蚂蚁科技

接入 iOS
使用指南

文档版本：20240808

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。

商标声明

免责声明

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。

根据 iOS 开发工程的进展和使用场景，接入移动开发平台 mPaaS 的方式推荐采用 **基于已有工程且使用 CocoaPods 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。

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 创建阿里云账号。

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/2wBDAAMCAgMCAgMDAwMEAwMEBQgFBQQEBQoHBwYIDAoMD
KCwsNDhIQDQ4RDgsLEBYQERMUFRUVDA8XGBYUGBIUFRT/2wBDAQMEBAUEBQkFBQkUDQsNFBQUFBQUFBQUFBQUFB
FBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBT/wAARCAADAAMDASIAAhEBAxEB/8QAHwAAAQUBAQE
QEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEI
KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDh
Gh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09f
+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSE
hJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpj
VmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tba3uLm6wsPExcbHyMnK0tPU1dbX2
a4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxFobXF3FfU78U5nKhRBay4TiPPuh2kFAwABAQAAAAQAAB4AAA
AAAADAAAAIcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAKcUJ9ig78iKP1rG1ocLPmgO/Rk8SIdQLM81Pl439j7fGla3uzZwNWXK/ExCB3qqBmfDMqBKks0ZyAtn91
djbngBAwF4dWZ3AwECBgEAEQEBCAFxdHNqb3bWQSxDMewbnV9aVcN3qtd5KtBXv5coWb2f91l9bhimpuDibB7Ge
F/pNB/JlfH8kyyoP1sUTDB2qAOuXBez479sBz+S+5vSbzsH7QvEju8CsHs6RJMjcwGe4pYkDC5zsNtqyYZWj4fX
QukgspHRg95zpzEVc3rPckT0FfZSqXfH+DufN5TzMn0q1tLojI70A6C1AwK0XHG+AsqubKcb/0W1k76Cxk2n/SK
eTMgulD3YZ1FAOv2fI8dm9IMz/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/4qdxJ3KmTp0J6HebLJeSOphFr0
y54J7O0GAAYm2KNGFUxgaehvuR2ai8XRUuPKse0cHSRBBi3e8qQcSEHBREa3UZguFyKYe95iWqFPw3e5Rc1hEtw
X4BwNbrWjEjkaI0ZwxEq+OyeLrSWJFrYi6DCLFzsLeF/l9uOHRAeba84/ruSNdg5X//qnso1J1/SXhOxWxd/uYb
U78k7YTeBL8wUW0Pat67QkBG0dL9+lW/050PPCDyfNIbk1JeQ8EhpwA0GStbW78LYJ21YWb4L0zYGIKvfAd5gG3
DE7YxtzQ==="
}
```

3.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

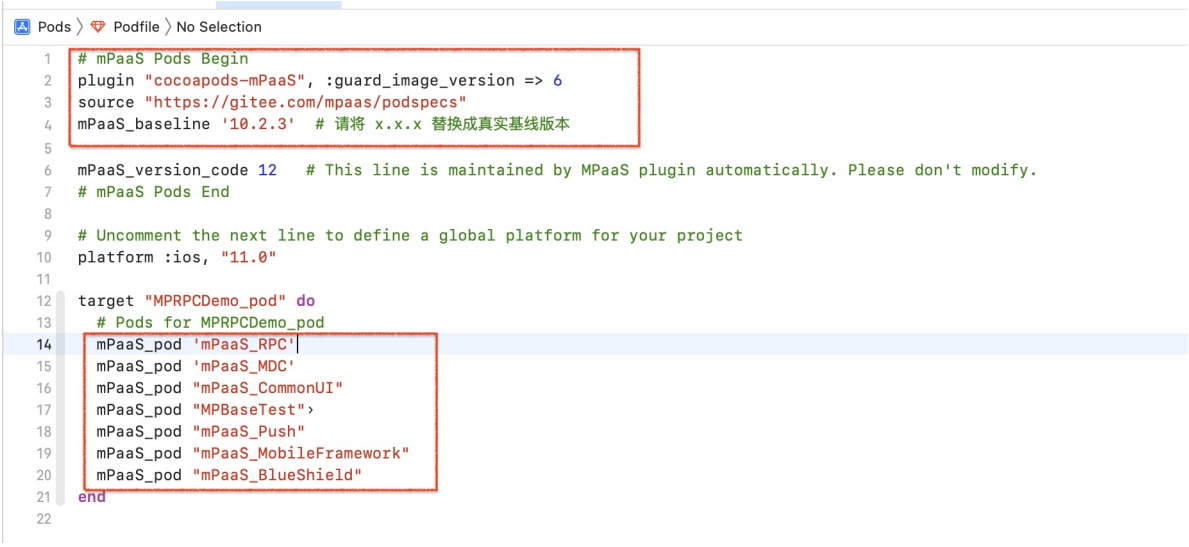
1. 在命令行执行 `pod mpaas update x.x.x` ，其中 `x.x.x` 为当前使用的基线版本号，例如 `10.2.3` 。

```
[➜  ~ 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。

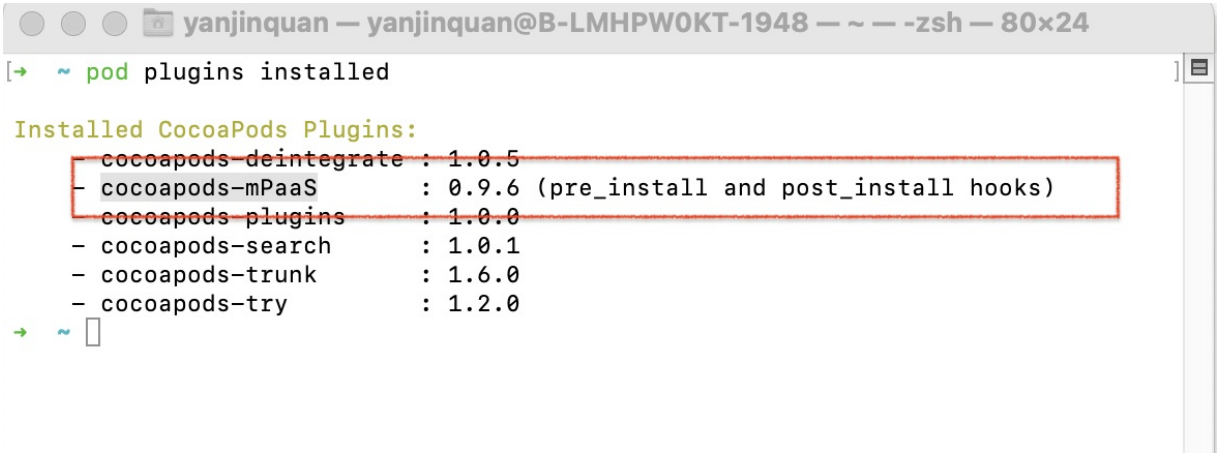
解决方案

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。

4.进阶指南

4.1. 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。

4.2. 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:(NSDictionary *)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DTFrameworkCallbackResultReturnNO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 handleDidFinishLaunchingWithOptions:(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:(NSDictionary *)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UIApplicationDidReceiveRemoteNotification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
didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(
UIBackgroundFetchResult result))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
UIApplicationDidReceiveLocalNotification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
didReceiveLocalNotification:(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
UIApplicationDidReceiveLocalNotification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
handleActionWithIdentifier:(NSString *)identifier forLocalNotification:
(UILocalNotification *)notification completionHandler:(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
UIApplicationDidRegisterForRemoteNotifications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
didRegisterForRemoteNotificationsWithDeviceToken:(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
didFailToRegisterForRemoteNotificationsWithError:(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DT
FrameworkCallbackResultContinue，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)annotation;

