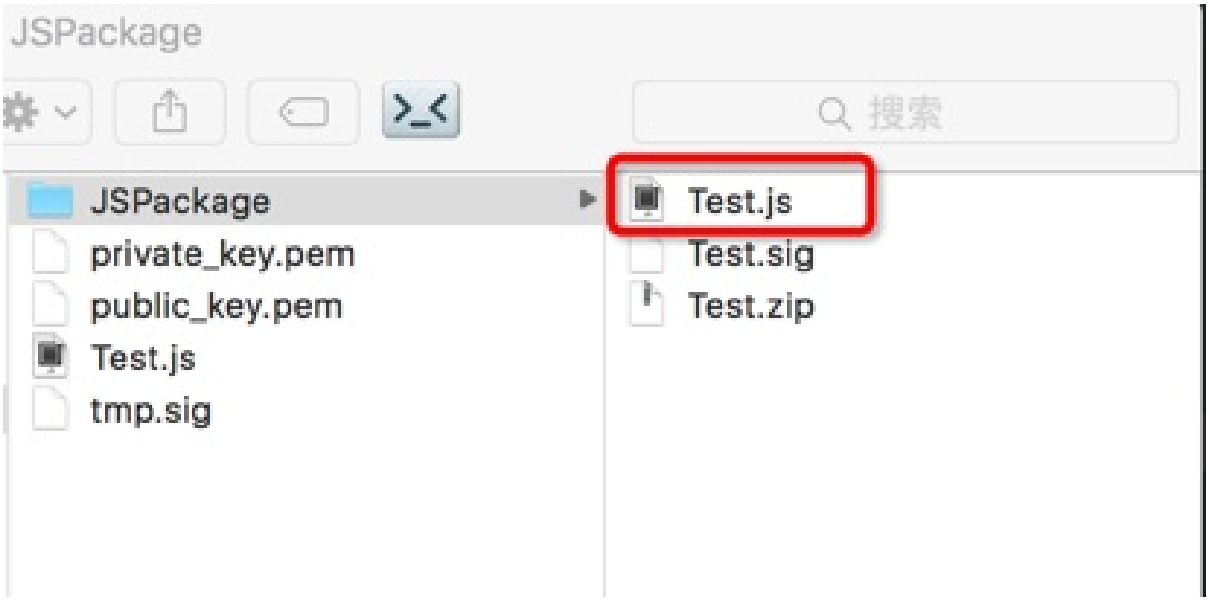
点击 **生成 Hotpatch 资源包** 选项，上传本地测试通过的 `.js` 脚本和 RSA 私钥文件，得到加密后的 Hotpatch 资源包文件。传入参数的含义如下：

- **脚本文件**：本地测试验证通过后的 `.js` 脚本。
- **私钥文件**：通过 `openssl` 获取，与添加到工程中的公钥文件配对的私钥文件。
- **Hotpatch Key**：appKey。该 key 会从云端配置文件中读取，您不用修改。
- **AppSecret**：应用对应的 App Secret。点击右侧帮助按钮查看更多信息。



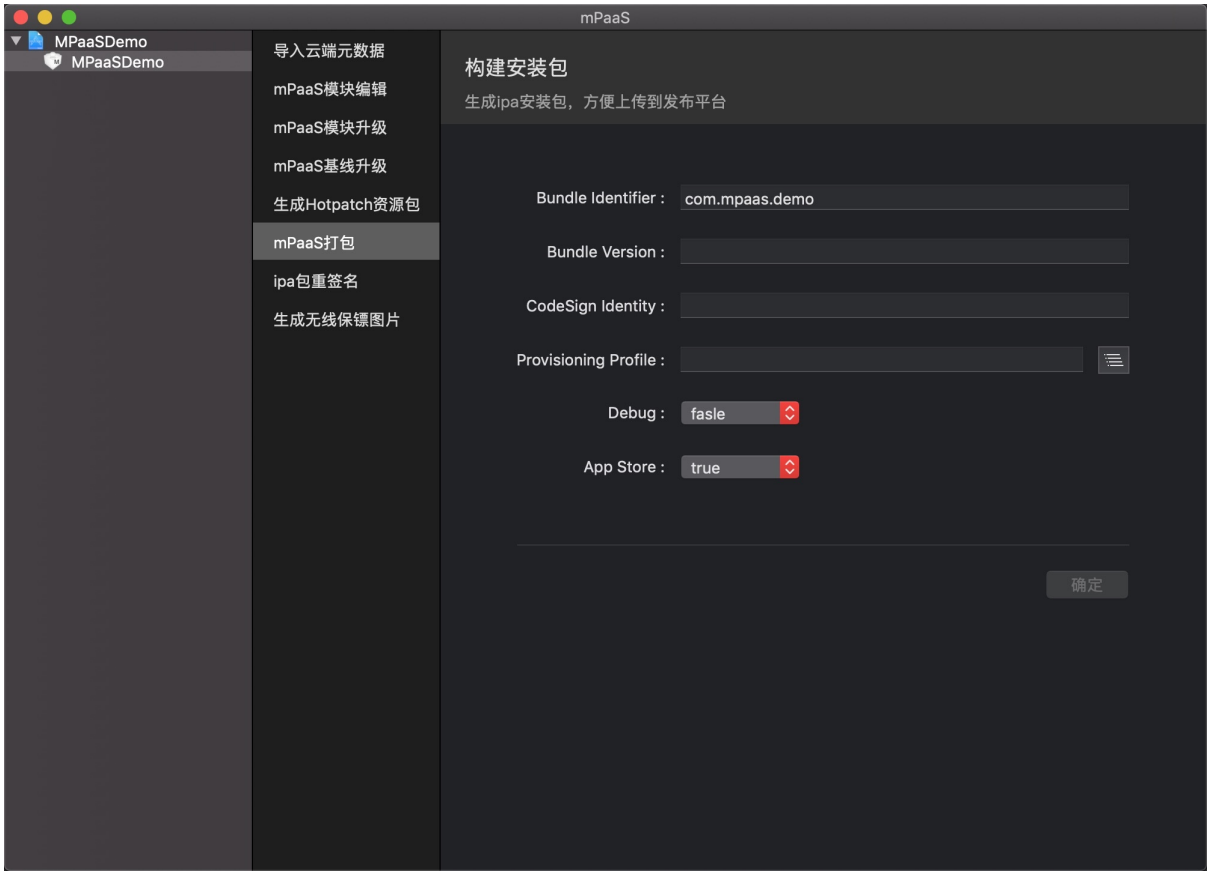
生成的热修复资源包，与上传的 `.js` 脚本文件保存在同级目录下，包含的内容如下：

- `Test.js`：最后上传到发布平台的加密文件。
- `Test.sig`：加密后的签名文件。
- `Test.zip`：加密后的脚本文件。通常在上传到发布平台之前，用来验证加密算法是否正确。



打包 mPaaS

mPaaS 插件提供一键打包功能，输入 `bundleID` 、`bundle version` 、签名参数等信息，可生成一个 `.ipa` 的安装包，使用 mPaaS 发布管理 进行 App 的升级，提醒用户安装新版本。



- **Bundle Identifier**：当前工程的 Bundle Identifier 。如果添加了移动网关服务，必须保证与云端元数据中的 bundleId 字段保持一致，否则移动网关验签失败。
- **Bundle Version**：当前工程的版本号。一般与工程里 `info.plist` 文件中的 Bundle Version 保持一致。

- **CodeSign Identity**：签名的证书名称。可在 Keychain 中导入给当前工程签名的证书，找到证书的常用名称即可。
- **Provisioning Profile**：与 Bundle Identifier 和签名证书相匹配的 provision 配置文件。
- **Debug**：是否使用 debug 配置信息。`true` 表示使用 debug 配置信息，`false` 表示使用 release 配置信息。
- **App Store**：是否为 App store 安装包。`true` 表示生成 App Store 安装包，需使用发布证书打包；`false` 表示生成非 App Store 安装包，可用开发证书打包。

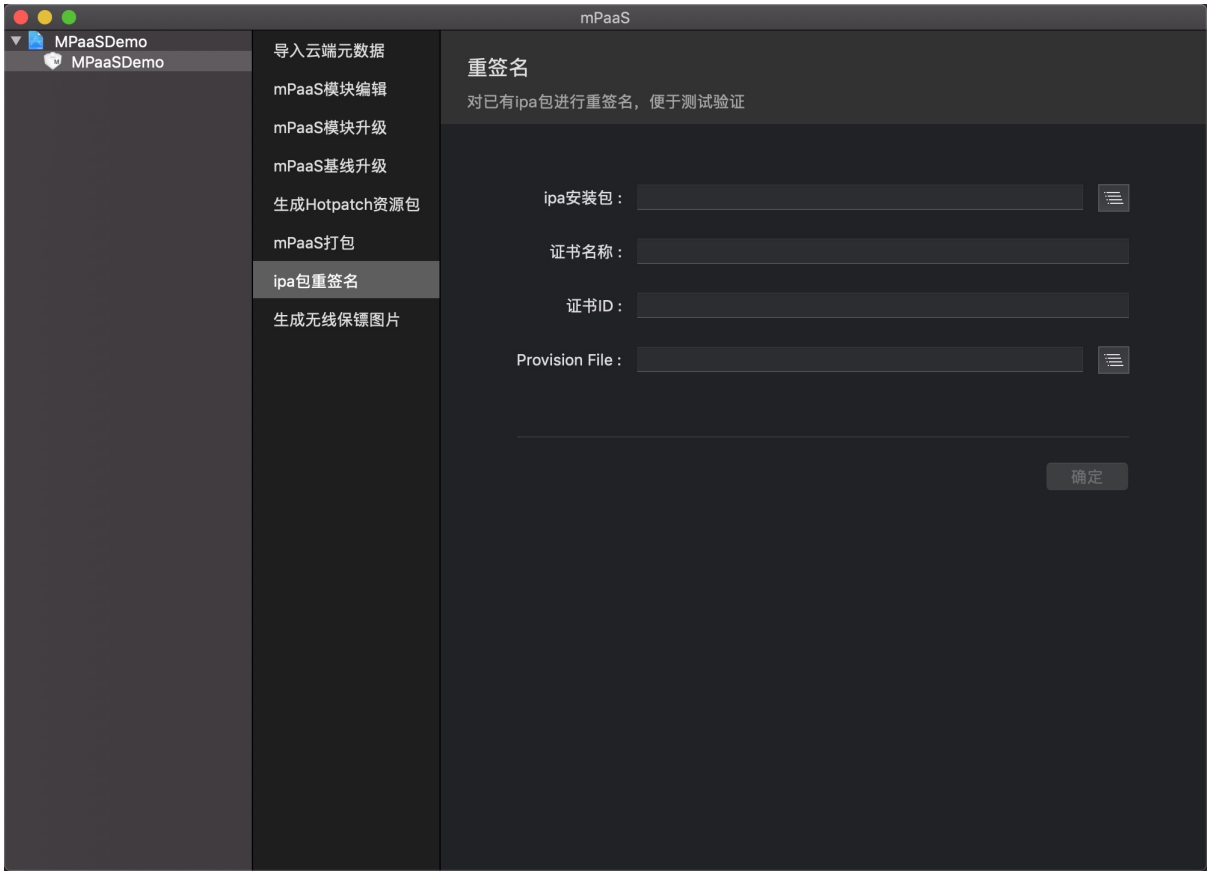
生成的 `.ipa` 包会保存在当前工程的 `Product` 目录下。

重签名

mPaaS 插件提供对原有 `.ipa` 安装包重签名的功能，可生成一个重签名后的 `.ipa` 安装包，可在更多的设备上安装使用，便于测试验证。

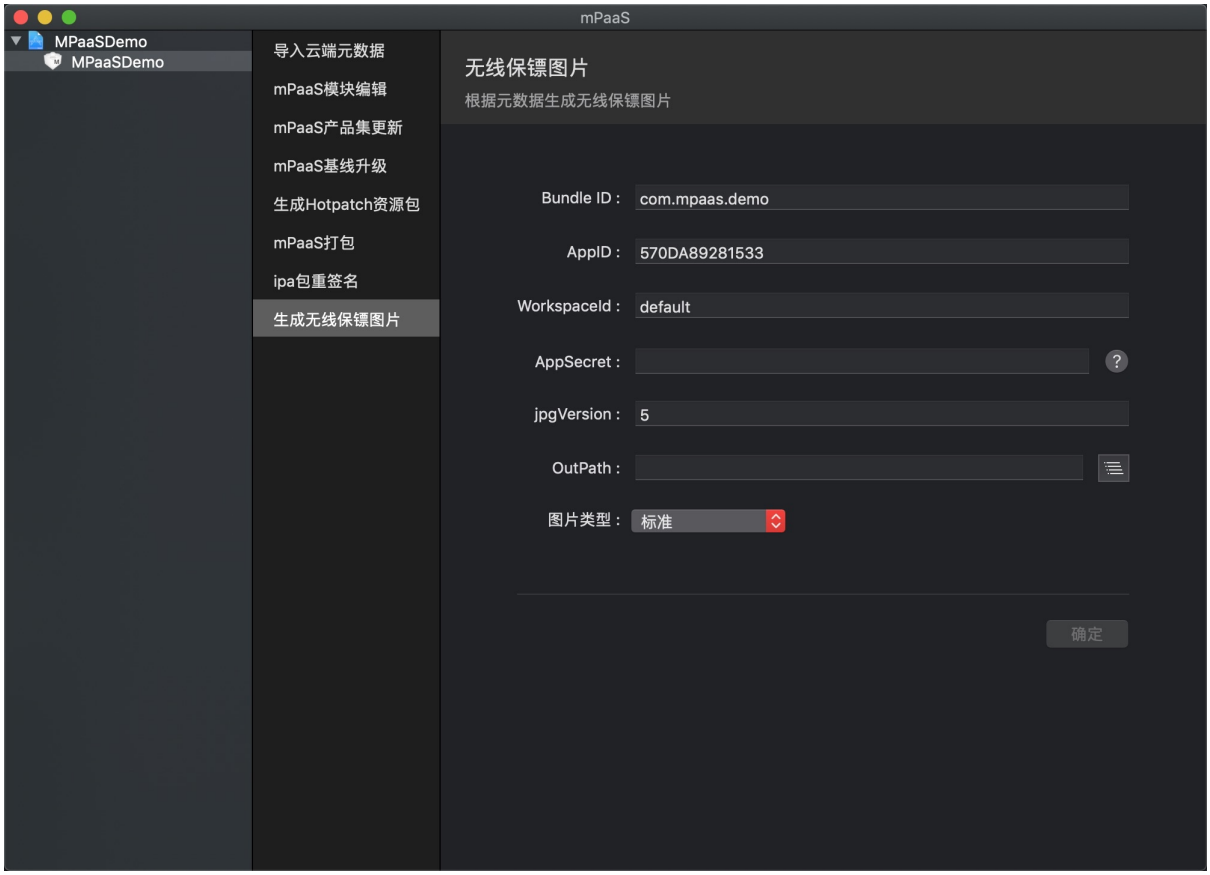
- **ipa 安装包**：已有的 ipa 安装包。
- **证书名称**：新证书的名称，可使用 `security find-identity -p codesigning -v ~/Library/Keychains/login.keychain` 查看。
- **证书ID**：新证书的 ID，可查看 keychain 中相应证书的 `用户 ID`。
- **Provision File**：跟新证书相匹配的 provision 配置文件。

点击确定后，选择重签名后的 `.ipa` 包的保存路径，新的安装包可以直接在真机上安装使用。

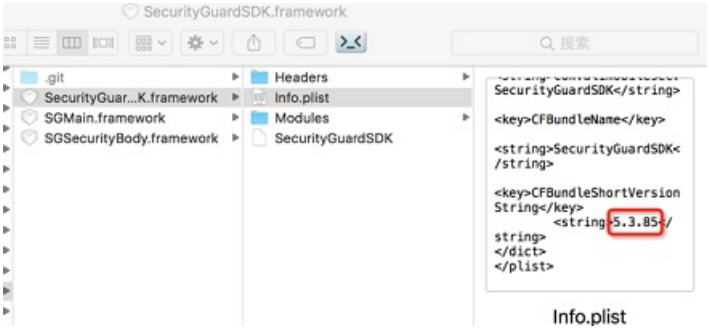


生成无线保镖图片

mPaaS 插件提供离线生成无线保镖安全图片的功能，输入 `bundleID`、`AppSecret` 等信息，可生成 RPC 验签和离线包等解密需要的 `yw_1222.jpg` 图片。