/**
 *  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shouldA
utoactivateShareKit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 *)applic
ation;

/**
 *  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 *re
plyInfo))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co
mpletionHandler()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:(voi
d (^)(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

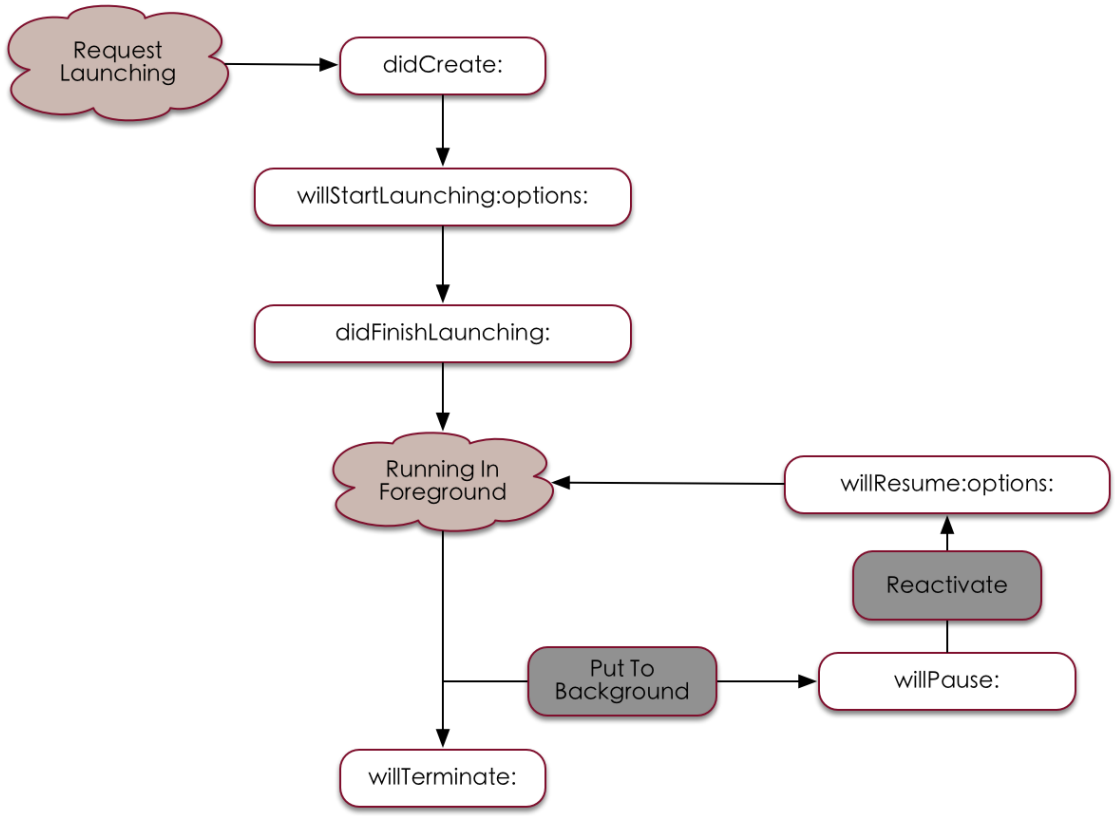
中文	英文	解释
框架上下文	Context	客户端微组件运行期上下文

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:(NS
Dictionary *)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：

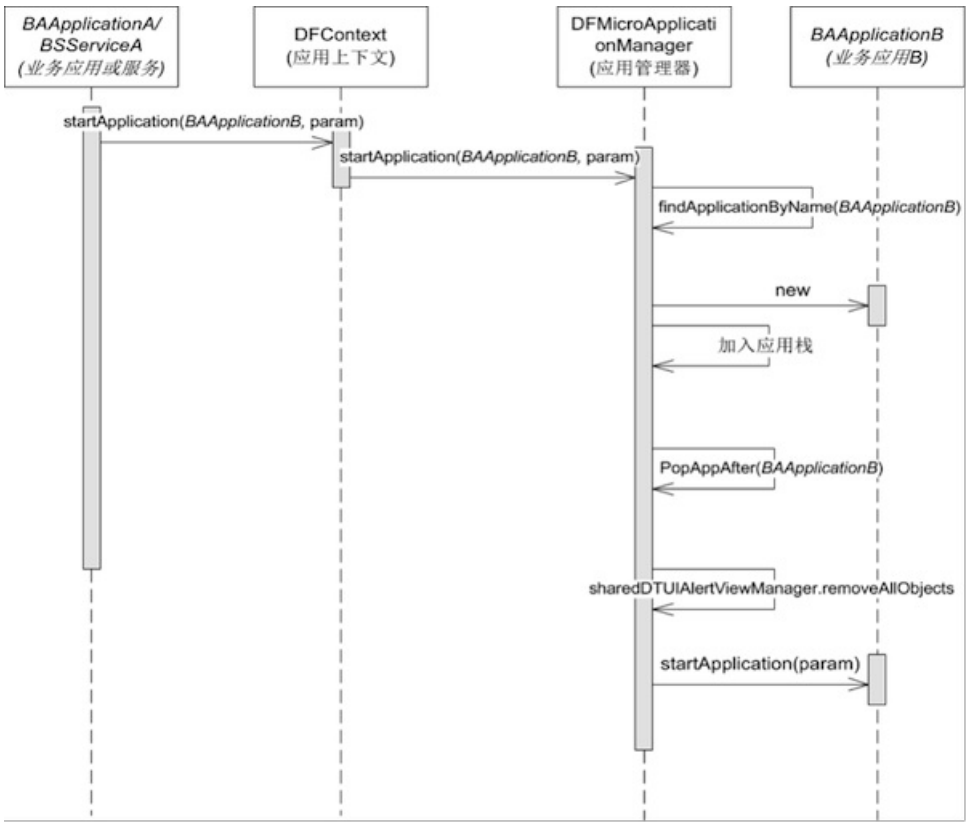
- 提供启动微应用的接口，可通过名字快速查找、关闭、管理微应用的跳转等。
- 提供启动服务的接口，管理服务的注册、发现和反注册。

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)
animated;

/**
 * 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:(DTM
icroApplicationLaunchMode)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：