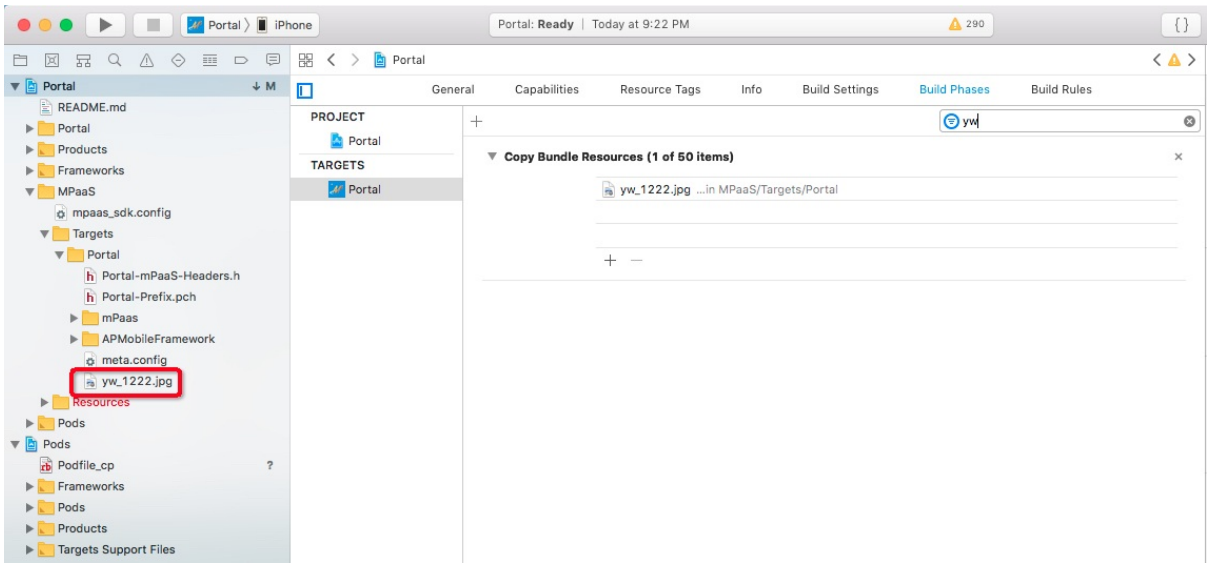


- **Bundle Identifier**：当前工程的 Bundle Identifier。
- **AppID**：当前工程的 AppID。与工程中 `meta.config` 文件中的 `appId` 字段保持一致。
- **WorkspaceID**：当前工程的 WorkspaceID。与工程中 `meta.config` 文件中的 `workspaceId` 字段保持一致。
- **AppSecret**：无线保镖验签的私钥。公有云用户可以在 mPaaS 控制台查询；专有云用户请向服务端开发人员咨询。点击右侧帮助按钮可以查看更多信息。
- **jpgVersion**：无线保镖图片 `yw_1222.jpg` 的版本号。根据客户端 `SecurityGuardSDK.framework` 的版本号区分：
  - 若版本号低于 5.3.85，则 `yw_1222.jpg` 版本号为 4。
  - 若版本号高于或等于 5.3.85，则 `yw_1222.jpg` 版本号为 5。



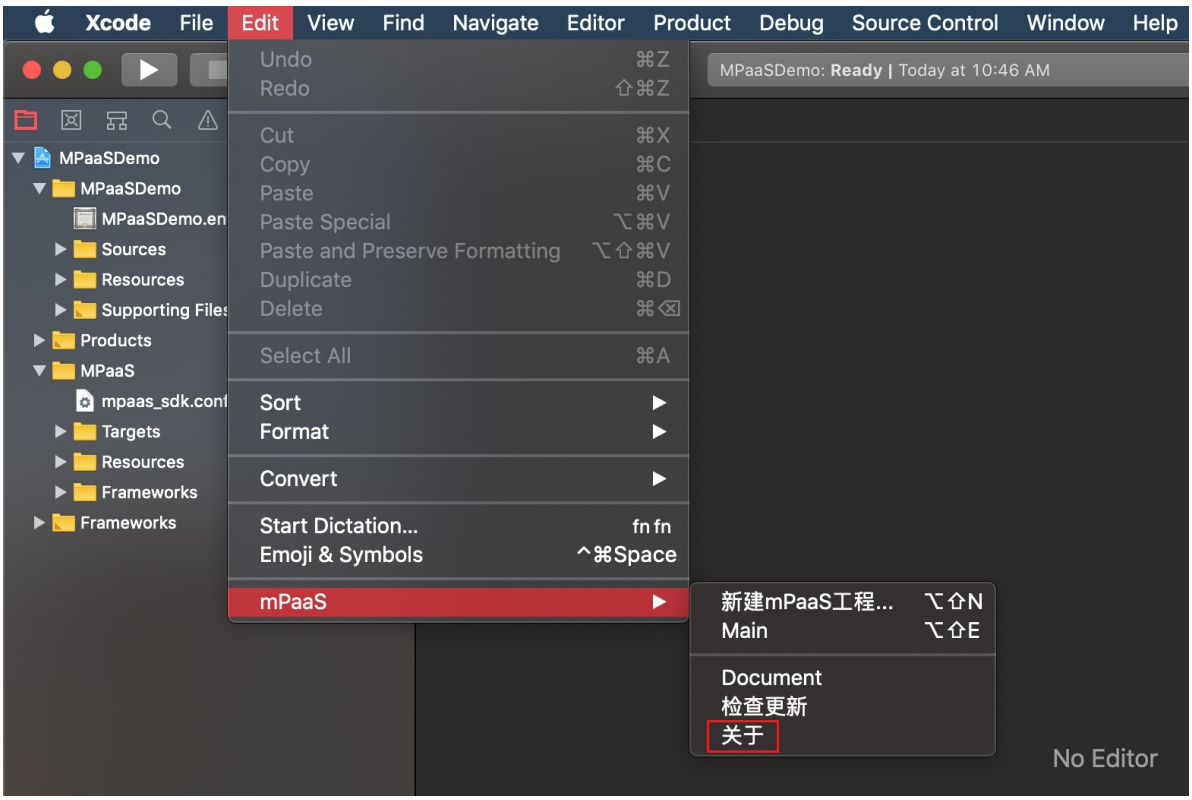
- **OutPath**：`yw_1222.jpg` 图片的输出路径。
- **图片类型**：类型可以选择 **标准** 和 **账户通**，默认为 **标准**。当选择 **账户通** 类型时，会生成小程序账户通专用的图片。

> **说明**
>
> - 公有云用户导入的 `meta.config` 文件中已生成了此图片，如无特殊需求，一般不用重新生成。
> - 专有云用户请使用此方法生成 `yw_1222.jpg` 图片，并添加到工程中。



查看 mPaaS 插件版本

- mPaaS 插件在 5.0.5 版本中提供了检查更新的功能，您可以在 mPaaS 插件中点击 **Edit > mPaaS > 关于**，查看当前正在使用的 mPaaS 插件的版本。



版本信息如下：

- 如果在您的 mPaaS 插件菜单中没有 **关于** 菜单，则说明您当前使用 mPaaS 插件版本小于 5.0.5。因此您需要在终端执行以下命令以获知准确的版本号。

```
mpaas --version
```

7.3.3. 更新 mPaaS 插件

mPaaS 插件在 5.0.5 版本中，增加了检查更新的功能。因此，在 5.0.5 之前和之后的版本需要采取不同的升级方式。

您可以在 mPaaS 插件中点击 **Edit** > **mPaaS** > **关于** 中查看当前正在使用的 mPaaS 插件的版本号，以此选择不同的升级方式。

插件版本 < 5.0.0

对于 5.0.0 之前的 mPaaS 插件，您需要重新安装 mPaaS 插件以获得最新版本。请您参考 安装开发者工具 进行安装。默认会安装最新版本。

插件版本 5.0.0 ~ 5.0.4

对于版本在 5.0.0 和 5.0.4 之前的mPaaS插件，您需要在终端执行以下命令以升级 mPaaS 插件，默认会升级到最新版本。

```
mpaas update
```

插件版本 ≥ 5.0.5

对于 5.0.5 版本的 mPaaS 插件，mPaaS 插件会自安装后的第一次启动起，每两个小时进行一次检查更新。如果有更新版本发布，mPaaS插件会自动提示版本更新。此时，您可以根据提示更新 mPaaS 插件，详情参见 在自动提示时更新插件。除此之外，您也可以手动检查更新，详情参见 手动更新插件。

在自动提示时更新插件

在新版本 mPaaS 插件发布后，mPaaS 插件会自动提示版本更新。

您可以选择 **跳过这个版本**、**稍后提示我** 或 **安装更新**。

- **跳过这个版本**：mPaaS 插件不会再提示更新该版本。
- **稍后提示我**：mPaaS 插件会在两个小时之后再次弹出提示。
- **安装更新**：mPaaS 插件会自动下载更新。下载完成后点击 **安装并重启应用** 来完成新版本的安装。

说明： 如果在更新 mPaaS 插件时命令行工具也需要更新，那么在安装更新并重启应用后，系统会要求输入管理员密码来完成后续安装，该过程需要 30 秒左右。

手动更新插件

在 mPaaS 插件中点击 **Edit > mPaaS > 检查更新**，mPaaS 插件会进行版本检查。

- 如果当前有更新版本，会提示更新版本已经发布，按照 在自动提示时更新插件 操作，即可完成更新。
- 如果没有更新版本，则会提示当前版本已经是最新版。

7.3.4. 卸载 mPaaS 插件

具体的卸载命令因不同 mPaaS 插件版本不同而有所区别，如何查看 mPaaS 插件版本请参考 查看 mPaaS 插件版本。

插件版本 ≥ 5.0.0

如果您之前安装了5.0 及以上版本的mPaaS 插件，可以使用如下的方法进行卸载。

1. 退出 Xcode。
2. 恢复 Xcode 签名，在终端中执行如下命令。

```
mpaas xcode restore
```

3. 卸载 Xcode 插件，在终端中执行如下命令。

```
mpaas xcode plugins uninstall
```

4. 卸载 mPaaSKit，在终端中执行如下命令。

```
rm -rf /Users/Shared/.mpaaskit
```

5. 卸载 mpaas ，在终端中执行如下命令。

```
sudo rm /usr/local/bin/mpaas
```

插件版本 < 5.0.0

如果您之前安装了 mPaaS 插件 5.0 以下的版本，可以使用如下的方法进行卸载。

1. 退出 Xcode。
2. 恢复 Xcode 签名，在终端中执行如下命令。

```
mpaas restore
```

3. 卸载 Xcode 插件，在终端中执行如下命令。

```
rm -rf ~/Library/Application\ Support/Developer/Shared/Xcode/Plug-ins/mPaaSPlugin.xcplugi
n
```

4. 卸载 mPaaSKit，在终端中执行如下命令。

```
rm -rf /Users/Shared/.mpaaskit
```

5. 卸载 mpaas 命令，在终端中执行如下命令。

```
sudo rm /usr/local/bin/mpaas
```

7.3.5. mPaaS 插件常见问题

查看与开发工具（Xcode、mPaaS Extension）相关的常见问题列表，点击具体的问题，查看解答：

- 如何恢复 Xcode 签名
- mPaaS 插件安装完毕之后，插件中选择文件时速度很慢
- mPaaS 插件安装完毕之后，启动 Xcode 的时候发生闪退

如何恢复 Xcode 签名

去除签名本身对使用 Xcode 没有任何影响，您可以使用以下方法之一恢复签名：

- 在系统的命令行终端运行以下命令：

```
mpaas xcode restore
```

- 安装 `reuse_xcode_plugins` 后恢复签名。

```
gem install reuse_xcode_plugins
reuse_xcode_plugins --restore
```

mPaaS 插件安装完毕之后，插件中选择文件时速度很慢

1. 禁止 Xcode 访问通讯录：
   - 打开 **System Preferences** > **Security & Privacy** > **Privacy** > **Contacts**，确保 Xcode 的勾选已去掉。
2. 若第 1 步无法解决问题，则说明触发了系统 bug。打开 **其他** 文件夹中的 **字体册** 应用。
3. 找到所有标记为 **关闭** 的字体，依次把他们开启即可。
4. 若还是没有解决，请将系统语言切换成英文。

mPaaS 插件安装完毕之后，启动 Xcode 的时候发生闪退

这是由于机器原本安装的插件不兼容 Xcode 8+，检查 `~/Library/Application\ Support/Developer/Shared/Xcode/Plug-ins` 目录下除了 `mPaaSPlugin.xcplugin` 是否有其他插件，如果有，请删除，然后重启 Xcode。

7.4. 命令行工具

7.4.1. 命令列表

安装了 **开发者工具** 后，除了 **mPaaS 插件**，您还可以使用 **命令行工具** 辅助开发。

命令列表如下：

分类	命令
	mpaas project create
	mpaas project target

分类	命令
工程管理命令	mpaas project import
	mpaas project edit
	mpaas project upgrade
SDK 管理命令	mpaas sdk version
	mpaas sdk list
	mpaas sdk search
基础工具命令	mpaas inst hotpatch sign
	mpaas inst hotpatch package
	mpaas inst sgimage
Xcode 插件命令	mpaas xcode unsign
	mpaas xcode restore
	mpaas xcode plugins version
	mpaas xcode plugins install
	mpaas xcode plugins uninstall
	mpaas xcode plugins update
	mpaas xcode plugins refresh
更新开发者工具命令	mpaas update
环境配置命令	mpaas env
诊断命令	mpaas diagnose report

7.4.2. 工程管理命令

工程管理命令用于 管理 Xcode 工程 以及 集成 mPaaS 组件。包括：

- mpaas project create
- mpaas project target
- mpaas project import
- mpaas project edit
- mpaas project upgrade

mpaas project create

```
mpaas project create [OPTIONS] <NAME>
```

创建 Xcode 工程，支持基于原生框架和基于 mPaaS 框架的工程。参数 `<NAME>` 表示创建的工程名称。

Options

`-o, --output=PATH`	创建的工程路径(默认为执行命令的当前目录)
`-c, --cloud-config=FILE`	应用对应的云端数据配置文件
`--modules=A,B`	添加的 mPaaS 模块
`--org=NAME`	工程的组织名称
`--class-prefix=PREFIX`	工程中类名前缀
`--project-type=TYPE`	工程的类型(基于原生框架的工程，mPaaS 框架工程) [sys, mpaas]
`--app-type=TYPE`	创建应用的类型(标签，抽屉，导航栏，空应用) [tab, drawer, navigation, empty]
`--force`	如果创建的输出目录存在是否强制覆盖
`--copy`	是否是 copy 模式

通用选项参数

`-h, --help`	显示某个命令的使用帮助信息
`--verbose`	显示更多的 debug 信息
`--silent`	静默模式，不显示任何输出信息

使用示例

```
mpaas project create -o /path/to/project/root -c /path/to/Ant-mpaas-0D4F511232152-default-IOS.config --modules=TinyApp --copy MPaaSDemo
```

您可以通过命令 `mpaas project create -h` 查看帮助信息。

mpaas project target

```
mpaas project target [OPTIONS]
```

获取 Xcode 工程的 targets 信息。

Options

`-p, --project=PATH`	读入工程的 .xcodeproj 文件或 .xcworkspace 文件 [Required]
`-l, --list`	显示所有 targets 名称列表
`--json-format`	以 JSON 格式输出

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas project target -p /path/to/MPaaSDemo.xcodeproj -l --json-format
```

您可以通过命令 `mpaas project target -h` 查看帮助信息。

mpaas project import

```
mpaas project import [OPTIONS]
```

向已集成 mPaaS 的 Xcode 工程导入云端配置数据。

Options

```
-p, --project=PATH          待编辑工程的 .xcodeproj 文件路径 [Required]
-c, --cloud-config=FILE     应用对应的云端数据配置文件 [Required]
-t, --target=TARGET         待编辑工程的 target 名称 [Required]
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas project import -p /path/to/MPaaSDemo.xcodeproj -c /path/to/Ant-mpaas-0D4F111111111-de
fault-IOS.config -t MPaaSDemo
```

您可以通过命令 `mpaas project import -h` 查看帮助信息。

mpaas project edit

```
mpaas project edit [OPTIONS]
```

对已有的 Xcode 工程（基于原生框架或基于 mPaaS 框架的工程均可）进行编辑，包括模块的新增和删除。

Options