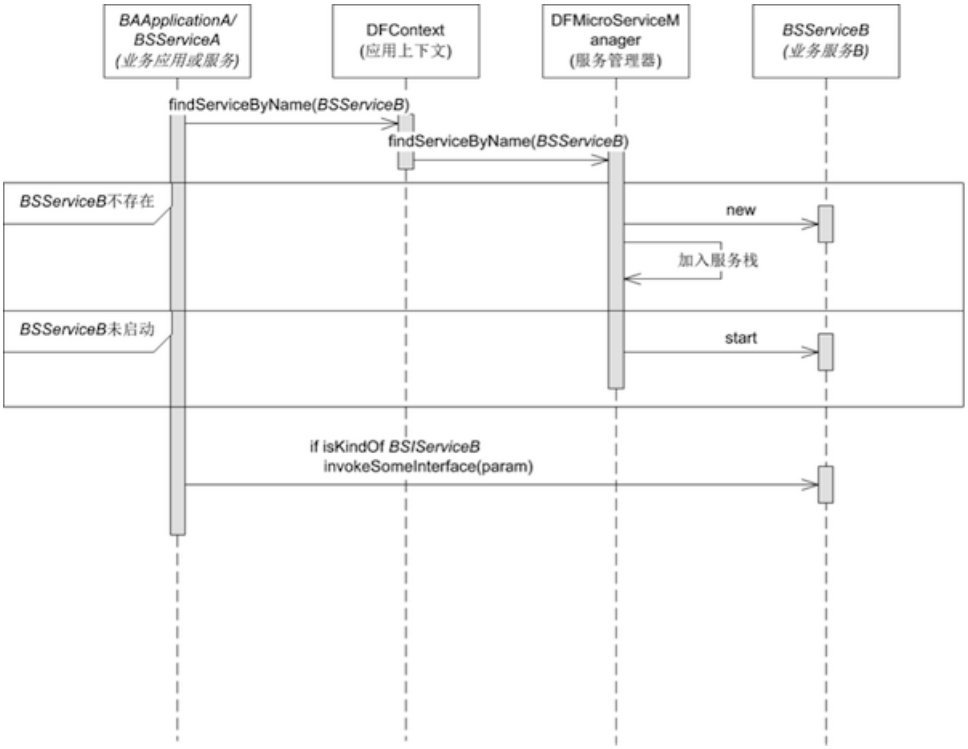


服务管理

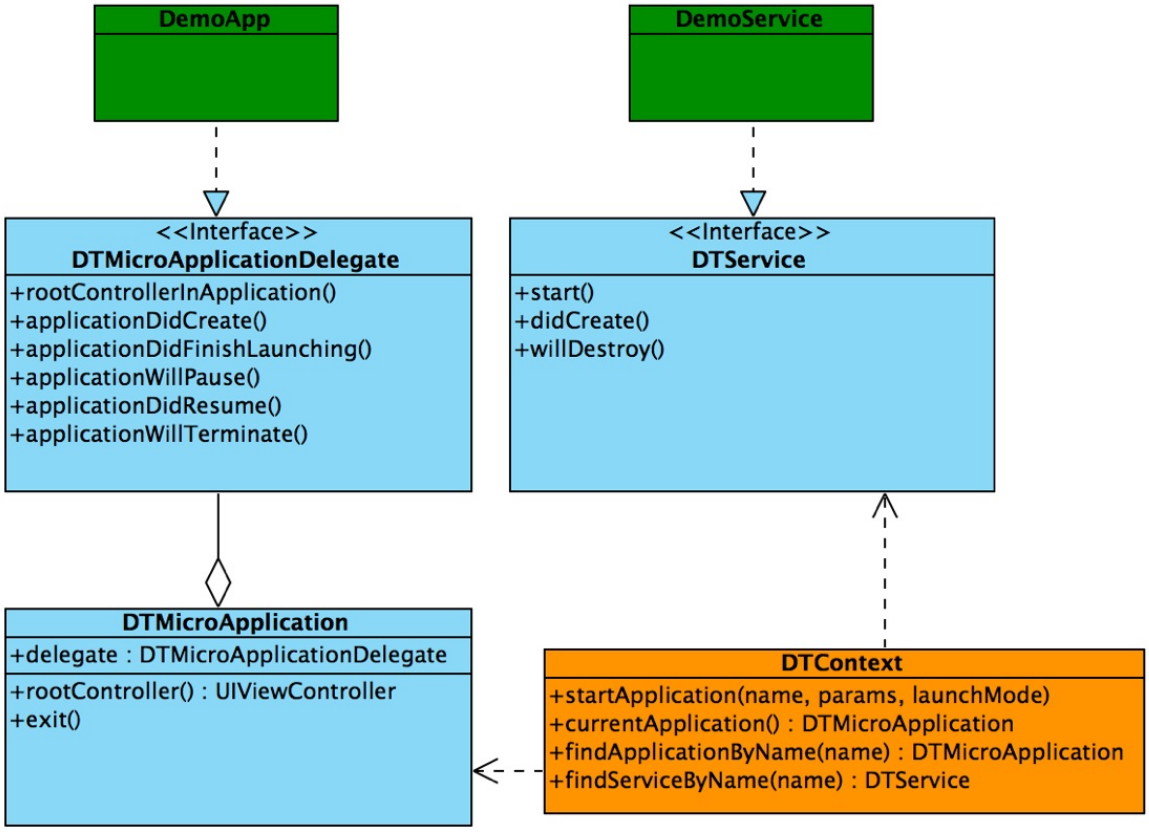
- 服务管理相关接口，具体内容参考 `DTContext.h` 文件。

```
/**
 * 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：



4.3. 微应用与服务

4.3.1. 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。

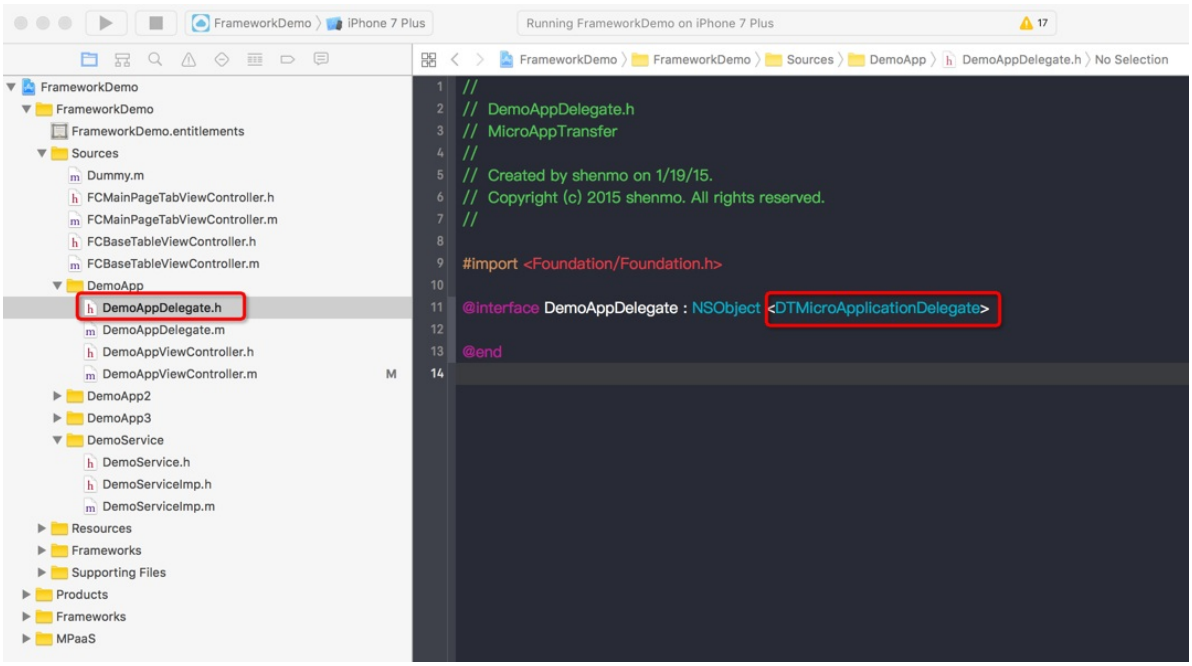
代码示例

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

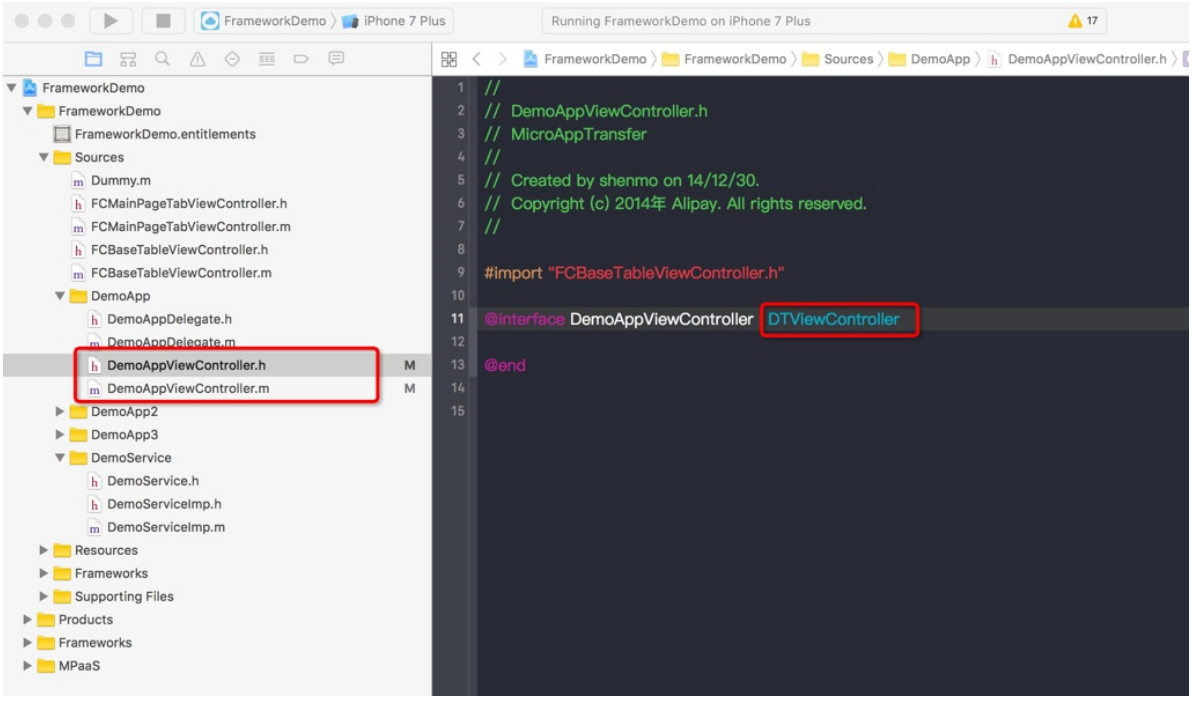
操作步骤

1. 添加微应用模板代码

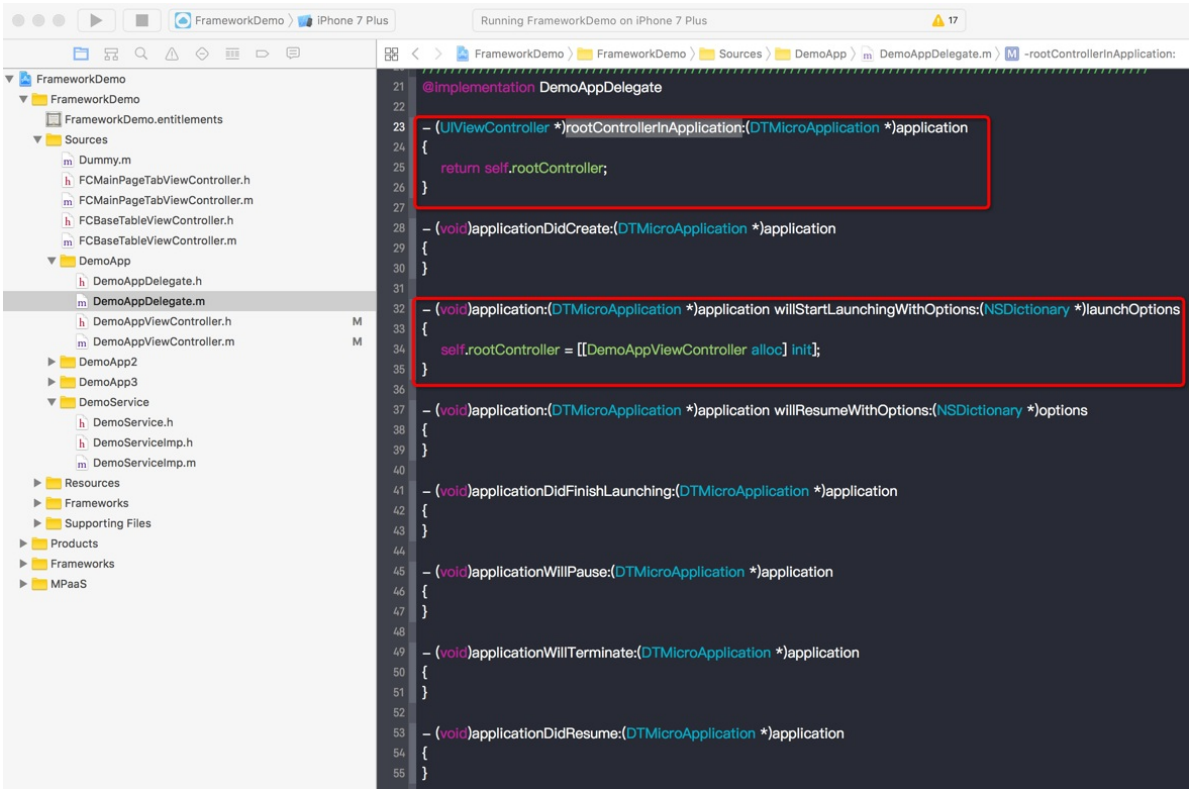