```
-p, --project=PATH            待编辑工程的 .xcodeproj 文件路径 [Required]
-c, --cloud-config=FILE       应用对应的云端数据配置文件 [Required]
-t, --target=TARGET           待编辑工程的 target 名称 [Required]
-a, --add=A,B                 新增的 mPaaS 模块
-d, --del=A,B                 移除的 mPaaS 模块
--copy                        是否是 copy 模式
```

通用选项参数

```
-h, --help                    显示某个命令的使用帮助信息
    --verbose                 显示更多的 debug 信息
    --silent                  静默模式，不显示任何输出信息
```

使用示例

```
mpaas project edit -p /path/to/MPaaSDemo.xcodeproj -c /path/to/Ant-mpaas-0D4F511232152-defa
ult-IOS.config -t MPaaSDemo -a MPCommonUI,Nebula -d APRemoteLogging
```

您可以通过命令 `mpaas project edit -h` 查看帮助信息。

mpaas project upgrade

```
mpaas project upgrade [OPTIONS]
```

对已经集成 mPaaS 组件的 Xcode 工程中的 mPaaS 模块进行升级，指定的模块会更新到最新版本。

Options

```
-p, --project=PATH            待编辑工程的 .xcodeproj 文件路径 [Required]
-t, --target=TARGET           待编辑工程的 target 名称 [Required]
-b, --baseline=VERSION        升级的基线版本
-m, --modules=A,B             升级的 mPaaS 模块
    --check                   检查工程的模块升级信息
    --copy                    是否是 copy 模式
```

通用选项参数

```
-h, --help                    显示某个命令的使用帮助信息
    --verbose                 显示更多的 debug 信息
    --silent                  静默模式，不显示任何输出信息
```

使用示例

```
mpaas project upgrade -p /path/to/MPaaSDemo.xcodeproj -t MPaaSDemo --modules=APMobileLBS
```

您可以通过命令 `mpaas project upgrade -h` 查看帮助信息。

7.4.3. SDK 管理命令

SDK 管理命令用于获取 mPaaS SDK 的相关信息。包括：

- mpaas sdk version
- mpaas sdk list
- mpaas sdk search

mpaas sdk version

```
mpaas sdk version [OPTIONS]
```

显示当前发布的最新 mPaaS SDK 版本号（基线版本）。

Options

```
-l, --list                  列出所有支持的版本列表
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas sdk version
```

您可以使用命令 `mpaas sdk version -h` 查看帮助信息。

mpaas sdk list

```
mpaas sdk list [OPTIONS]
```

查看本地安装的 mPaaS 组件 SDK 的信息。

Options

```
-b, --baseline=VERSION      指定的版本，默认为最新版本
-d, --details               展示组件 SDK 的详细信息
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas sdk list -d
```

您可以使用命令 `mpaas sdk list -h` 查看帮助信息。

mpaas sdk search

```
mpaas sdk search [OPTIONS] <NAME>
```

查找本地或远程某个 SDK，并显示其详细信息。支持模糊查询。

Options

```
-b, --baseline=VERSION        指定的基线版本，默认为最新基线
-d, --details                 展示组件 SDK 的详细信息
-l, --local                   在本地安装的 SDK 中查找
```

通用选项参数

```
-h, --help                    显示某个命令的使用帮助信息
    --verbose                 显示更多的 debug 信息
    --silent                  静默模式，不显示任何输出信息
```

使用示例

```
mpaas sdk search log -d -l
```

您可以使用命令 `mpaas sdk search -h` 查看帮助信息。

7.4.4. 基础工具命令

本文介绍为 mPaaS 开发提供的基础工具集，包括热修复和无线保镖图片相关命令。具体命令如下：

- mpaas inst hotpatch sign
- mpaas inst hotpatch package
- mpaas inst sgimage

mpaas inst hotpatch sign

```
mpaas inst hotpatch sign [OPTIONS]
```

获取生成热修复包的密钥签名。

Options

```
-i, --input=INPUT             生成热修复资源包对应的 rsa 公钥（.pem 文件） [Required]
-o, --output=PATH             签名文件的输出路径
-p, --private-pem=PEM         生成热修复资源包对应的 rsa 私钥（.pem 文件） [Required]
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas inst hotpatch sign -i /path/to/rsa_public_key.pem -p /path/to/rsa_private_key.pem -o
/path/to/output.sig
```

您可以使用命令 `mpaas inst hotpatch sign -h` 查看帮助信息。

mpaas inst hotpatch package

```
mpaas inst hotpatch package [OPTIONS]
```

根据应用的云端数据配置信息和 RSA 密钥，对原始脚本文件进行加签，生成热修复包。

> **说明**
>
> - 专有云用户需要提供 App Secret 的值（`--app-secret`），公有云用户需要提供云端数据配置文件（`-c`）。
> - 您可以使用已有的密钥文件（`-p`），也可以让命令自动生成（`--gen-key`）。

Options

```
-i, --input=INPUT           需要打包的脚本文件 [Required]
-o, --output=PATH           生成热修复包的输出路径
-c, --cloud-config=CONFIG   应用的云端数据配置文件（公有云必要参数）
    --app-secret=SECRET     应用的 app secret 值（专有云必要参数）
-p, --private-pem=PEM       打包所用的 rsa 私钥文件（.pem 文件）
    --gen-key               是否需要自动生成 rsa 密钥
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas inst hotpatch package -i /path/to/script.js -p /path/to/rsa_private_key.pem -o /path/
to/output --app-secret=xxxx
```

您可以使用命令 `mpaas inst hotpatch package -h` 查看帮助信息。

mpaas inst sgimage

```
mpaas inst sgimage [OPTIONS]
```

生成无线保镖图片。

Options

```
-c, --cloud-config=CONFIG     应用的云端数据配置文件 [Required]
-V, --jpg-version=VERSION     生成无线保镖图片的版本（默认 V5 版本）
-o, --output=OUTPUT           无线保镖图片的输出路径
```

通用选项参数

```
-h, --help                    显示某个命令的使用帮助信息
    --verbose                 显示更多的 debug 信息
    --silent                  静默模式，不显示任何输出信息
```

使用示例

```
mpaas inst sgimage -c /path/to/Ant-mpaas-0D4F511111111-default-IOS.config -o /path/to/outpu
t
```

您可以使用命令 `mpaas inst sgimage -h` 查看帮助信息。

7.4.5. Xcode 插件命令

Xcode 插件命令用来管理 mPaaS 的 Xcode 插件。包括：

- mpaas xcode unsign
- mpaas xcode restore
- mpaas xcode plugins version
- mpaas xcode plugins install
- mpaas xcode plugins uninstall
- mpaas xcode plugins update
- mpaas xcode plugins refresh

mpaas xcode unsign

```
mpaas xcode unsign
```

去除 Xcode 签名。

通用选项参数

```
-h, --help                    显示某个命令的使用帮助信息
    --verbose                 显示更多的 debug 信息
    --silent                  静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode unsign
```

mpaas xcode restore

```
mpaas xcode restore
```

恢复 Xcode 签名。

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode restore
```

mpaas xcode plugins version

```
mpaas xcode plugins version
```

显示当前安装 mPaaS 的 Xcode 插件版本号。

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode plugins version
```

mpaas xcode plugins install

```
mpaas xcode plugins install [OPTIONS]
```

安装 mPaaS 的 Xcode 插件。

> 说明 新版 Xcode 插件和 mPaaS 开发者工具有一定的版本依赖，默认的安装会选取匹配的最新版本。

Options

```
-V VERSION                  安装指定版本的 Xcode 插件
--local=PATH                从本地安装包进行安装
--latest                    安装最新版本的 Xcode 插件！慎用，最新版有可能和当前 mPaaS 工具不兼容
导致功能失效。
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode plugins install
```

您可以通过命令 `mpaas xcode plugins install -h` 查看帮助信息。

mpaas xcode plugins uninstall

```
mpaas xcode plugins uninstall
```

卸载当前安装的 mPaaS 的 Xcode 插件。

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode plugins uninstall
```

mpaas xcode plugins update

```
mpaas xcode plugins update
```

更新当前安装的 mPaaS 的 Xcode 插件。

> **说明** 新版 Xcode 插件和 mPaaS 开发者工具有一定的版本依赖，默认的安装会选取匹配的最新版本。

Options

```
-V VERSION                  安装指定版本的 Xcode 插件
--local=PATH                从本地安装包进行安装
--latest                    安装最新版本的 Xcode 插件！慎用，最新版有可能和当前 mPaaS 工具不兼容导致功能失效。
```

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode plugins update
```

您可以通过命令 `mpaas xcode plugins update -h` 查看帮助信息。

mpaas xcode plugins refresh

```
mpaas xcode plugins refresh
```

刷新当前安装 mPaaS 的 Xcode 插件的 UUID。

通用选项参数

```
-h, --help                  显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas xcode plugins refresh
```

7.4.6. 更新开发者工具命令

更新开发者工具命令 `mpaas update` 用于更新 mPaaS 开发者工具，更新会覆盖原有的安装。

通用选项参数

```
    -h, --help              显示某个命令的使用帮助信息
    --verbose               显示更多的 debug 信息
    --silent                静默模式，不显示任何输出信息
```

使用示例

```
mpaas update
```

7.4.7. 环境配置命令

`mpaas env` 命令用于查看当前系统环境和 mPaaS 环境配置。

通用选项参数

-h, --help	显示某个命令的使用帮助信息
--verbose	显示更多的 debug 信息
--silent	静默模式，不显示任何输出信息

使用示例

```
mpaas env
```

7.4.8. 诊断命令

当使用 mPaaS 工具出现问题时，您可以执行 `mpaas diagnose report [OPTIONS]` 诊断命令生成诊断报告，进而进行问题排查。

Options

-o, --output=PATH	诊断报告的输出目录（默认为当前目录）

通用选项参数

-h, --help	显示某个命令的使用帮助信息
--verbose	显示更多的 debug 信息
--silent	静默模式，不显示任何输出信息

使用示例