1. 创建微应用的代理类并实现 mPaaS iOS 框架的微应用管理器 `DTMicroApplicationDelegate` 的代理方法。



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

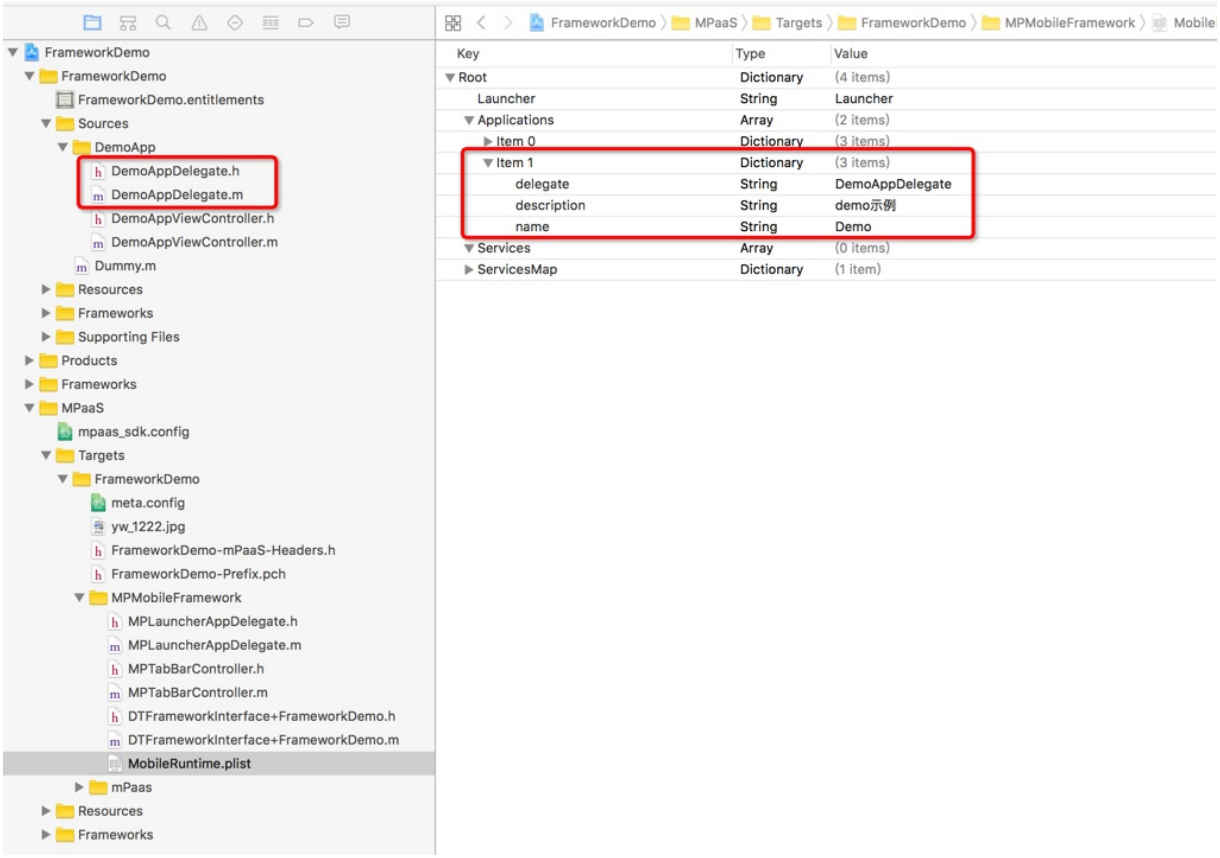


3. 指定微应用的 `rootViewcontroller`。可通过微应用的代理方法，在微应用的生命周期中进行相关的业务处理。



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。

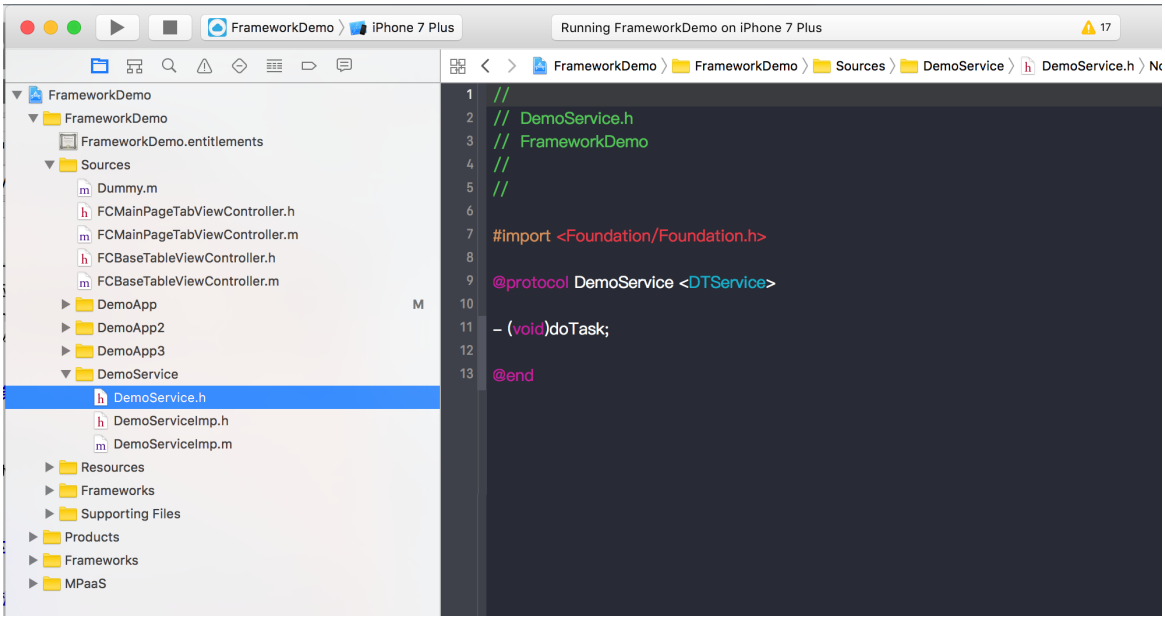
4.3.2. 创建服务

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。

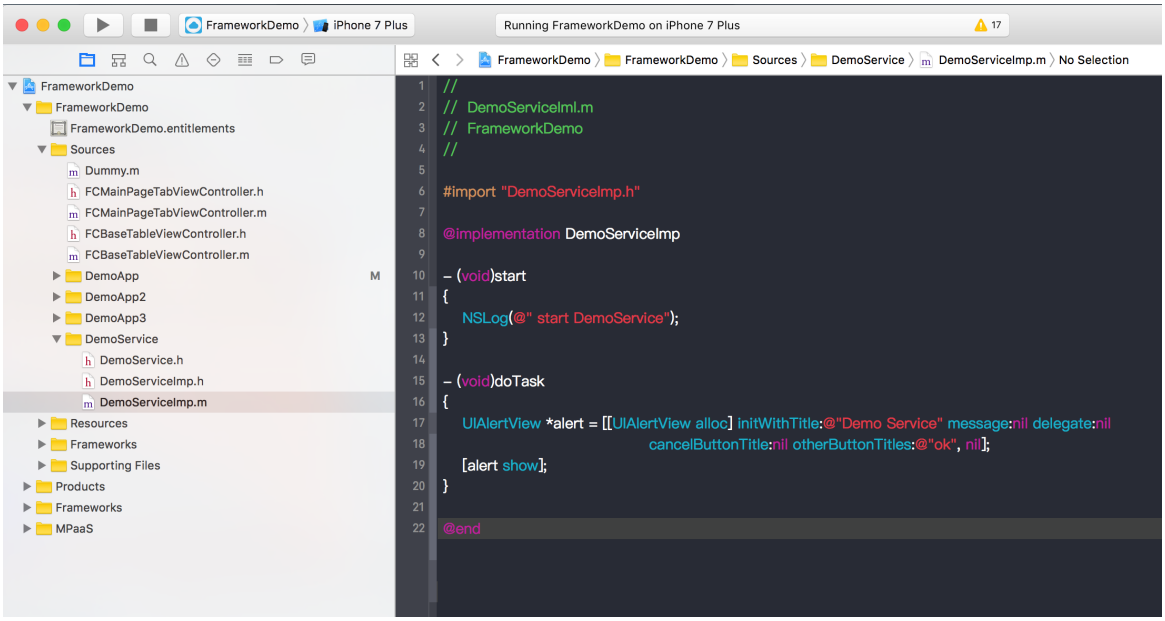
操作步骤

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。

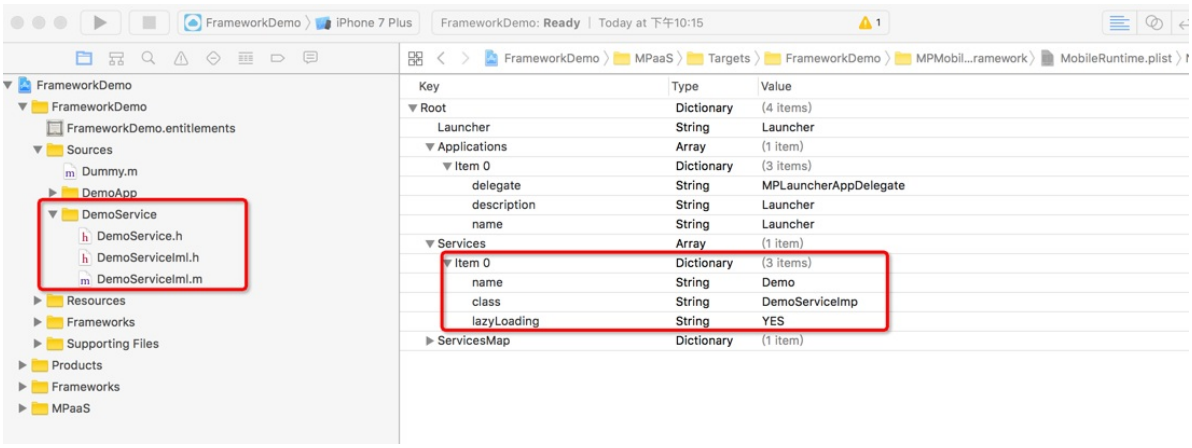


ii. 定义实现服务接口方法的类。



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。



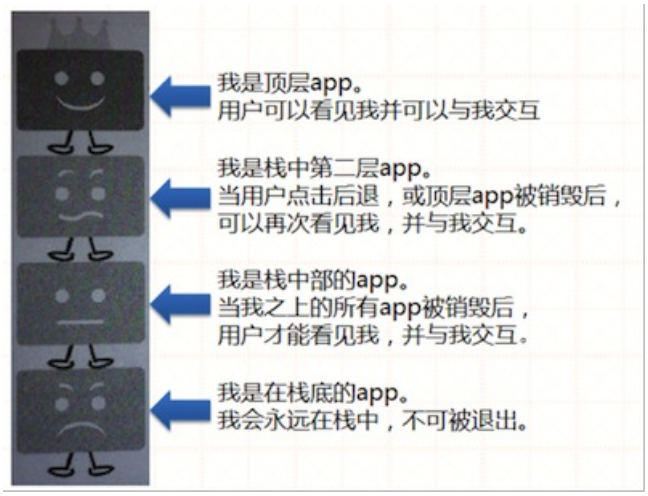
字段	说明
name	服务的唯一标识。
class	服务的实现类，框架在创建该服务时，会利用运行时的反射机制，创建服务实现类的实例。
lazyLoading	是否延迟加载。如果是延迟加载，在框架启动时，该服务不会被实例化，只有在用到时才会实例化并启动。如果是非延迟加载，在框架启动时会实例化并启动该服务。默认为 `NO` 。

4.3.3. 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:kDTMicroApplicationLaunchModePushWithAnimation];
}
```