```
mpaas diagnose report -o ~/Desktop
```

7.5. 开发小助手

7.5.1. 关于开发小助手

开发小助手 DevHelper 提供了常用的辅助开发工具和 mPaaS 专用辅助工具，该工具集通过 mPaaS 插件提供给开发者。在将开发小助手接入开发工程后，您就可以使用开发小助手进行调试、帮助开发。

在应用中，开发小助手以悬浮窗的形式出现，在应用中的任何界面都可以点击悬浮窗，唤出开发小助手主页。

7.5.2. 开发小助手的功能

开发小助手提供的功能可分为三类：mPaaS 工具、诊断工具、常用工具。

mPaaS 工具

mPaaS 工具类包含 离线包中心、环境切换、启动速度 和 埋点日志 四个功能。

> 说明
>
> mPaaS 工具类功能要求您将开发工程接入 mPaaS。

离线包中心

在离线包中心，您可以查看离线包应用信息、已安装离线包列表、JsApis 列表和 Plugins 列表，还可以关闭离线包验签、一键更新离线包。

> 说明
>
> 为了保证功能的完整性和独立性，离线包中心功能需要单独接入 离线包开发助手，更多详情请参见 接入开发小助手。


离线包应用中心
查看应用信息
已安装离线包列表
关闭离线包验签
获取 JsApis 列表
获取 Plugins 列表
一键更新

查看应用信息

输入要查看的应用 ID，即可以查看该应用的具体信息。

已安装离线包列表

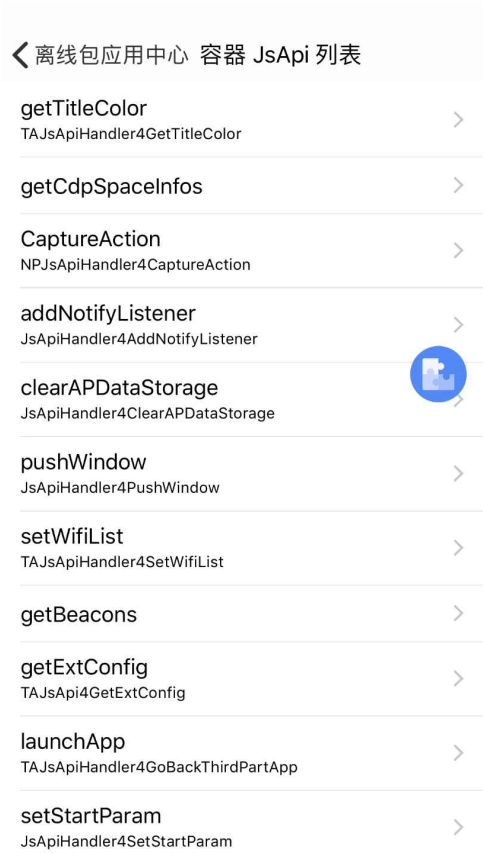
点击 **已安装离线包列表**，即可查看 App 中安装的离线包。

关闭离线包验签

点击 关闭离线包验签，即可将离线包验签功能关闭。

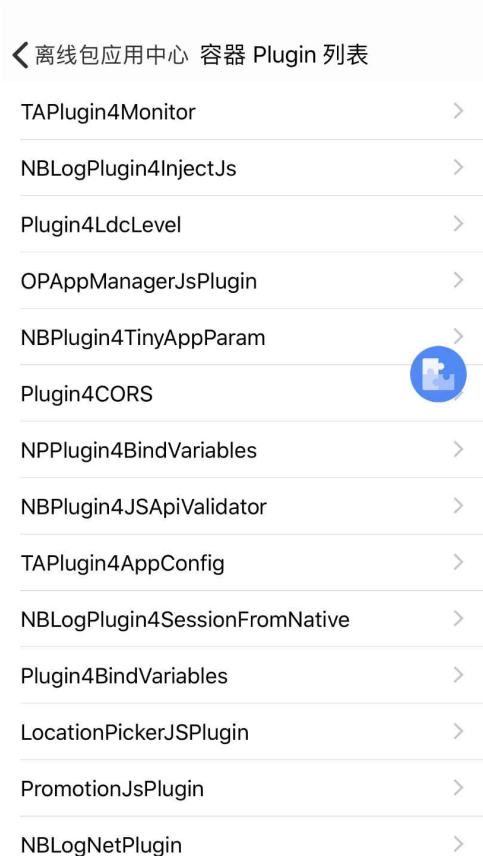
获取JsApis列表

点击 获取 JsApis 列表，即可获得所有 JsApis 并以列表形式展示。



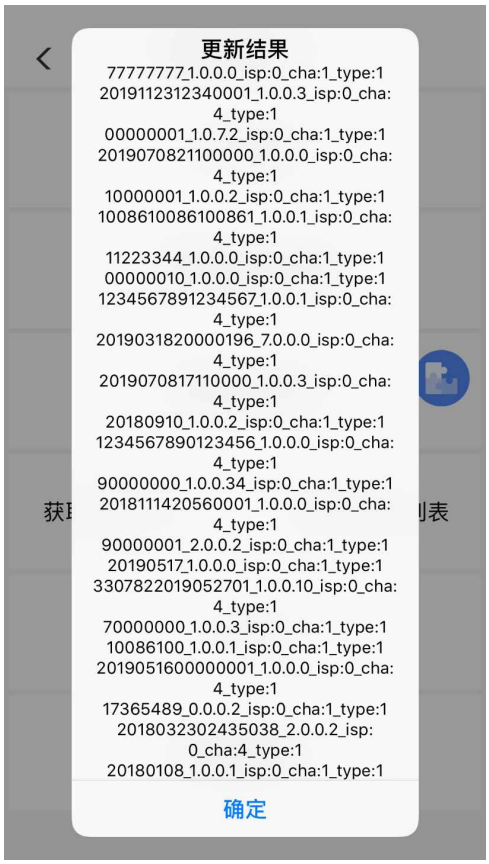
获取 Plugins 列表

点击 获取 Plugins 列表，可以获得当前应用的所有 Plugin。



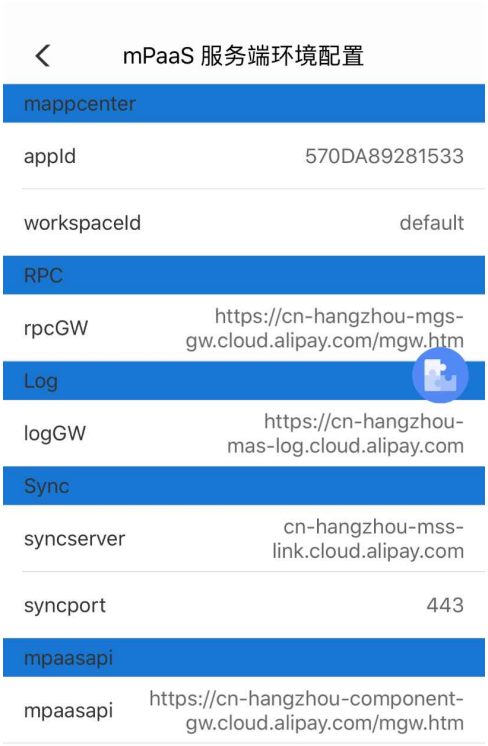
一键更新

点击 一键更新，即可将已经应用中预置的离线包进行批量更新，并将更新结果进行显示。



环境切换

环境切换功能支持在 mPaaS 服务端环境间进行切换。点击环境配置项目，即可修改该配置项，实现环境切换。

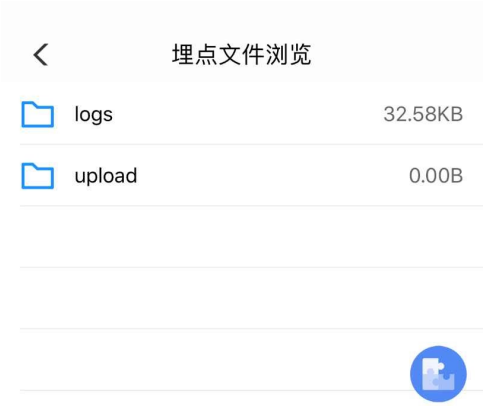


启动速度

启动速度功能支持查看 App 的启动速度，该功能需要接入 mPaaS 移动分析组件。

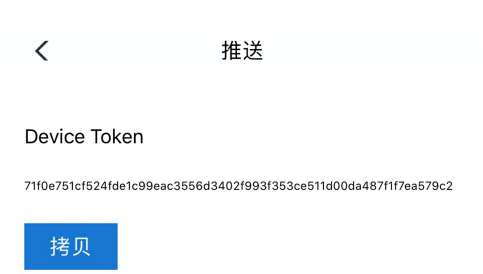
埋点日志

埋点日志支持查看暂未上传到服务端的日志。



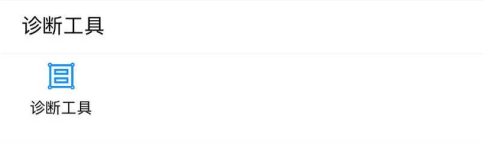
推送

推送功能支持查看推送的设备符号（Device Token）。



诊断工具

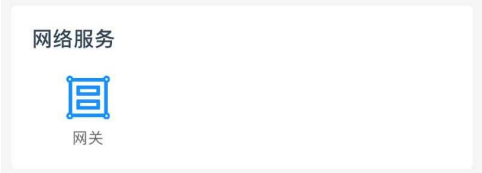
诊断工具类中只包含 **诊断工具** 一个功能。诊断工具支持诊断接入的 mPaaS 组件问题，目前提供了诊断网络服务。



网络服务诊断

网络服务诊断支持对网关服务进行诊断。通过诊断，可提供环境配置信息和组件诊断信息，如下图所示。

> **说明**
>
> 只有在出现特定错误日志时，组件诊断才会显示错误信息。

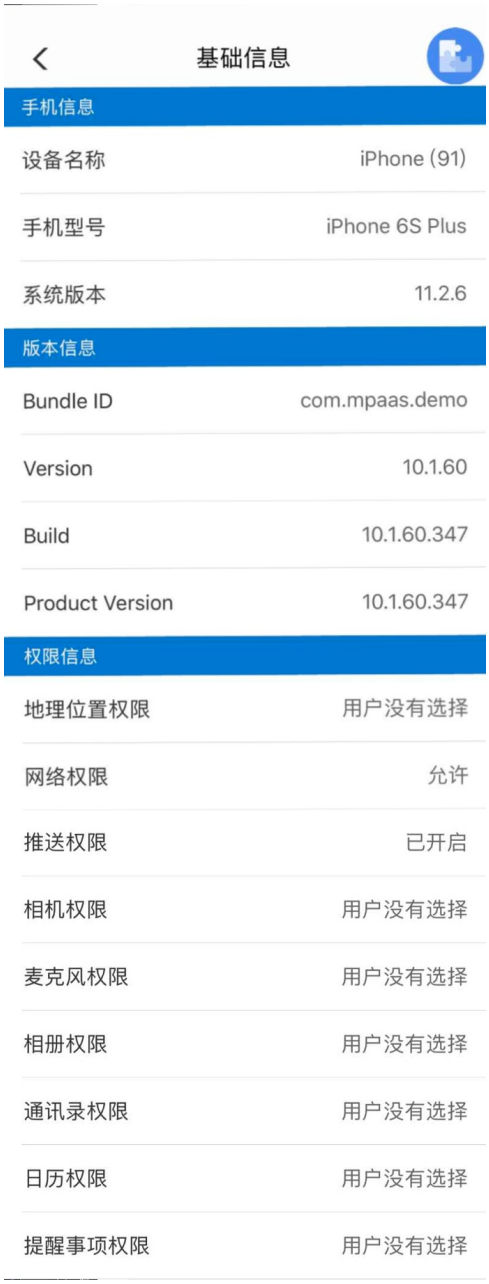


常用工具

常用工具类包含 基础信息、沙盒浏览、任意门、清除数据、NSlog、性能监控 和 组件检查 七个功能。

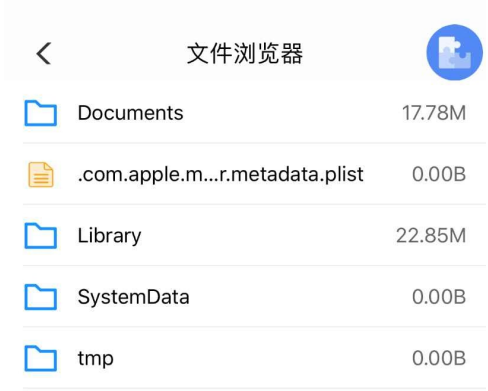
基础信息

基础信息能够将手机信息、应用版本信息和应用所获得的权限集中进行展示。具体信息如下图所示。



沙盒浏览

沙盒浏览功能支持查看 App 沙盒内中所有文件。



任意门

任意门功能支持通过开发小助手打开离线包或者在线页面。在输入网址、离线包 appid 或小程序 appid 后，即可在应用内打开网址、离线包或小程序；同样，您也可以使用扫描二维码实现在应用内打开网址、离线包或小程序。

清理数据

清理数据功能用于清理 App 的所有缓存以及数据信息。


清除本地数据
清理的目录：Documents、Library、tmp
占用空间：40.12 M
清理数据
清除本地数据
清理的目录：Documents、Library、tmp
占用空间：40.12 M
提示
确定要删除本地数据?
取消
确定
清理数据

NSLog

记录查看 NSLog 输出的所有日志。NSLog 日志的开关默认是关闭的，在开启 NSLog 日志记录开关 后，需要重启应用功能才能生效。

在启动 NSLog 日志记录开关 后，点击 查看 NSLog，即可查看 NSLog 日志记录。

性能监控

性能监控支持对帧率、CPU 占用率、内存占用率进行监控，监控结果会实时地以浮层形式进行显示。三种性能的监控可以分别开启、独立控制。

组件检查

组件检查功能可查看页面中 UI元素的位置信息，实时地在页面中进行标注。

> **说明**
>
> 使用组件检查功能时，页面中元素的功能将暂时失效，因此请在需要检查的页面使用此功能。检查结束后，点击 **开发小助手** 悬浮窗，就可以关闭组件检查功能。

拓展功能

凭借开发小助手预留的插件机制，您可以方便地对开发小助手进行功能拓展。开发小助手支持以下两种类型的插件以拓展其功能：**普通插件** 和 **注册插件**。

- **普通插件**：是指用户点击该插件图标之后才触发加载的插件。
- **注册插件**：是指开发小助手被打开后即刻触发加载的插件。

拓展后的功能会显示在开发小助手的主页上，您可以将拓展功能添加到现有的分类中，也可以新建分类来单独显示拓展的插件。关于如何拓展开发小助手，请参见 拓展开发小助手。

7.5.3. 使用开发小助手

本文将从接入、启动和卸载三方面向您介绍使用开发小助手相关的信息。

接入开发小助手

为保证功能的完整性和独立性，离线包中心（需先集成 H5 容器及离线包）功能需要以 **离线包开发助手** 的形式在 mPaaS 插件中单独接入，其余功能通过接入 **开发小助手** 完成接入。离线包小助手单向依赖开发小助手，离线包开发小助手需要和开发小助手同时接入，开发小助手可以单独接入。

作为开发助手，在最终发布到 App Store 时，需要从应用中移除开发小助手。为节省您的时间，建议您在专门用于测试的 Target 下添加开发小助手。

由于采用不同开发框架时，需要采用不同接入步骤不同。接下来将向您分别介绍在使用 mPaaS 插件和使用 CocoaPods 两种情况下的接入步骤。

使用 mPaaS 插件

前置条件

需保证 mPaaS 基线版本 ≥ 10.1.60。

接入步骤

在插件中选择 开发小助手 或 离线包开发助手。

后置条件

在正式发布的时候需要移除开发小助手。更多详情，参见 卸载开发小助手。

使用 CocoaPods

前置条件

- 已安装了 cocoapods-mPaaS 插件。
- 需保证 mPaaS 基线版本 ≥ 10.1.60。

接入步骤

使用 cocopods-mPaaS 的命令 可以生成如下 Podfile 文件，然后执行 `pod install` （或 `pod update` ）。

```
# mPaaS Pods Begin
plugin "cocoapods-mPaaS"
source "https://code.aliyun.com/mpaas-public/podspecs.git"
mPaaS_baseline '10.1.60'  # 请将 x.x.x 替换成真实基线版本
# mPaaS Pods End
# ------------------------------------------------------------------------
# Uncomment the next line to define a global platform for your project
platform :ios, '9.0'

target 'DevDemo' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!
  mPaaS_pod 'mPaaS_DevHelper'
  # mPaaS_pod 'mPaaS_NebulaDevHelper'
  # Pods for DevDemo
end
```

后置条件

在正式发布的时候需要移除开发小助手。更多详情，参见 卸载开发小助手。

启动开发小助手

开在 APP 启动完成时，调用 `[MPDevHelper start]` 。您可以在一个启动过程方法中调用，比如：

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

- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // 开启开发助手
    [MPDevHelper start];
    // ...
}
```

拓展开发小助手

在实际使用中，您可以根据实际需求，对开发小助手进行功能拓展，功能拓展分为普通插件拓展和注册插件拓展。接下来将分别向您介绍这两种拓展方式。

普通插件拓展

1. 定义插件。

   实现 `MDHPluginProtocol` 协议中的 `pluginDidLoad` 方法。

   ```
   @interface MDHNSLogPlugin : NSObject<MDHPluginProtocol>

   @end

   @implementation MDHNSLogPlugin

   - (void)pluginDidLoad
   {
       // 自定义的 MDHNSLogViewController 需要继承 MDHBaseViewController
       MDHNSLogViewController *vc = [[MDHNSLogViewController alloc] init];
       [MPDevHelper openPlugin:vc];
   }
   ```

2. 注册插件。

   在 `MPDevHelper.bundle` / `MDHPlugins.plist` 的 plugins 中注册插件。

   - title：名称。
   - icon：图标。
   - class：实现类。
   - module：所属模块。

3. 在 `MPDevHelper.bundle` 中添加插件使用图标 icon 。

4. 定义插件在开发小助手中显示的位置。

   拓展后的功能会显示在开发小助手的主页上，您可以将拓展功能添加到现有的分类中，也可以新建分类 module 来单独显示拓展的插件。

注册插件拓展

1. 定义插件。

   实现 `MDHInitiatePluginProtocol` 协议方法 `initiatePluginDidLoad` 做初始化。

2. 注册插件。

在 `MPDevHelper.bundle` / `MDHPlugins.plist` 的 initiatePlugins 中注册插件。

- desc：描述。
- class：实现类。

移除拓展插件

如果要移除某个拓展插件，只需要删掉插件的定义代码，将注册信息从 plist 文件中删除即可。

卸载开发小助手

- 如果您是在专门用于测试的 Target 中添加的开发小助手，那么开发小助手不会对您的发布产生影响，您可以不用卸载开发小助手。
- 如果您是在正式发布中使用的 Target 中添加的开发小助手，则需要按照以下两种方式移除开发小助手的库和资源。
  - 通过mPaaS插件移出开发小助手，并删除助手启动代码 `[MPDevHelper start]` 。
  - 手动删除以下库和资源 MPDevHelper、MPComponentDiagnose、MPBaseTest、AppDoctor（只有在集成了离线包开发助手时才会引入 AppDoctor）。

在按照以上任一方式移除开发小助手后，再人工确认 framework 和 bundle 中不包含上述的四个库或资源，即说明已成功卸载开发小助手。

7.6. 公有云答疑小助手

关于公有云答疑小助手

公有云答疑小助手是 mPaaS 研发团队提供的钉钉群答疑机器人，它不仅可以帮助开发者快速了解、接入 mPaaS 框架，还可以协助排查使用 mPaaS 框架过程中出现的常见问题，提高开发效率。

> **说明**
>
> 目前公有云答疑小助手只支持对公有云使用相关的问题进行答疑。

公有云答疑小助手使用指南

在 mPaaS 公有云相关的官方答疑钉钉群里直接 **@公有云答疑小助手** 并输入相关问题或者关键字，公有云答疑小助手会根据输入的内容进行回答。

公有云答疑小助手还在成长中，如果公有云答疑小助手不能识别输入的内容，请更换内容尝试。我们会继续完善公有云答疑小助手的功能，让它变得越来越智能、强大。

8.iOS 适配说明

8.1. mPaaS 10.1.68 升级指南

mPaaS 10.1.68 发布说明

1. 从 10.1.68 基线开始正式废弃 UIWebView，只支持 WKWebView，详情可参考 mPaaS 适配 WKWebView。App Store 从 2020 年 4 月起不再接受使用 UIWebView 的新 APP，从 2020 年 12 月起不再接受使用 UIWebview 的 APP 的更新。请您尽快升级到 10.1.68 基线，适配 WKWebView。
2. 支持 Xcode 11 构建静态库打包，兼容 Xcode 11 开发。

mPaaS 10.1.68 升级指南

使用 mPaaS Xcode Extension 进行升级

前提条件

mPaaS Xcode Extension 已更新为 1.1.0 或更高版本。关于更新 mPaaS Xcode Extension 请参见 更新 mPaaS Xcode Extension。

操作步骤

1. 在 Xcode 菜单中，点击 **Editor > mPaaS > 编辑工程**，打开 mPaaS Xcode Extension。
   - 如果您已经集成了10.1.68 基线版本，请选择 **更新产品集** 选项，升级到最新产品集即可。
   - 如果您使用的基线版本非 10.1.68，请选择 **升级基线** 选项，选择升级的基线版本为 10.1.68。升级时间可能比较长，请您耐心等待，不要关闭插件窗口。
2. 更新或升级成功后，插件上即展示工程的 SDK 版本为 **10.1.68**。

使用 CocoaPods 接入升级

前提条件

已安装 CocoaPods mPaaS 插件。

- 如您尚未安装 CocoaPods mPaaS插件，请您在终端执行以下脚本安装 CocoaPods 插件。

```
  sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPodsPlugin.sh)
```

- 如您已安装了 CocoaPods mPaaS 插件，则可以直接使用升级命令 `pod mpaas update --all` 升级插件。更多 CocoaPods mPaaS 插件使用信息，请参见 基于原生框架且使用 CocoaPods 接入。

操作步骤

1. 在 Podfile 中，将 SDK 版本设置改为 **10.1.68**。

2. 执行 `pod mpaas update 10.1.68`，即可安装 **10.1.68** 基线的最新 SDK。
3. 根据需要执行 `pod install` 或 `pod update` 即可完成对应工程下 10.1.68 的升级。

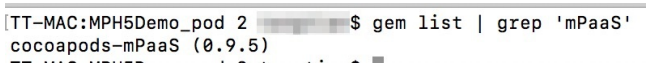
后续步骤

如果在 CocoaPods 接入时出现类似如下的错误：

```
Invalid `Podfile` file: [!] No mPaaS_Nebula : 10.1.68 found !!! Check name & version in Pod file.
```

可尝试按照以下步骤解决：

1. 运行 `gem list | grep 'mPaaS'` 命令检查 CocoaPods 插件版本，如下图所示。



2. 若 CocoaPods 插件版本 < 0.9.5，请运行以下脚本重新安装插件。

```
sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPodsPlugin.sh)
```