- 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:kDTMicroApplicationLaunchModePushNoAnimation];
}
```

- 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];
}
```

4.3.4. 微应用层级代码示例

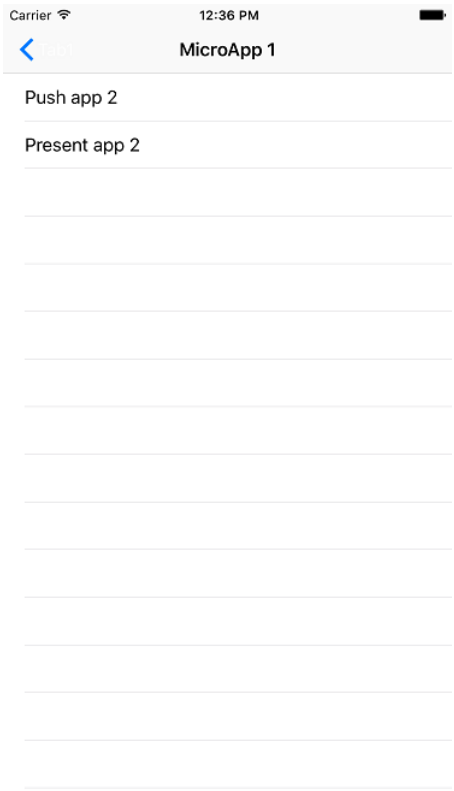
本代码示例介绍 mPaaS 微应用之间的层级关系。有关 iOS 框架的详细介绍，查看 mPaaS 框架介绍。

下载代码

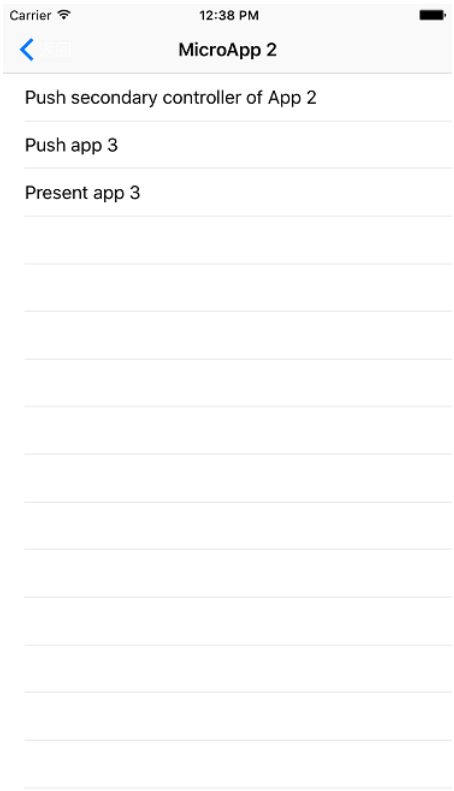
请参考 获取代码示例 下载示例代码。微应用层级演示工程为 `mpaas_demo_ios/FrameworkDemo` 。

微应用层级演示

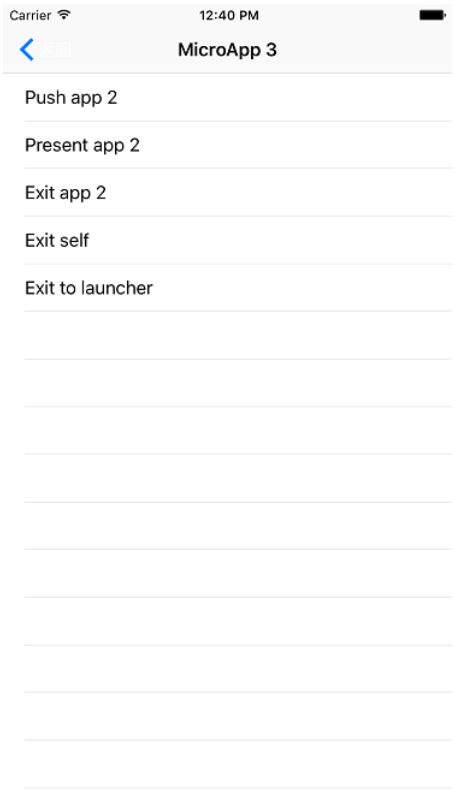
- 点击 **Push app**，启动 MicroApp 1；



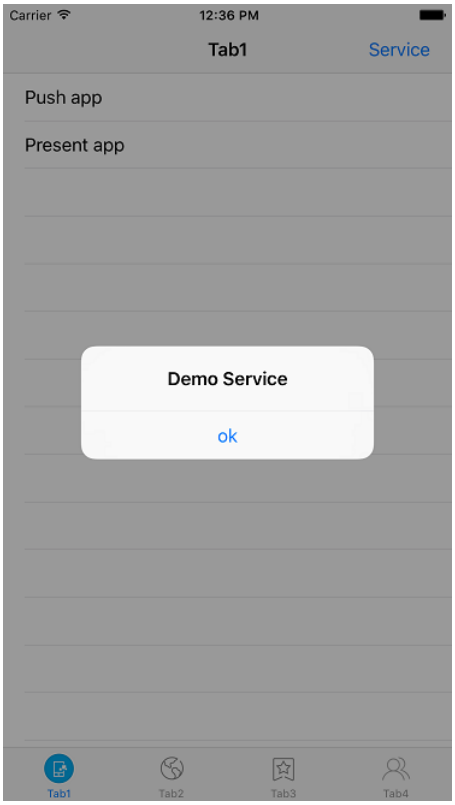
- 点击 **Push app 2**，启动 MicroApp 2；



- 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。



4.4. 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。

4.5. 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/citiesWithCounty.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/hotCities.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。

5.iOS 适配说明

5.1. 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 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。

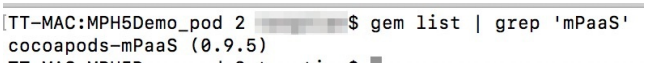
后续步骤

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
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/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 容器和小程序组件，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。

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

> **说明**
>
> - 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

10.1.68 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。

5.2. 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

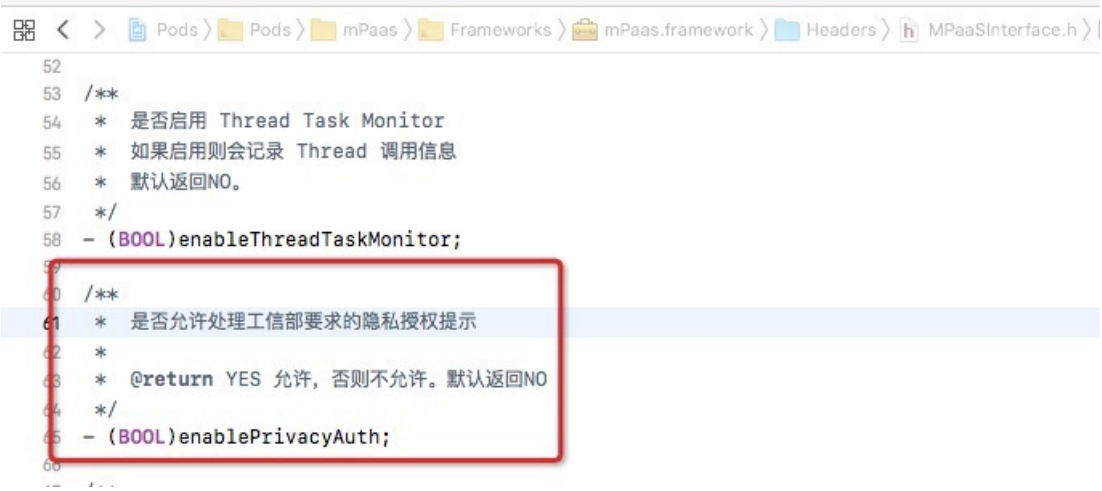
根据是否让 mPaaS iOS 框架托管 App 的生命周期，需要采用不同的使用方法。通过查看工程 `main.m` 文件中是否启用了框架的 `DFApplication` 和 `DFClientDelegate`，可以判断是否让 mPaaS iOS 框架托管了 App 的生命周期；启用了框架的 `DFApplication` 和 `DFClientDelegate` 即表示进行了托管。

```
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:(NS
Dictionary *)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:(NS
Dictionary *)launchOptions
{
  ...
  // 初始化容器 Context
  [MPNebulaAdapterInterface setNBContextWhenEnablePrivacyAuth];

  ...
}
```

5.3. 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 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。