组件使用升级指南

如果您当前的基线版本 <10.1.60 且集成了 H5 容器、小程序和热修复组件，那么请您详细阅读下列说明：

- 请阅读 H5 容器 10.1.60 升级指南 了解 H5 容器和离线包升级的更多信息。
- 请阅读 小程序 10.1.60 升级指南 了解小程序升级的更多信息。
- 若当前基线版本 < 10.1.60，则热修复库也必须升级至 10.1.68 版本。由于线上可用的热修复库是通过技术支持人员提供，请您加入钉钉答疑群 41708565 联系 mPaaS 支持人员。

组件 API 变更

mPaaS 组件从 10.1.32 基线开始添加了适配层，如您使用的基线未使用适配层 API，请先行阅读 mPaaS 10.1.32 适配 iOS 13。

建议您在升级 SDK 后使用适配层的 API，具体可参考以下各组件文档中的旧版本升级注意事项：

- 移动网关
- 移动分析
- H5 离线包
- 移动同步
- 客户端诊断
- 发布管理
- 升级无线保镖到 V5

> **说明**
>
> - 需重点关注项目中 mPaaS 各组件的配置类的 category 和 info.plist 中的配置发生的变化。
> - 强烈建议您修改代码，使用中间层（适配器）方法而非直接使用底层方法，因为某些底层方法可能会在将来的版本中发生变更或废弃。如果您继续使用，在将来的更新中可能需要花费更多的时间进行适配。
> - iOS 热修复仅限专有云使用，更多详情，请咨询 mPaaS 技术支持。

定制库处理

10.1.68 基线版本各组件合入了定制化的需求，但是为了稳妥起见，如果此前您的依赖中包含定制库且是从低版本 SDK（如 10.1.32）升级至 10.1.68 版本，您的定制库可能需要基于新版本重新定制，请搜索群号 41708565 加入钉钉群进行咨询。

分享组件

10.1.68 版本分享组件中的第三方 SDK 已升级，包括微信 SDK，微博 SDK，QQ 互联 SDK。由于微信和 QQ 的分享在最新版本中增加了 Universal Link 的特性，因此请您注意对新版 SDK 进行适配，适配内容包括：

1. 对应平台的应用配置信息更新（在第三方开发者账号中的应用管理中查看），具体适配方式查看参考链接的内容。
2. 对于微信分享，mPaaS 分享组件的配置信息中需要增加字段 “universalLink”，取值为实际应用的 Universal Link 地址。

8.2. 隐私权限弹框的使用说明

背景

监管部门要求在用户点击隐私协议弹框中“同意” 按钮之前，App 不可以调用相关敏感 API。为应对此监管要求，mPaaS iOS 10.1.60.27 以上（60 版本） 和 10.1.32.18 以上（32 版本）的基线提供了支持，请您根据实际情况参考本文对工程进行改造。

使用方法

根据是否让 mPaaS iOS 框架托管 App 的生命周期，需要采用不同的使用方法。通过查看工程 `main.m` 文件中是否启用了框架的 `DFApplication` 和 `DFClientDelegate` ，可以判断是否让 mPaaS iOS 框架托管了 App 的生命周期；启用了框架的 `DFApplication` 和 `DFClientDelegate` 即表示进行了托管。

```
return UIApplicationMain(argc, argv, @"DFApplication", @"DFClientDelegate"); // NOW USE MPAAS FRAMEWORK
```

框架托管 App 生命周期

1. 允许隐私弹框提示

在 `MPaaSInterface category` 中重写以下 `enablePrivacyAuth` 接口方法，并返回 `YES` 。

代码示例

```objectivec
@implementation MPaaSInterface (Portal)

- (BOOL)enablePrivacyAuth
{
    return YES;
}

@end
```

2. 实现权限弹框

重写框架提供的 `- (DTFrameworkCallbackResult)application:(UIApplication *)application privacyAuthDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions completionHandler:(**void** (^)(**void**))completionHandler;` **方法。**

代码示例

```objectivec
- (DTFrameworkCallbackResult)application:(UIApplication *)application privacyAuthDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions completionHandler:(void (^)(void))completionHandler
{
    UIWindow *authWindow = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    authWindow.backgroundColor = [UIColor redColor];
    authWindow.windowLevel = UIWindowLevelStatusBar+5;
    AuthViewController *vc = [[AuthViewController alloc] init];
    vc.completionHandler = completionHandler;
    vc.window = authWindow;
    authWindow.rootViewController = vc;
    [authWindow makeKeyAndVisible];

    return DTFrameworkCallbackResultContinue;
}
```

3. 启动 mPaaS 框架

在用户点击 同意 授权后，回调 `completionHandler` **，继续启动 mPaaS 框架。示例代码如下所示：**

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

NS_ASSUME_NONNULL_BEGIN

@interface AuthViewController : UIViewController

@property (nonatomic, copy) void (^completionHandler)(void);
@property (nonatomic, strong) UIWindow *window;

@end

NS_ASSUME_NONNULL_END
```

```
#import "AuthViewController.h"

@interface AuthViewController ()<UIAlertViewDelegate>

@end

@implementation AuthViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    [self showAlertWithTitle:@"隐私权限"];
}

- (void)showAlertWithTitle:(NSString *)title
{
    if ([title length] > 0) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:title
                                                        message:nil
                                                       delegate:self
                                              cancelButtonTitle:@"取消"
                                              otherButtonTitles:@"确定", nil];
        [self.window makeKeyWindow];
        [alert show];
    }
}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if (buttonIndex == 1) {
        if (self.completionHandler) {
            self.completionHandler();
            self.window.rootViewController = nil;
            self.window = nil;
        }
    }else {
        exit(0);
    }
}



@end
```

4. 手动初始化容器 Context

如果您集成了 H5 容器和离线包、小程序组件，则需要在 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 方法中手动初始化容器 `Context`。代码示例如下。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  ...
  // 初始化容器Context
  [MPNebulaAdapterInterface setNBContextWhenEnablePrivacyAuth];

  ...
}
```

非框架托管 App 生命周期

1. 支持隐私弹框

在 `MPaaSInterface category` 中重写 `enableUserOverWriteAuthAlert` 接口方法，并返回相关隐私权限状态。

代码示例

```
@implementation MPaaSInterface (mPaaSdemo)

- (BOOL)enableUserOverWriteAuthAlert {
    // 如果隐私条款用户已经点过 “同意”，这里返回 “NO”，表示 mPaaS 组件可以正常调用相关 API。
    // 反之，返回 “Yes”，mPaaS 组件会 hold 住相关 API 调用。
    return ![[NSUserDefaults standardUserDefaults] boolForKey:@"xx_pr"];
}

@end
```

2. 阻止提前上报日志埋点

如果接入过埋点相关组件，需要在启动流程中额外调用 `MPAnalysisHelper holdUploadLogUntilAgreed` 方法，来阻止提前上报日志埋点。

> **说明**
>
> 可通过是否有 `APRemoteLogging.framework` 来判断是否接入过埋点相关组件。

代码示例

推荐在尽量早的时机调用。

```
#import "AppDelegate.h"
#import <MPMasAdapter/MPMasAdapter.h>

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    [MPAnalysisHelper holdUploadLogUntilAgreed];

    return YES;
}


- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can occur for certain types of tempor
```

3. 手动初始化容器 Context

如果您集成了 H5 容器和离线包、小程序组件，则需要在 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 方法中手动初始化容器 `Context` 。代码示例如下。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  ...
  // 初始化容器 Context
  [MPNebulaAdapterInterface setNBContextWhenEnablePrivacyAuth];

  ...
}
```

8.3. mPaaS 10.1.60 升级指南

关于 mPaaS 10.1.60

1. 10.1.60 基线已正式支持 WKWebView 接口，详情请参考 10.1.60 适配 WKWebView。由于 App Store 从 2020 年 4 月起不再接受使用 UIWebView 的新 APP，从 2020 年 12 月起不再接受使用 UIWebview 的 APP 的更新，详情请参见 苹果官方声明 。因此，请开发者注意使用 WKWebView 替换 UIWebView。
2. 10.1.60 基线最新版已适配 iOS 13 和 Xcode 11，详情可参考 mPaaS 10.1.60 适配 iOS 13。
3. 10.1.60 基线新增加了小程序组件。小程序正式版拥有更加完善的 API，且在稳定性、兼容性等方面有了大幅提高。关于小程序升级请参见 小程序升级指南，关于小程序 IDE 新增调试、预览、发布等功能的详情请参见 开发小程序。
4. 10.1.60 基线对 H5 容器整体进行大幅优化，提供了更加简化的接入流程，持续补强能力，在兼容性、稳定性等方面有显著提高。关于 H5 容器和离线包升级，请参见 H5 容器升级指南。
5. 10.1.60 基线新增加 智能投放 组件。智能投放提供了在应用内个性化投放广告的能力，支持针对定向人群进行个性化广告投放，帮助 APP 运营人员精准、及时触达用户，详情请参见 智能投放。
6. 10.1.60 基线的整体组件的兼容性、稳定性都有了大幅提高，功能也有着显著提升，具体的发布说明请参见 iOS SDK 发布说明。

7. 10.1.60 基线已不支持 iOS 8。

mPaaS 10.1.60 升级指南

使用 mPaaS Xcode Extension 进行升级（推荐）

前提条件

mPaaS Xcode Extension 已更新为 1.1.0 或更高版本。关于更新 mPaaS Xcode Extension 请参见 更新 mPaaS Xcode Extension。

操作步骤

1. 在 Xcode 菜单中，点击 **Editor > mPaaS > 编辑工程**，打开 mPaaS Xcode Extension。
   - 如果您已经集成了10.1.60-beta基线版本，请选择 **更新产品集** 选项，升级到最新产品集即可。
   - 如果您使用的基线版本非 10.1.60-beta，请选择 **升级基线** 选项，选择升级的基线版本为 10.1.60。升级时间可能比较长，请您耐心等待，不要关闭插件窗口。
2. 更新或升级成功后，插件上即展示工程的 SDK 版本为 **10.1.60**。

使用 mPaaS 插件接入升级

前提条件

Xcode mPaaS 插件已更新到 5.0.7 或以上的版本。

操作步骤

1. 在当前工程下打开插件面板。
   - 如果您已经集成了10.1.60-beta基线版本，切换至 **mPaaS产品集更新** 选项，升级到最新产品集即可。
   - 如果您使用的基线版本非 10.1.60-beta，请选择 **mPaaS基线升级** 选项，选择升级的基线版本为 10.1.60。升级时间可能比较长，请您耐心等待，不要关闭插件窗口。
2. 更新或升级成功后，插件上即展示工程的 SDK 版本为 **10.1.60**。

使用 CocoaPods 接入升级

前提条件

已安装 CocoaPods mPaaS 插件。

- 如您尚未安装 CocoaPods mPaaS插件，请您在终端执行以下脚本安装 CocoaPods 插件。

```
  sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPodsPlugin.sh)
```

- 如您已安装了 CocoaPods mPaaS 插件，则可以直接使用升级命令 `pod mpaas update --all` 升级插件。更多 CocoaPods mPaaS 插件使用信息，请参见 基于原生框架且使用 CocoaPods 接入。

1. 在 Podfile 中，将 SDK 版本设置改为 10.1.60。

2. 执行 `pod mpaas update 10.1.60`，即可安装 10.1.60 基线的最新 SDK。
3. 根据需要执行 `pod install` 或 `pod update` 即可完成对应工程下 10.1.60 的升级。

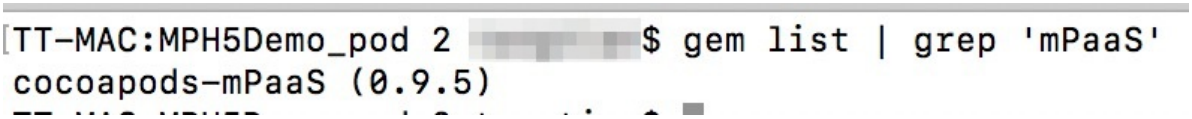
后续步骤

如果在 CocoaPods 接入时出现类似如下的错误：

```
Invalid `Podfile` file: [!] No mPaaS_Nebula : 10.1.60-beta found !!! Check name & version i
n Podfile.
```

可尝试按照以下步骤解决：

1. 运行 `gem list | grep 'mPaaS'` 命令检查 CocoaPods 插件版本，如下图所示。



2. 若 CocoaPods 插件版本 < 0.9.5，请运行以下脚本重新安装插件。

```
sh <(curl -s http://mpaas-ios.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPod
sPlugin.sh)
```

组件使用升级指南

10.1.60 基线中的 H5 容器、小程序和热修复组件在接入、使用等方面做了大幅调整。如您接入了上述组件，请详细阅读下列说明：

- 请阅读 H5 容器 10.1.60 升级指南 了解 H5 容器和离线包升级的更多信息。
- 请阅读 小程序 10.1.60 升级指南 了解小程序升级的更多信息。
- 热修复升级至 10.1.60 后，**热修复库也必须升级至 10.1.60 版本**。由于线上可用的热修复库是通过工单或技术支持人员提供，请您 提交工单 申请或联系 mPaaS 支持人员。

组件 API 变更

mPaaS 组件从 10.1.32 基线开始添加了适配层，如您使用的基线未使用适配层 API，请先行阅读 mPaaS 10.1.32 适配 iOS 13。

建议您在升级 SDK 后使用适配层的 API，具体可参考以下各组件文档中的旧版本升级注意事项：

- 移动网关
- 移动分析
- H5 离线包
- 移动同步

- 客户端诊断
- 发布管理
- 升级无线保镖到 V5

> 说明
>
> - 需重点关注项目中 mPaaS 各组件的配置类的 category 和 info.plist 中的配置发生的变化。
> - 强烈建议您修改代码，使用中间层（适配器）方法而非直接使用底层方法，因为某些底层方法可能会在将来的版本中发生变更或废弃。如果您继续使用，在将来的更新中可能需要花费更多的时间进行适配。
> - iOS 热修复仅限专有云使用，更多详情，请咨询 mPaaS 技术支持。

定制库处理

10.1.60 基线版本各组件合入了定制化的需求，但是为了稳妥起见，如果此前您的依赖中包含定制库且是从低版本 SDK（如 10.1.32）升级至 10.1.60 版本，您的定制库可能需要基于新版本重新定制，请 提交工单 或联系 mPaaS 支持人员确认。

8.4. mPaaS 适配 WKWebView

WKWebView 是苹果在 iOS8 中引入的新一代内置浏览器组件，用于替换陈旧的 UIWebView 组件。该组件最大的特点是支持多进程的渲染方式，包括页面滚动不影响图片资源加载，crash 不影响主进程，内存使用少等。相较于 UIWebView， WKWebView 在性能、稳定性和体验上都有较大提升。经过几年的发展和完善（iOS8, iOS9, iOS10, iOS11, iOS12），目前 WKWebView 已经逐渐处理了在推出初期存在的各类问题，运行情况趋于稳定。

在 iOS12 发布后，苹果 API 开始提示用户逐步废弃 UIWebView 接口。直至 2019 年 08 月，使用了 UIWebView 组件的 App 在提交到 App Store 进行审核时，均会收到以下警告，提醒开发者尽快切换到 WKWebView。

同时苹果在 2019 年 12 月 23 日宣布， App Store 从 2020 年 4 月起不再接受使用 UIWebView 开发的新 App，从 2020 年 12 月起不再接受使用 UIWebview 开发的已有 App 的更新。

针对此情况，mPaaS 对 WKWebView 进行了适配，支持您从 UIWebView 切换到 WKWebView。为了保证 H5 页面从 UIWebView 切换到 WKWebView 后的稳定性，mPaaS 对 WKWebView 的适配过程分为以下 2 个阶段：

- 第一阶段：自 2019 年 11 月起，mPaaS 基线支持 UIWebView 与 WKWebView 并存，通过灰度能力逐渐切换到 WKWebView。
- 第二阶段：自 2020 年 03 月起，mPaaS 基线完全删除 UIWebView 相关代码，所有 H5 业务全部切换到 WKWebView。

mPaaS 10.1.60 基线已完成第一阶段的适配工作，请集成 mPaaS H5 容器和小程序组件的用户尽快按以下说明升级到 10.1.60 最新基线（升级基线），并切换到 WKWebView（使用 WKWebView）。

升级基线

根据应用发布情况，请集成 mPaaS H5容器和小程序组件的用户，按以下原则选择对应基线进行升级适配 WKWebView。

- 2020 年 4 月前已经上架 App Store 的老应用：为保证您已有业务切换到 WKWebView 的稳定性，建议您升级到 10.1.60 基线，以支持该版本线上灰度和回滚能力。升级指南请参考 10.1.60 正式版本升级指南。

- 2020 年 4 月前仍未上架 App Store 的新应用：由于 4 月后 App Store 不再接受带有 UIWebView 的新 App，你必须使用完全删除 UIWebView 相关代码的 10.1.68 版本，同时做好业务回归测试验证。升级指南请参考 mPaaS 10.1.68-beta 升级指南。

使用 WKWebView

mPaaS 容器默认使用 UIWebView 加载 H5 页面，框架支持通过 **全局开启** 和 **灰度开启** 两种方式开启 WKWebView。

10.1.60 基线

10.1.60 基线下，mPaaS 容器中会同时并存 UIWebView 和 WKWebView，默认使用 UIWebView 加载 H5 页面，您可以按以下方式全局开启 WKWebView 开关，使所有使用 mPaaS 容器加载的页面均采用 WKWebView。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDict
ionary *)launchOptions
{
   //...
   // 全局开启 WKWebView 开关
   [MPNebulaAdapterInterface shareInstance].nebulaUseWKArbitrary = YES;
    //...
}
```

开启 WKWebView 后，您可以查看当前 H5 页面的 UA，若其中包含了 `WK` 字符串（如下图所示），则说明当前页面已成功切换使用 WKWebView。

其他说明

全局开启 WKWebView 之后，为保证线上 H5 页面功能的稳定性，mPaaS 框架提供了 线上止血 能力，帮助您快速切换 WKWebView 至 UIWebView。操作方法如下：在实时发布组件中添加 配置开关，在离线包或 URL 维度上限制其使用 UIWebView。

该配置开关的 配置键（key）为 `h5_wkArbitrary` ，资源值（value） 如下：

```
{
  "enable": true,
  "enableSubView": false,
  "exception": [
    {
      "appId": "^(70000000|20000193)$"
    },
    {
      "url": "https://invoice[.]starbucks[.]com[.]cn/"
    },
    {
      "url":"https://front[.]verystar[.]cn/starbucks/alipay-invoice"
    }]
}
```

value 配置项说明如下表所示。

配置项		说明	备注
enable		是否开启 WKWebView。`true` 为开启，`false` 为不开启。	默认为 `false` 。
enableSubView		小程序中内嵌 webView 是否开启 WKWebView。`true` 为开启，`false` 为不开启。	默认为 `false` 。
exception	appId	appId 正则匹配的离线包内的所有 H5 页面不使用 WKWebView。	默认为 `nil` 。此字段仅在 `enable` 为 `true` 时下生效。
	url	url 正则匹配的所有 H5 页面不使用 WKWebView。	

10.1.68-beta 基线

10.1.68-beta 版本容器默认使用 WKWebView 加载离线包和小程序，您可以查看当前 H5 页面的 UA。若其中包含了 `WK` 字符串（如下图所示），则说明当前页面已成功切换使用 WKWebView。

8.5. mPaaS 10.1.68 适配 Xcode 13

背景

2022 年 4 月起苹果要求所有提交至 AppStore 的 App 都必须使用 Xcode 13 来构建。针对全新的工具链，需要对 App 进行相关的适配。

现状

目前 mPaaS 已在 10.1.68.47 及以上的基线版中完成了对 Xcode 13 版本的适配和测试工作。

升级 SDK/组件

基于 Extension 插件升级

使用 mPaaS Xcode Extension 插件升级 SDK/组件，您可以选择以下两种方式：

- 更新产品集
- 升级基线

根据自身情况选择升级方式：

- 已经使用 Extension 插件管理组件依赖，但当前使用的基线版本低于 10.1.68，可使用 升级基线 功能升级至 10.1.68 版本。

  > 说明
  >
  > 当前使用的基线版本可在插件的 基线升级 中查看。

- 已经使用插件管理组件依赖，且当前使用的基线版本为 10.1.68，可使用 更新产品集 功能升级所使用到的模块。
- 未使用插件管理组件依赖。可按照以下步骤进行升级：

  i. 安装 mPaaS Xcode Extension。

  ii. 使用 编辑模块 功能选择 10.1.68 版本基线并添加所需模块。

基于 CocoaPods 升级

按照以下步骤，即可安装 10.1.68 版本的最新 SDK：

1. 首先确保 Podfile 中 mPaaS 组件的版本号为 10.1.68。
2. 执行 `pod mpaas update 10.1.68` 命令。如果提示命令报错，需通过 `pod mpaas update --all` 命令先更新插件，再重新执行。
3. 执行 `pod install` 命令 。

API 变更

本次适配暂无接口使用的变化。

定制基线的处理

如果您使用的是定制基线，可能需要基于新版本重新定制，请搜索群号 41708565 加入钉钉群咨询 mPaaS 支持人员。

适配 Xcode 13 更新的库清单

- 地图组件升级默认高德地图到 7.1.14 版本。
- 分享组件。
- 部分内部依赖组件。

测试验证范围

由于苹果工具链的升级为黑盒操作，经常会带来稳定性等问题，在完成 App 对 Xcode 13 的适配后，建议对 App 进行全面的回归测试。

8.6. mPaaS 10.1.68 适配 iOS 15

本文介绍了 mPaaS 10.1.68 版本基线为 iOS 15 进行的适配，以及用户需要完成的适配工作。

背景

iOS 15 已于 2021 年 9 月正式发布，针对全新的系统特性和接口，APP 需要进行相关的适配。mPaaS 在 10.1.68.38 及以上版本的基线中完成了对 iOS 15 的适配和测试工作。

现状

mPaaS 作为基础库，已经在 Xcode 12 构建 ipa 包下完成了 iOS 15 的适配和测试工作。如您的应用计划在苹果 App Store 上线，请 **使用 Xcode 12 打包**。Xcode 13 的相关工具链正在完善中。在工具链完善后，mPaaS 也会推出 Xcode 13 构建下适配 iOS 15 的版本。

升级 SDK/组件

基于 Extension 插件升级

使用 mPaaS Xcode Extension 插件升级 SDK/组件，您可以选择以下两种方式：

- 更新产品集
- 升级基线

您需要根据自身情况选择升级方式。如果您：

- 已经使用 Extension 插件管理组件依赖，但当前使用的基线版本低于 10.1.68，可使用 升级基线 功能升级至 10.1.68 版本。

  > **说明**
  >
  > 当前使用的基线版本可在插件的 **基线升级** 中查看。

- 已经使用插件管理组件依赖，且当前使用的基线版本为 10.1.68，可使用 更新产品集 功能升级所使用到的模块。
- 未使用插件管理组件依赖。可按照以下步骤进行升级：

  i. 安装 mPaaS Xcode Extension。

  ii. 使用 编辑模块 功能选择 10.1.68 版本基线并添加所需模块。

基于 CocoaPods 升级

按照以下步骤，即可安装 10.1.68 版本的最新 SDK：

1. 首先确保 Podfile 中 mPaaS 组件的版本号为 10.1.68。
2. 执行 `pod mpaas update 10.1.68` 。如果提示命令报错，需通过 `pod mpaas update --all` 命令先更新插件，再重新执行。
3. 执行 `pod install` 。

API 变更

本次适配无接口变化。

定制基线的处理

如果您使用的是定制基线，可能需要基于新版本重新定制，请搜索群号 41708565 加入钉钉群进行咨询。

适配 iOS 15 更新的库清单

- 小程序
- H5 容器

- 部分内部依赖组件

完成对 iOS 15 的适配后，建议在 iOS 15 中进行全面回归测试。

8.7. mPaaS 10.1.68 适配 iOS 14

背景

苹果已于 2020 年 9 月 17 日正式发布 iOS 14。针对全新的系统特性和接口，App 需要进行相关的适配。目前 mPaaS 已在 ≥10.1.68.17 版本的基线版中完成对 iOS 14 版本的适配和测试工作。

> **重要**
>
> mPaaS 作为基础库，目前 10.1.68.27 及以后的版本已经完成了 Xcode 12 构建下的 iOS 14 的适配工作。

升级 SDK/组件

基于 Extension 插件升级

使用 mPaaS Xcode Extension 插件升级 SDK/组件，您可以选择以下两种方式：

- 更新产品集
- 升级基线

您需要根据自身情况选择升级方式。如果您：

- 已经使用 Extension 插件管理组件依赖，但当前使用的基线版本低于 10.1.68，可使用 **升级基线** 功能升级至 10.1.68 版本。**说明：** 当前使用的基线版本可在插件的 **基线升级** 中查看。
- 已经使用插件管理组件依赖，且当前使用的基线版本为 10.1.68，可使用 **更新产品集** 功能升级所使用到的模块。
- 未使用插件管理组件依赖。可按照以下步骤进行升级：
  i. 安装 mPaaS Xcode Extension。
  ii. 使用 **编辑模块** 功能选择 10.1.68 版本基线并添加所需模块。

基于 CocoaPods 升级

按照以下步骤，即可安装 10.1.68 版本的最新 SDK：

1. 首先确保 Podfile 中 mPaaS 组件的版本号为 10.1.68。
2. 执行 `pod mpaas update 10.1.68` 。如果提示命令报错，需通过 `pod mpaas update --all` 命令先更新插件，再重新执行。
3. 执行 `pod install` 。

API 变更

接口使用的变化，可参考各组件中的适配部分。

- 定位

定制基线的处理

如果您使用的是定制基线，可能需要基于新版本重新定制，请 提交工单 或联系 mPaaS 支持人员确认。

适配 iOS 14 更新的库清单

- APMobileLBS
- MPLBSAdapter

8.8. mPaaS 10.1.60 适配 iOS 13

背景

iOS 13 已经于 2019 年 9 月 20 日正式发布。在 iOS 13 beta 及正式版的测试中，发现系统的部分行为发生了变化，因此 App 需要进行适配，否则可能会出现功能异常、Crash 等问题。

在 mPaaS 适配之前，在 iOS 13 设备上，Xcode 10 构建的 mPaaS SDK 受到的影响主要为：**由于 iOS 13 优化了 App 启动，修改了镜像的加载机制，导致系统 category 可能会覆盖 SDK 中定义的 category 方法，进而导致自定义的方法无法返回预期结果。**

> **说明**
>
> mPaaS 作为基础库，目前 10.1.60.26 及以后的版本已经完成了 Xcode 11 构建下的 iOS 13 的适配工作。

升级 SDK/组件

基于 Extension 插件升级

使用 mPaaS Xcode Extension 插件升级 SDK/组件，您可以选择以下两种方式：

- 更新产品集
- 升级基线

您需要根据自身情况选择升级方式。如果您：

- 已经使用 Extension 插件管理组件依赖，但当前使用的基线版本低于 10.1.60，可使用 升级基线 功能升级至 10.1.60 版本。

  > **说明**
  >
  > 当前使用的基线版本可在插件的 **基线升级** 中查看。

- 已经使用插件管理组件依赖，且当前使用的基线版本为 10.1.60，可使用 更新产品集 功能升级所使用到的模块。
- 未使用插件管理组件依赖。可按照以下步骤进行升级：

  i. 安装 mPaaS Xcode Extension。

  ii. 使用 编辑模块 功能选择 10.1.60 版本基线并添加所需模块。

热修复库

若当前基线不是 10.1.60，**则热修复库也必须升级至 10.1.60 版本**。由于线上可用的热修复库是通过技术支持专人提供的，您需要加入钉钉答疑群 41708565 联系 mPaaS 支持人员。

API 变更

mPaaS 组件在 10.1.32 及以上版本中添加了适配层，建议您在升级 SDK 后使用适配层的 API，具体可参考以下各组件文档中的旧版本升级注意事项：

- 移动网关
- 移动分析
- H5 离线包
- 数据同步
- 客户端诊断
- 发布管理
- 热修复管理

> 说明
>
> iOS 热修复仅限专有云使用，默认公有云拉取到的 SDK 为空实现。

- 无线保镖升级到 V5

> 说明
>
> - 需重点关注项目中 mPaaS 各组件的配置类的 category 和 info.plist 中的配置发生的变化。
> - 强烈建议您修改代码，使用中间层（适配器）方法而非直接使用底层方法。因为某些底层方法可能会在将来的版本中发生变更或废弃，如果您继续使用，在将来的更新中可能需要花费更多的时间进行适配。

定制库处理

10.1.60 版本各组件合入了定制化的需求，但是为了稳妥起见，如果此前您的依赖中包含定制库，则需要按以下情况处理：

- 如果您是从低版本 SDK（例如 10.1.20）升级至 10.1.60 版本，您的定制库可能需要基于新版本重新定制，请加入钉钉答疑群 41708565 联系 mPaaS 支持人员确认。
- 如果您已使用 10.1.60 版本，则只需更新部分组件。参见下文的 适配 iOS 13 更新的库清单，检查您的定制库是否包含在其中。
  - 如果不包含，您可继续使用该定制库。
  - 如果包含，您的定制库可能需要重新定制，请加入钉钉答疑群 41708565 联系 mPaaS 支持人员。

适配 iOS 13 更新的库清单

- mPaas
- MPDataCenter
- MPPushSDK
- APMultimedia
- BEEAudioUtil
- BeeCapture
- BeeCityPicker

- BeeMediaPlayer
- BeePhotoBrowser
- BeePhotoPicker
- NebulaAppBiz
- NebulaBiz
- NebulaSecurity
- NebulaKernel
- NebulaSDKPlugins
- NebulaSDK
- NebulaConfig
- NebulaTinyAppDebug
- NebulaNetwork
- TinyAppCommon
- APConfig
- AntUI
- MPPromotion
- BeeLocation
- MPMpaaSService
- TinyAppService
- AMap

8.9. mPaaS 10.1.32 适配 iOS 13

iOS 13 已于 2019 年 9 月 19 日正式发布，在对 iOS 13 的测试中发现系统的部分行为发生了变化，因此 App 需要对其进行适配，否则可能会出现功能异常、Crash 等问题。

重要

自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。建议您选用 10.1.68 或 10.1.60 系列基线。

背景

iOS 13 已于 2019 年 9 月 19 日正式发布，在对 iOS 13 的测试中发现系统的部分行为发生了变化，因此 App 需要对其进行适配，否则可能会出现功能异常、Crash 等问题。

在 mPaaS 对 iOS 13适配之前，在 iOS 13 设备上由 Xcode 10 构建的 mPaaS SDK 受到的影响主要是：**由于 iOS 13 优化 App 启动，修改了镜像的加载机制，导致系统 category 可能会覆盖 SDK 中定义的 category 方法，进而导致自定义的方法无法返回预期结果。**

现状

mPaaS 作为基础库，目前已经完成了 Xcode 10 构建下的 iOS 13 的适配工作。由于 mPaaS 当前仅在 Xcode 10 打包下进行了适配，所以 **务必使用 Xcode 10 打包** 提交 App Store。Xcode 11 的相关工具链尚未完善，随着工具链的完善，mPaaS 会推出 Xcode 11 构建下的 iOS 13 适配版本。

升级 SDK/组件

基于插件升级

使用 mPaaS Xcode 插件升级 SDK/组件，您可以选择以下两种方式：

- mPaaS 模块升级
- mPaaS 基线升级

您需要根据自身情况选择升级方式。如果您：

- 已经使用插件管理组件依赖，且当前使用的 SDK 版本为 10.1.32。可使用 **mPaaS 模块升级** 功能升级所使用到的模块。
- 未使用插件管理组件依赖。可按照以下步骤进行升级：
  i. 安装插件。
  ii. 使用 **mPaaS 模块编辑** 功能选择您所需的模块。
  iii. 使用 **mPaaS 模块升级** 功能升级至 10.1.32 版本。

项目当前基线版本可以在插件的 **mPaaS 基线升级** 中查看，具体操作步骤参见 mPaaS 插件使用说明 。

基于 CocoaPods 升级

按照以下步骤，即可安装 10.1.32 版本的最新 SDK：

1. 首先确保 Podfile 中 mPaaS 组件的版本号为 10.1.32。
2. 执行 `pod mpaas update 10.1.32` 。如果提示命令报错，需通过 `pod mpaas update --all` 命令先更新插件，再重新执行。
3. 执行 `pod install` 。

热修复库

若当前基线不是 10.1.32，则**热修复库也必须升级至 10.1.32 版本**。由于线上可用的热修复库是通过技术支持专人提供的，您需要加入钉钉答疑群 41708565 联系 mPaaS 支持人员。