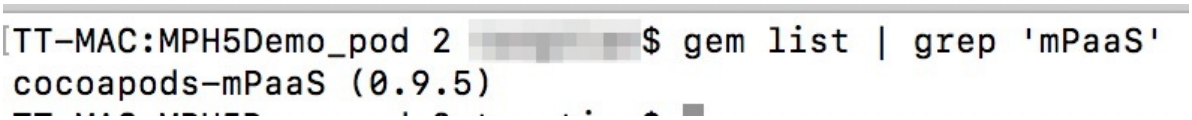
后续步骤

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 & versi
on in 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/installmPaaSCocoaPodsPlugin.sh)
```

组件使用升级指南

10.1.60 基线中的 H5 容器和小程序在接入、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。

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

> **说明**
>
> - 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 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。

5.4. 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:(NS
Dictionary *)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。

5.5. 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 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。

5.6. 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 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。

5.7. 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

5.8. 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 CocoaPods 升级

按照以下步骤，即可安装 **10.1.60** 版本的最新 SDK：

1. 首先确保 Podfile 中 mPaaS 组件的版本号为 **10.1.60**。
2. 执行 `pod mpaas update 10.1.60` 。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

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

> **说明**
>
> - 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

5.9. 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. 使用 **mPaaS 模块编辑** 功能选择您所需的模块。
  ii. 使用 **mPaaS 模块升级** 功能升级至 10.1.32 版本。

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` 。

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

> **说明**
>
> - 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

6.参考

6.1. 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

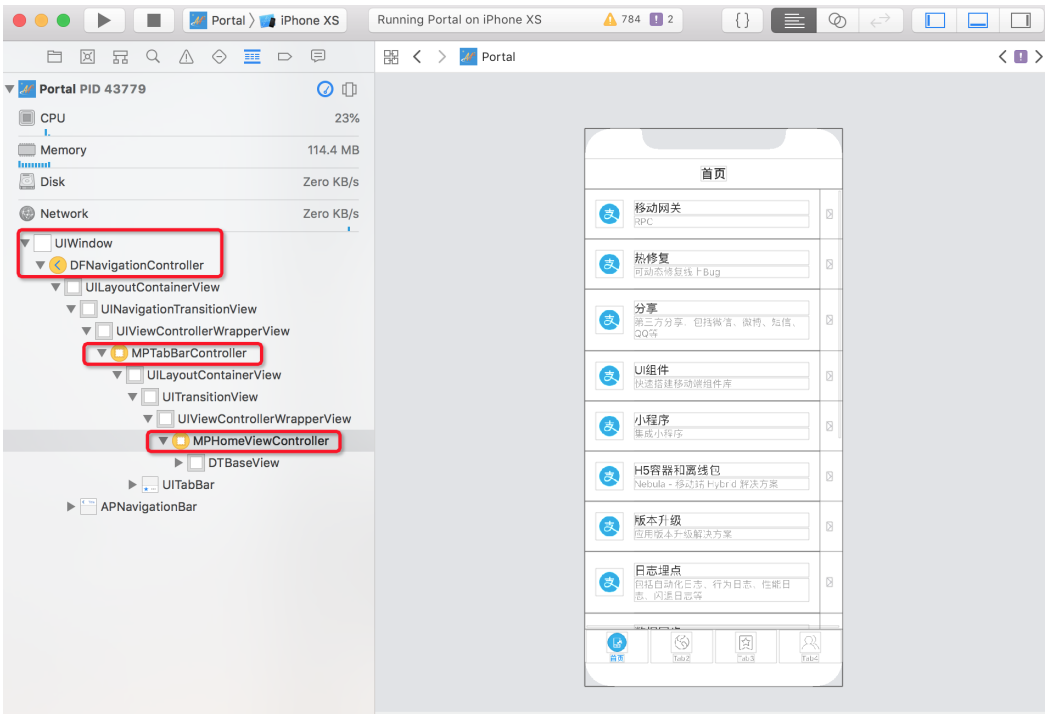
导航栏元素分布

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：

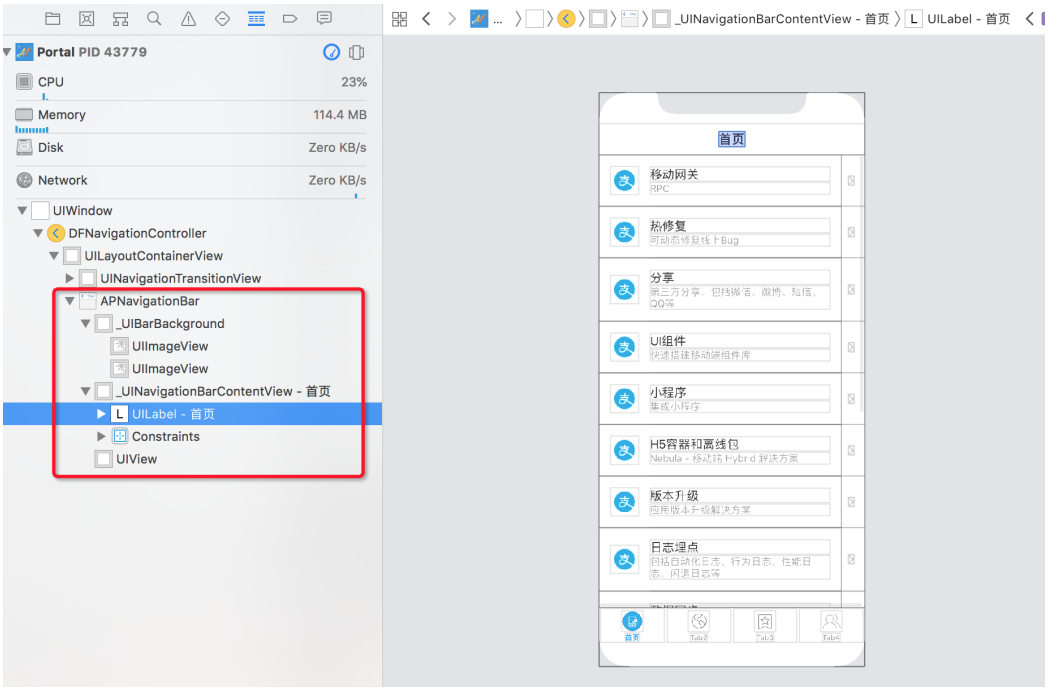
1. back：返回按钮控制区域，由 mPaaS 页面基类创建，默认样式为返回箭头 + 返回文案。
2. title/subTitle：标题栏控制区域，默认不显示。若需显示，请调用系统方法设置当前页面的 title。
3. optionMenu：页面菜单选项区域，默认不显示。如需显示，请调用系统方法设置当前页面的 rightNavigationItem。

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:[UIColo
r whiteColor] translucent:NO];
    [self.navigationController.navigationBar setNavigationBarBottomLineColor:[UICol
or whiteColor]];
}

- (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:
[AUBarButtonItem 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:
[AUBarButtonItem 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(@"ilustration_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:[UICo
lor whiteColor] translucent:NO];
      [self.navigationController.navigationBar setNavigationBarBottomLineColor:[UIC
olor 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:
[AUBarButtonItem class]]) {
              AUBarButtonItem *backItem = leftBarButtonItems[0];
              backItem.titleColor = [UIColor blackColor];
          }
      }
  }

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

6.2. 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。

6.3. 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。

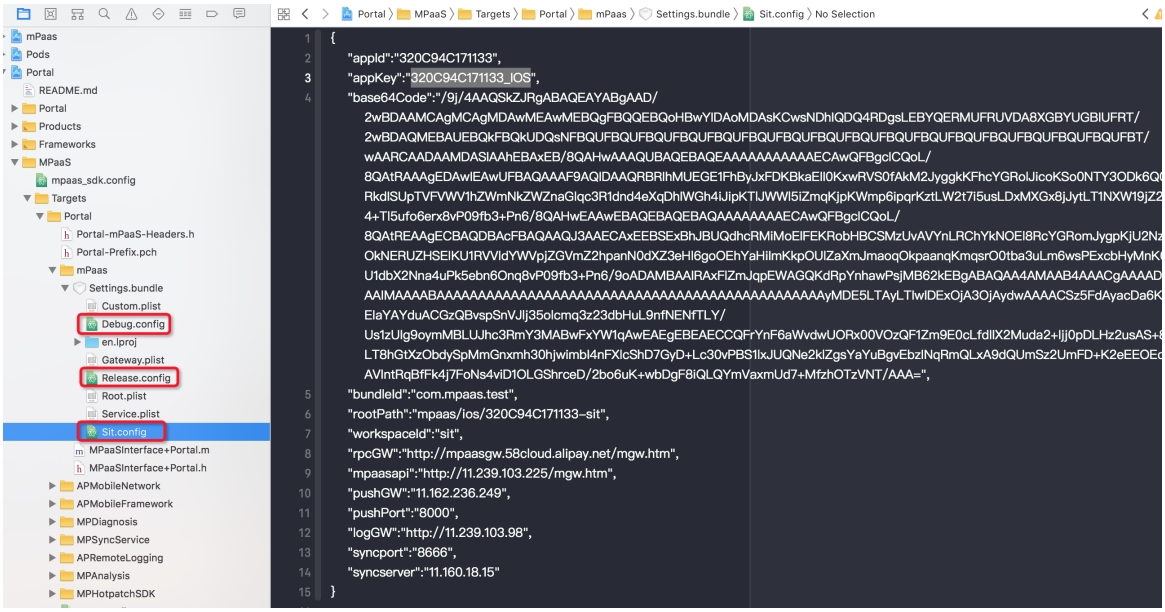
```
![ddd](http://docs-aliyun.cn-hangzhou.oss.aliyun-inc.com/assets/pic/111262/AntCloud_zh/1552905359956/%E5%9B%BE%E7%89%874.png)
```

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。

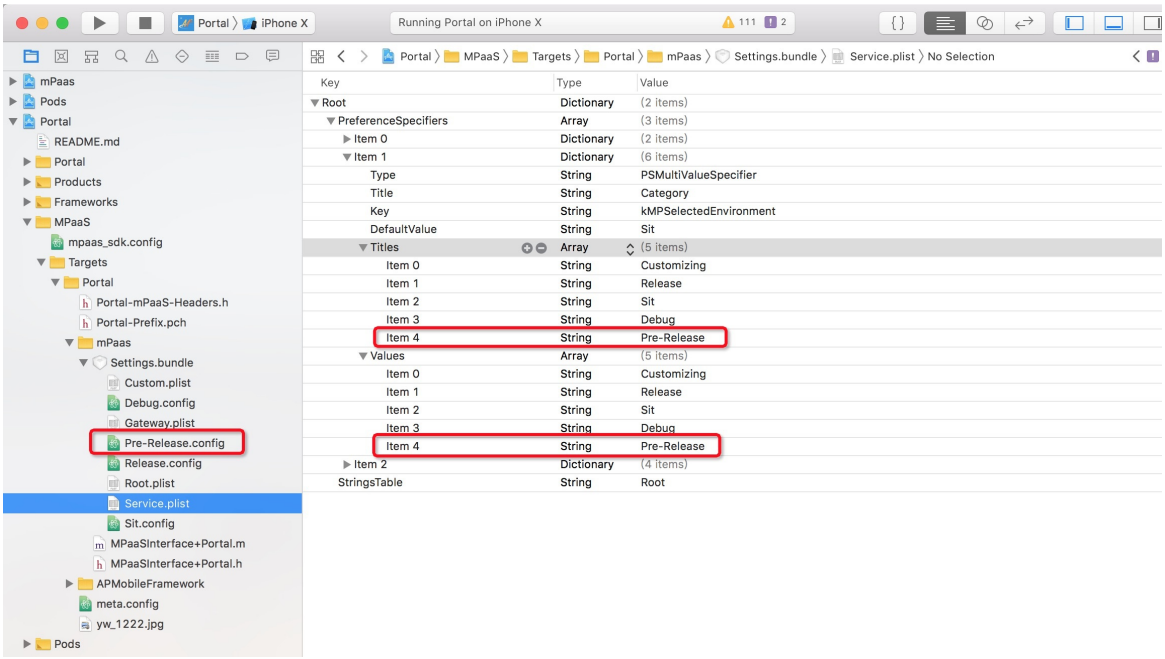
```
@implementation MPaaSInterface (Portal)

- (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。

7.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:kDTMicroApplicationLaunchModePushWithAnimation];
```

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
```