API 变更

mPaaS 组件在 10.1.32 版本中添加了适配层，建议您在升级 SDK 后使用适配层的 API，具体可参考以下各组件文档中的旧版本升级注意事项：

- 移动网关
- 移动分析
- H5 离线包
- 移动同步
- 客户端诊断
- 发布管理
- 热修复管理

> 说明
>
> iOS 热修复仅限专有云使用，默认公有云拉取到的 SDK 为空实现。

- 生成无线保镖图片

> 说明
>
> - 需重点关注项目中 mPaaS 各组件的配置类的 category 和 info.plist 中的配置发生的变化。
> - 强烈建议您修改代码，使用中间层（适配器）方法而非直接使用底层方法。因为某些底层方法可能会在将来的版本中发生变更或废弃，如果您继续使用，在将来的更新中可能需要花费更多的时间进行适配。

定制库处理

10.1.32 版本各组件合入了定制化的需求，但是为了稳妥起见，如果此前您的依赖中包含定制库，则需要按以下情况处理：

- 如果您是从低版本 SDK 升级至 10.1.32 版本，您的定制库可能需要基于新版本重新定制，请您需要加入钉钉答疑群 41708565 联系 mPaaS 支持人员确认。
- 如果您已使用 10.1.32 版本，则只需更新部分组件。参见下文的 适配 iOS 13 更新的库清单，检查您的定制库是否包含在其中。
  - 如果不包含，您可继续使用该定制库。
  - 如果包含，您的定制库可能需要重新定制，请您需要加入钉钉答疑群 41708565 联系 mPaaS 支持人员。

适配 iOS 13 更新的库清单

- mPaaS
- MPDataCenter
- APMultimedia
- BEEAudioUtil
- BeeCapture
- BeeCityPicker
- BeeMediaPlayer
- BeePhotoBrowser
- BeePhotoPicker
- NebulaAppBiz
- NebulaBiz
- NebulaSDKPlugins
- APConfig
- AntUI
- NebulaSDK

- TinyAppCommon
- MPPromotion

9.参考

9.1. iOS 定制导航栏

在 App 开发的过程中，经常会要求对顶部导航栏进行自定义。本文将介绍在基于 mPaaS 框架创建的页面中，自定义导航栏的方法。包括定制应用主题和定制某一个页面导航栏样式。

基础概念

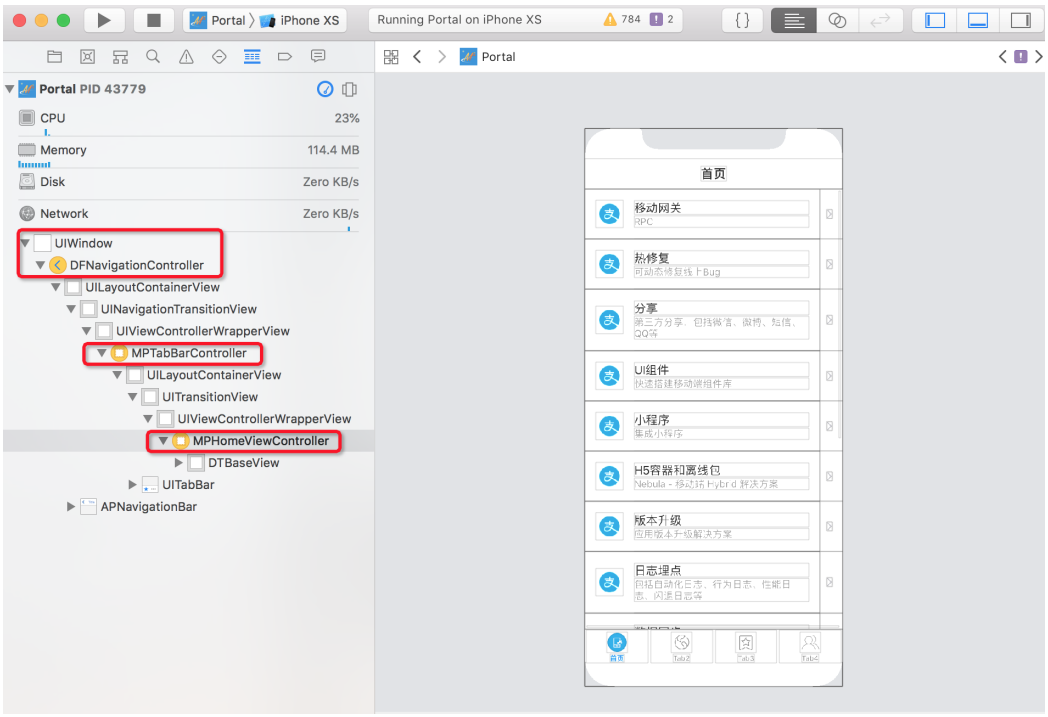
导航栏元素分布

导航栏元素主要分布在三个区域。一般对导航栏的定制化需求，最终都会变为对这几个区域的修改：

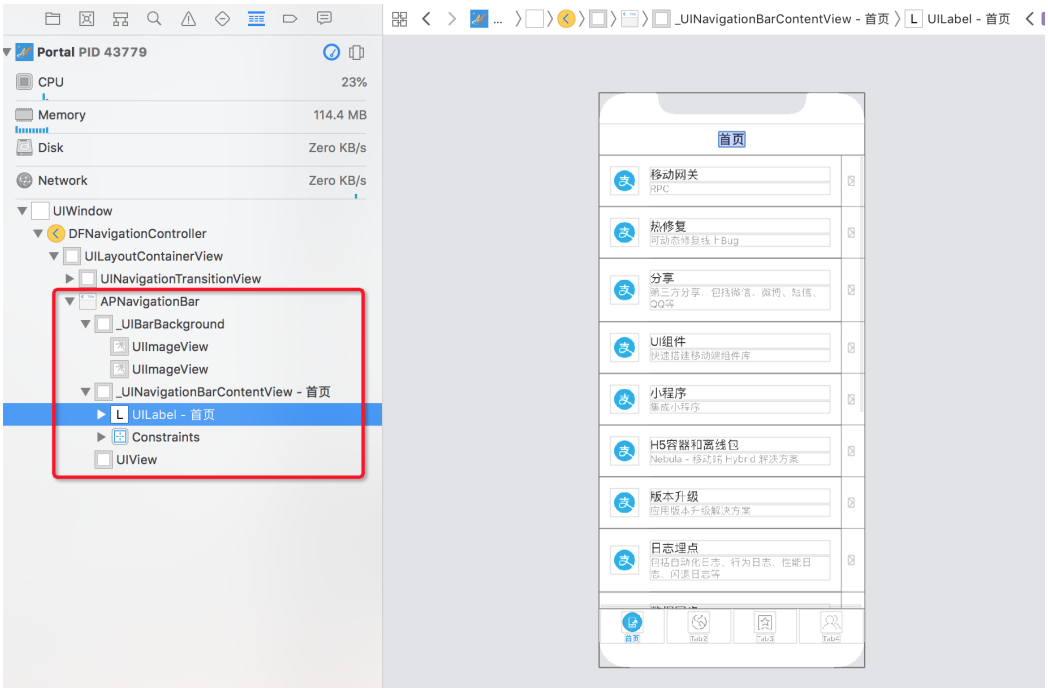
1. back：返回按钮控制区域，由 mPaaS 页面基类创建，默认样式为返回箭头 + 返回文案。
2. title/subTitle：标题栏控制区域，默认不显示。若需显示，请调用系统方法设置当前页面的 title。
3. optionMenu：页面菜单选项区域，默认不显示。如需显示，请调用系统方法设置当前页面的 rightNavigationItem。

导航栏结构

- 如下图所示，基于 mPaaS 框架创建的应用，默认的 UI 结构为：`window/navigationController` > `tabViewController` > 每个 tab 嵌入一个 `viewController` 。即应用主 window 的根应用是一个 `UINavigationController` 的对象，`UINavigationController` 的根应用是一个 `UITabViewController` 。



- 由以上 UI 结构可以看出，整个应用全局只有一个 `navigationController`，因此所有页面共用同一个导航栏（默认使用 `APNavigationBar` 创建）。



- 为了统一所有页面的导航栏样式，要求 mPaaS 应用中，所有页面所在的 VC 都要继承 `DTViewControler`，包括 native 和 H5 页面。
- 基于 mPaaS 框架创建的应用的默认主题，主白底黑字蓝按钮：

定制应用主题

每个应用都会有自己的主题风格，根据以下描述修改 mPaaS 应用的默认主题：

- 修改导航栏背景色、返回控制区域、标题控制区域等，可重写 AUThemeManager 类的 `au_defaultTheme_extraInfo` 方法，修改以下 key 对应的返回值。

- 接口方法

```
@interface AUThemeManager(AUExtendinfo)
/*支付宝客户端存在默认主题，独立 App 可修改该默认值
* 在该方法中只需返回与默认主题不同的键值对即可，请使用 AUTheme.h 中定义好的 key
*/
+(NSDictionary *)au_defaultTheme_extraInfo;

@end
/*
*  例如
*  +(NSDictionary*)au_defaultTheme_add_Info
*  {
*    NSMutableDictionary *dict = [INSMutableDictionary alloc] init];
*    dictITITLEBAR_BACKGROUND_COLOR] = AU_COLOR_APP_GREEN; // AUTitleBar 背景色
*    dit[TITLEBAR TITLE TEXTCOLOR1 = [UIColor redColor];   // AUTitleBar 标题色
*    ...
*    return dict;
*  }
*/
```

- 代码示例

```
@implementation AUThemeManager (Portal)

+ (NSDictionary *)au_defaultTheme_extraInfo
{
    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    dict[TITLEBAR_BACKGROUND_COLOR] = @"COLOR(#108EE9,1)"; // 导航栏背景色
    dict[TITLEBAR_LINE_COLOR] = @"COLOR(#108EE9,1)";       // 导航栏底部分割线或边线的颜色
    dict[TITLEBAR_TITLE_TEXTCOLOR] = @"COLOR(#ffffff,1)";  // 导航栏标题色
    dict[TITLEBAR_TITLE_TEXTSIZE_BOLD] = @"FONT(18)";      // 导航栏标题大小
    dict[TITLEBAR_TEXTCOLOR] = @"COLOR(#ffffff,1)";        // 导航栏返回按钮颜色

    return dict;
}

@end
}
```

> **说明**
>
> 颜色值必须使用类如 COLOR(#108EE9,1) 的方式，否则会报错。

- 修改主题配置中返回按钮图片，需重写 AUBarButtonItem 类中的 `au_default_backButtonImg` 方法。

- 接口方法

```
#import "AUUILoadDefine.h"//程序自动生成
#ifdef ANTUI_UI_TitleBar_AUBarButtonItem//程序自动生成
//
// AUBarButtonItem+AUExtendInfo.h
// AntUI
//
// Copyright © 2017 Alipay. All rights reserved.
//
#import "AUBarButtonItem.h"

@interface AUBarButtonItem(AUExtendInfo)

//支付宝返回按钮默认是蓝色 icon，独立 App 可修改返回按钮默认图标
+(UIImage *)au_default_backButtonImg;

@end
```

- 代码示例

```
@implementation AUBarButtonItem (CGBBarButtonItem)

+ (UIImage *)au_default_backButtonImg
{
    // 自定义返回按钮的图片
    return  APCommonUILoadImage(@"back_button_normal_white");

}
@end
```

- 修改所有页面的返回按钮样式和文案。

定制某一个页面导航栏样式

除了定制主题外，有时也需定制当前页面的导航栏的样式，如修改背景颜色、返回按钮样式等，根据修改时机不同，mPaaS 提供了不同的方法。

- **页面加载前**，在默认导航栏样式基础上修改导航栏颜色，可以在当前页面所在的 VC 中，实现 `DTNavigationBarAppearanceProtocol` 中的定义方法，来修改对应区域的颜色。

- 接口方法

```
@protocol DTNavigationBarAppearanceProtocol<NSObject>

@optional

/** 这个 DTViewController 是否要自动隐藏navigationBar，默认为 NO。业务某个 ViewController 需要隐藏 NavigationBar 可以重载此方法并返回 YES.
**/
-(BOOL)autohideNavigationBar;

/** 当前 VC 隐藏导航栏后，如果需要设置一个全透明的导航栏，且当前页面需设置与框架逻辑一致的返回文案，请重载此方法，并返回一个 APCustomNavigationView 的实例
-(UIView *)customNavigationBar;

/** 如果某个 viewcontroller 希望自己的 titlebar 是不透明，并且指定一个颜色，可以重写这个方法，并返回希望的颜色。
* 仅限于被 Push 的 VC，tabbar 里的 VC 还是不允许修改 navigationBar 的半透明属性
*/
-(UIColor *)opaqueNavigationBarColor;

/**
 *  如果某个viewcontroller希望修改状态栏的样式，请重写此方法，并返回希望的style
 */
- (UIStatusBarStyle)customStatusBarStytle;

/**
 *  如果某个viewcontroller希望修改导航栏标题的颜色，请重写此方法，并返回希望的颜色
 */
- (UIColor *)customNavigationBarTitleColor;
```

- 代码示例

```
#pragma mark DTNavigationBarAppearanceProtocol：进入页面时修改导航栏样式
- (UIColor *)opaqueNavigationBarColor
{
    // 设置当前页面导航栏背景为红色
    return [UIColor redColor];

//    // 设置当前页面导航栏透明
//    return [UIColor colorWithRGB:0xff0000 alpha:0];
}

- (BOOL)autohideNavigationBar
{
    // 设置当前页面导航栏是否隐藏
    return NO;
}

- (UIStatusBarStyle)customStatusBarStytle
{
    // 设置当前页面状态栏样式
    return UIStatusBarStyleDefault;
}

- (UIColor *)customNavigationBarBackButtonTitleColor
{
    // 设置当前页面返回按钮文案颜色
    return [UIColor greenColor];
}

- (UIImage *)customNavigationBarBackButtonImage
{
    // 设置当前页面返回按钮图片
    return APCommonUILoadImage(@"back_button_normal_white");
}

- (UIColor *)customNavigationBarTitleColor
{
    // 设置当前页面标题颜色
    return [UIColor greenColor];
}
```

- **页面打开后，在用户操作的过程中动态修改导航栏样式，如背景颜色滑动渐变、修改右侧菜单按钮等，根据修改的区域不同，主要分为以下几类：**

- 背景区域：包括隐藏/显示导航栏、透明导航栏、修改导航栏背景颜色、修改状态栏颜色。

```
- (void)gotoHideNavigator
{
    // 隐藏导航栏
    [self.navigationController.navigationBar setHidden:YES];
}

- (void)gotoShowNavigator
{
    // 显示导航栏
    [self.navigationController.navigationBar setHidden:NO];
}

- (void)gotoTransparency
{
    // 透明导航栏
    [self.navigationController.navigationBar setNavigationBarTranslucentStyle];
}

- (void)gotoUpdateBackgroundColor
{
    // 修改导航栏背景颜色
    [self.navigationController.navigationBar setNavigationBarStyleWithColor:[UIColor wh
iteColor] translucent:NO];
    [self.navigationController.navigationBar setNavigationBarBottomLineColor:[UIColor w
hiteColor]];
}

- (void)gotoUpdateStatusBarStyle
{
    // 修改状态栏颜色
    [[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleLightContent];
}
```

- 返回控制区域：修改默认返回按钮文案颜色、修改默认返回按钮返回箭头样式、重新设置返回按钮样式。

```
- (void)gotoUpdateBackTitleColor
{
    // 修改默认返回按钮文案颜色
    NSArray *leftBarButtonItems = self.navigationItem.leftBarButtonItems;
    if ([leftBarButtonItems count] == 1) {
        if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButtonI
tem class]]) {
            AUBarButtonItem *backItem = leftBarButtonItems[0];
            backItem.titleColor = [UIColor blackColor];
        }
    }
}

- (void)gotoUpdateBackImage
{
    // 修改默认返回按钮返回箭头样式
    NSArray *leftBarButtonItems = self.navigationItem.leftBarButtonItems;
    if ([leftBarButtonItems count] == 1) {
        if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButtonI
tem class]]) {
            AUBarButtonItem *backItem = leftBarButtonItems[0];
            backItem.backButtonImage = APCommonUILoadImage(@"back_button_normal");
        }
    }
}

- (void)gotoUpdateBackItem
{
    // 重新设置返回按钮样式
    self.navigationItem.leftBarButtonItem = [AUBarButtonItem barButtonItemWithImageType
:AUBarButtonImageTypeDelete target:self action:@selector(onClickBack)];
}

- (void)onClickBack
{
    [self.navigationController popViewControllerAnimated:YES];
}
```

- 标题控制区域：修改默认标题颜色、设置上下主副标题、修改标题为图片显示。

```
- (void)gotoUpdateTitleColor
{
    // 修改标题颜色
    [self.navigationController.navigationBar setNavigationBarTitleTextAttributesWithTextColor:[UIColor blackColor]];
}

- (void)gotoTwoTitle
{
    // 修改标题样式：上下主副标题
    self.navigationItem.titleView = [[AUDoubleTitleView alloc] initWithTitle:@"主标题" detailTitle:@"副标题"];
}

- (void)gotoTitleImage
{
    // 修改标题样式：图片
    UIImageView *imageView = [[UIImageView alloc] initWithImage:APCommonUILoadImage(@"illustration_ap_expection_alert")];
    imageView.frame = CGRectMake(0, 0, self.self.view.width-100, 64);
    self.navigationItem.titleView = imageView;
}
```

- 菜单控制区域：设置单个或多个右侧菜单按钮。

```
- (void)gotoSetOptionMenu
{
    // 设置右侧单按钮
    self.navigationItem.rightBarButtonItem = [AUBarButtonItem barButtonItemWithImageType:AUBarButtonImageTypeGroupChat target:self action:@selector(onClickRightItem)];
}

- (void)gotoSetTwoOptionMenu
{
    // 设置右侧双按钮
    AUBarButtonItem *item1 = [AUBarButtonItem barButtonItemWithImageType:AUBarButtonImageTypeGroupChat target:self action:@selector(onClickRightItem)];
    AUBarButtonItem *item2 = [AUBarButtonItem barButtonItemWithImageType:AUBarButtonImageTypeHelp target:self action:@selector(onClickRightItem)];
    self.navigationItem.rightBarButtonItems = @[item1, item2];
}
```

- 沉浸式导航栏：进入时导航栏透明，滑动到指定位置后不透明。主要分为以下两类：
  - 进入页面时，设置导航栏透明：在当前页面所在的 VC 中重写以下接口。

```
- (UIColor *)opaqueNavigationBarColor
  {
      // 设置当前页面导航栏透明
      return [UIColor colorWithRGB:0xff0000 alpha:0];
  }
```

- 页面滑动到指定位置后，修改导航栏背景区域、返回区域、标题区域及菜单控制区域等样式。

```
- (void)gotoUpdateBackgroundColor
{
    // 修改导航栏背景颜色
    [self.navigationController.navigationBar setNavigationBarStyleWithColor:[UIColor
whiteColor] translucent:NO];
    [self.navigationController.navigationBar setNavigationBarBottomLineColor:[UIColor
whiteColor]];
}

- (void)gotoUpdateBackTitleColor
{
    // 修改默认返回按钮文案颜色
    NSArray *leftBarButtonItems = self.navigationItem.leftBarButtonItems;
    if ([leftBarButtonItems count] == 1) {
        if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButto
nItem class]]) {
            AUBarButtonItem *backItem = leftBarButtonItems[0];
            backItem.titleColor = [UIColor blackColor];
        }
    }
}

- (void)gotoUpdateTitleColor
{
    // 修改标题颜色
    [self.navigationController.navigationBar setNavigationBarTitleTextAttributesWithT
extColor:[UIColor blackColor]];
}
```

9.2. iOS 冲突处理

接入 mPaaS 时，mPaaS SDK 可能会和工程中引入的其他开源库或三方库发生冲突，导致工程编译不通过。本文介绍了两类常见冲突的解决方案。

根据引起冲突的库的类型，可以将解决方案分为以下两类：

- mPaaS 定制库：若发生冲突的 mPaaS SDK 为定制库，则必须使用这些 mPaaS 库。
- 非 mPaaS 定制库：若发生冲突的 mPaaS SDK 非 mPaaS 定制库，可以将 mPaaS 引入的库进行删除。

mPaaS 定制库冲突解决方案

若发生冲突的 mPaaS SDK 为定制库，则必须使用这些 mPaaS 库。

开源库名	mPaaS 库名	冲突解决方案
AlipaySDK	AlipaySDK	必须使用 mPaaS 版本（解决了与 mPaaS RPC、UTDID 等模块的冲突）

OpenSSL	APOpenSSL	必须使用 mPaaS 版本（对原有国密算法进行优化）。更多详细信息，请参见 如何解决 iOS 工程中的 OpenSSL 三方库冲突 。
protocolBuffers	APProtocolBuffers	必须使用 mPaaS 版本

非 mPaaS 定制库冲突解决方案

若发生冲突的 mPaaS SDK 非 mPaaS 定制库，可以将 mPaaS 引入的库进行删除，支持删除的库如下表所示。详情请参见 移除冲突的三方库 移除引起冲突的库。

remove_pod 支持的组件	包含的开源库
mPaaS_SDWebImage	SDWebImage
mPaaS_Masonry	Masonry
mPaaS_MBProgressHud	MBProgressHUD
mPaaS_TTTAttributedLabel	TTTAttributedLabel
mPaaS_Lottie	Lottie
mPaaS_AMap	AMapSearchKit
	AMapFoundationKit
	MAMapKit
mPaaS_Security	SecurityGuardSGMain
mPaaS_APWebP	WebP

移除冲突的三方库

若发生冲突的 mPaaS SDK 非 mPaaS 定制库，可参照以下步骤删除 mPaaS 引入的库。

操作步骤

1. 安装 beta 版 cocoapods-mPaaS 插件。

> 说明
>
> cocoapods-mPaaS 插件 beta 版仅支持在 10.1.68 基线中使用。

```
sh <(curl -s http://mpaas-ios-test.oss-cn-hangzhou.aliyuncs.com/cocoapods/installmPaaSCocoaPodsPlugin.sh)
```

安装完成后，使用命令 `pod mpaas version --plugin` 确认是 beta 版本。

2. 重新运行命令更新本地基线：`pod mpaas update 10.1.68` 。
3. 使用 `mPaaS_pod` 命令之前，在 `podfile` 里引入 `remove_pod "mPaaS_xxx"` 。比如，在 `mPaaS_pod "mPaaS_CommonUI"` 之前使用 `remove_pod "mPaaS_SDWebImage"` 去除 `SDWebImage` 。

```
remove_pod "mPaaS_SDWebImage"

mPaaS_pod "mPaaS_CommonUI"

pod 'xxx' # 对应的三方原生库
```

4. 去除 mPaaS 的组件库后，可使用 `pod install` 命令引入原生的版本。

9.3. iOS 环境切换

应用开发过程中，经常会有更换应用环境信息或多套环境并行研发的需求。mPaaS 提供工具可帮助您在开发过程中方便地进行环境切换。根据切换环境的需求不同，分为以下两种方式：

- 静态切换环境
- 动态切换环境

静态切换环境

静态切换环境指客户端手动替换工程中默认的 `meta.config` 配置文件后，重新打包访问新环境。

> 说明
>
> 此方式仅适用于只更新当前应用环境配置信息的场景。

1. 使用 mPaaS 插件替换当前工程的 `meta.config` 文件。
2. 删除应用并重新安装，新的环境配置信息即可生效。

动态切换环境

动态切换环境指客户端不重新打包的情况下，通过修改手机设置中环境选项，动态修改应用的环境信息。

> 说明
>
> - 动态切换环境适用于开发阶段多套环境并存且频繁切换的场景。
> - 动态切换环境功能仅支持在专有云环境下使用。

由于 mPaaS 安全验签机制的限制，更新环境配置信息会修改无线保镖验签 `yw_1222.jpg` 图片，因此动态切换环境有两个限制：

- 动态切换环境仅适用于开发阶段，上线前请注意删除对应的配置。
- mPaaS 控制台需关闭网络请求验签开关，否则会因验签图片信息不对导致请求失败。

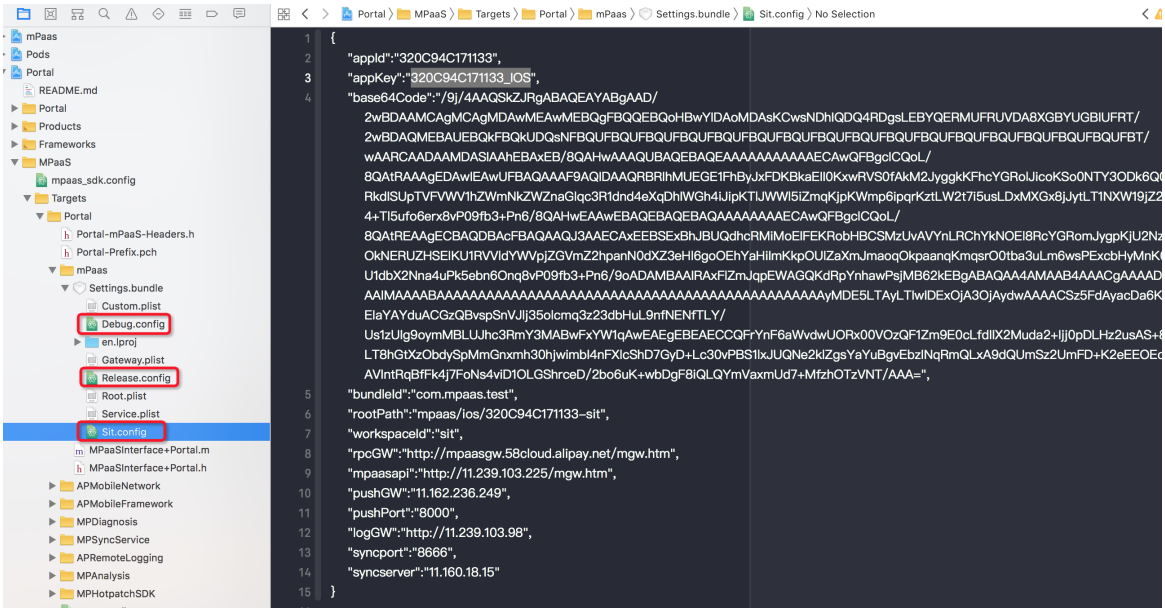
```
![ddd](http://docs-aliyun.cn-hangzhou.oss.aliyun-inc.com/assets/pic/111262/AntCloud_zh/1552905359956/%E5%9B%BE%E7%89%874.png)
```

环境信息配置

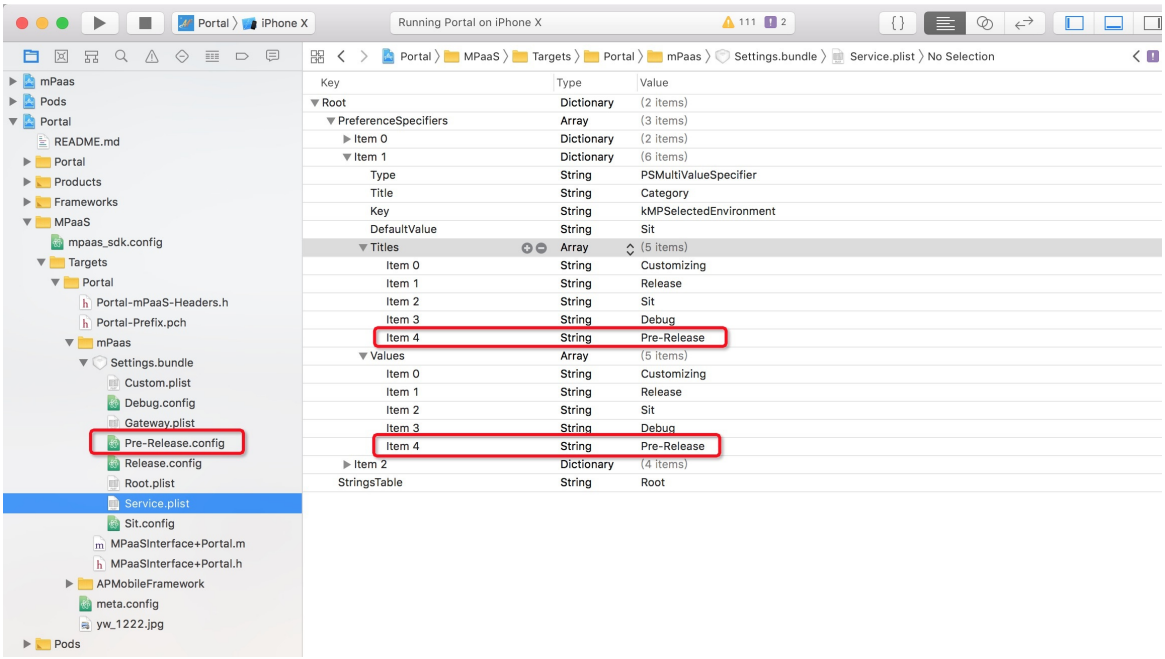
1. 打开动态切换环境开关：在 `MPaaSInterface` 的 `category` 中重写 `enableSettingService` 方法，并返回 YES。

```
@implementation MPaaSInterface (Portal)

- (BOOL)enableSettingService
{
    return YES;
}

@end
```

2. 下载 Settings.bundle.zip 环境配置文件，并添加到工程中。
3. 解读 `Setting.bundle` 中环境配置信息：
   - `Setting.bundle` 中默认提供四个环境，分别对应 Debug、Sit、Release 和 Custom 四个 config 文件。
   - 其中 Debug、Sit、Release 为预置的三个环境，您需要从 mPaaS 控制台下载您需要设置环境的 config 文件，重命名为的 Debug、Sit、Release 等名称后，用来替换原有的 `Setting.bundle` 中的示例文件。



- Custom 环境适用于在客户端不重新打包的情况下，直接在手机设置中配置应用的环境信息，配置路径： 手机设置 - 当前应用 - Settings - Customizing，根据下载的配置信息填写对应的 value 值即可。
- 如果除了默认的 Debug、Sit、Release、Custom 四个环境外，还需预置更多的环境，您可在 `Setting.bundle/Service.plist` 中增加设置项，并添加对应的 config 文件。格式如下：



动态切换环境

- 添加 Settings.bundle 环境配置文件后，应用的配置信息会覆盖工程中默认的 `meta.config` 文件，以 `Settings.bundle` 中选择的环境为主。当前选择的环境查看路径为：**手机设置 > 当前应用 > Settings > Category**，默认选择 Sit 环境。
- 如需切换到其他环境，直接在 **手机设置 > 当前应用 > Settings > Category** 中选择您需要的环境，杀进程重启后即可生效。

10.mPaaS 框架常见问题

本文介绍了使用 mPaaS 过程中的常见框架问题和相应的解决办法。

查看 mPaaS 框架常见问题列表，点击具体的问题查看解答：

升级 RubyGems 时出现 `ERROR: Failed to build gem native extension.d` 的错误

若升级 RubyGems 时出现错误 `ERROR: Failed to build gem native extension.` ，则安装 Xcode 命令行工具，然后再重试。

```
xcode-select --install
```

安装 RVM 时出现 `Library not loaded` 的错误

若使用 RVM 安装 Ruby 2.2.4 时出现错误 `For dyld: Library not loaded: /usr/local/lib/libgmp.10.dylib` ，则运行下面的命令，然后再重试。

```
brew update && brew install gmp
```

安装 RVM 时出现 `lazy symbol binding failed` 的错误

若使用 RVM 安装 Ruby 2.2.4 时出现错误 `dyld: lazy symbol binding failed: Symbol not found: _clock_gettime` ，则安装 Xcode 命令行工具，然后再重试。

```
xcode-select --install
```

如何使用自己的 UIApplication 代理类

如果不使用 mPaaS 的框架，您可以直接用自己的类覆盖 main 方法里的 `DFClientDelegate` 。

如何退出所有微应用，回到 Launcher

```
[DTContextGet() startApplication:@"Launcher 的 appid" params:nil animated:kDTMicroApplicationLaunchModePushNoAnimation];
```

当前应用 A 上层有 B 应用，B 应用如何重新启动 A 应用并传递参数

假设 A 应用已经启动，上层又启动了 B 应用，那么重新启动 A 应用会退出 B 应用（及 A 所有上层应用）。

```
[DTContextGet() startApplication:@"A 的 name" params:@{@"arg": @"something"} launchMode:kDT
MicroApplicationLaunchModePushWithAnimation];
```

同时 A 应用的 `DTMicroApplicationDelegate` 会接收到下面事件，`options` 里会携带参数。

```
- (void)application:(DTMicroApplication *)application willResumeWithOptions:(NSDictionary *
)options
{
}
```

基类继承自 `DTViewController` 之后，使用 xib 方式创建的 VC 打开白屏

请在 `DTViewController` 的 category 中重写 `loadView` 方法，代码示例如下：

```
@interface DTViewController (NibSupport)
@end

@implementation DTViewController (NibSupport)

- (void)loadView
{
    [super loadView];
}

@end
```