蚂蚁科技

H5 容器和离线包
使用指南

文档版本：20250729

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.H5 容器和离线包

1.1. H5 容器简介

H5 容器是一款移动端 Hybrid 解决方案 SDK（Nebula SDK）。提供了良好的外部扩展功能，拥有功能插件化、事件机制、JSAPI 定制和 H5App 推送更新管理能力。

功能

H5 容器组件提供以下功能：

- 加载 H5 页面，并按照会话（Session）的概念管理各个页面。
- 提供丰富的内置 JSAPI，实现如页面推送（push）或页面弹窗（pop）、标题设置等功能。
- 支持用户自定义 JSAPI 和插件功能，扩展业务需求。
- 接入实时发布（Mobile Delivery Service，MDS）平台后，方便管理离线包。
- Android 使用 UCWebView，拥有解决系统级 WebView Crash 的能力，内存管理更合理， 网络加载提升更快，兼容性更好。彻底告别了在 Android 下兼容不同 WebView 的问题。

特点

H5 容器在以下方面有着出众而强大的特点。

优异的稳定性

- 经过亿级用户考验，崩溃率、ANR 率（Application Not Responding，应用程序无响应）以及其他稳定性指标有保障。
- Android 平台基于 UCWebView 深度定制，崩溃率和 ANR 率远低于系统 WebView，拥有解决系统 WebView 问题的能力。

强大的离线包能力

- **强大的离线包统一推包平台**：通过 MDS 推包平台，能够迅速将离线包推送到客户端，保证客户端数据在短时间能够获得最新同步。
- **预置离线包**：针对特殊场景，支持在客户端预置，提高打开效率。

广泛的生态基础

支持蚂蚁内部接入，并接入了全部的蚂蚁 App，拥有稳定的生态基础。

1.2. 离线包简介

传统的 H5 技术容易受到网络环境影响，因而降低 H5 页面的性能。通过使用离线包，可以解决该问题，同时保留 H5 的优点。

离线包 是将包括 HTML、JavaScript、CSS 等页面内静态资源打包到一个压缩包内。预先下载该离线包到本地，然后通过客户端打开，直接从本地加载离线包，从而最大程度地摆脱网络环境对 H5 页面的影响。

使用 H5 离线包可以给您带来以下优势：

- **提升用户体验**：通过离线包的方式把页面内静态资源嵌入到应用中并发布，当用户第一次开启应用的时候，就无需依赖网络环境下载该资源，而是马上开始使用该应用。
- **实现动态更新**：在推出新版本或是紧急发布的时候，您可以把修改的资源放入离线包，通过更新配置让应用自动下载更新。因此，您无需通过应用商店审核，就能让用户及早接收更新。

离线包原理

您将从以下方面了解离线包原理：

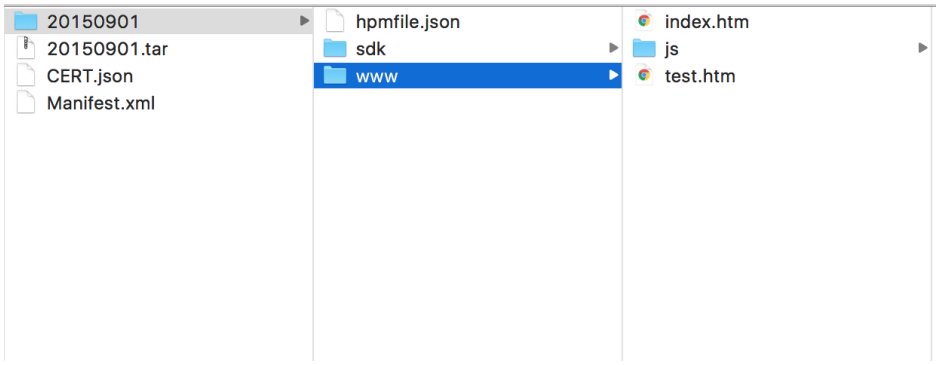
- 离线包结构
- 离线包类型
- 渲染过程

离线包结构

离线包是一个 `.amr` 格式的压缩文件，将后缀 `amr` 改成 `zip` 解压缩后，可以看到其中包含了 HTML 资源和 JavaScript 代码等。待 H5 容器加载后，这些资源和代码能在 WebView 内渲染。

以 iOS 系统为例，下图显示了一般资源包的目录结构：

- 一级目录：一般资源包的 ID，如 `20150901`。
- 二级目录及子目录即为业务自定义的资源文件。建议所有的前端文件最好保存在一个统一的目录下，目录名可自定义，如 `/www`，并设定当前离线包默认打开的主入口文件，如 `/www/index.html`。



离线包类型

通常，在 H5 的开发过程中，会存在使用一些基础通用库的情况，比如 zepto，fastclick 等。在 App 中的 WebView，有时候缓存不可靠，曾经发现有机型在退出后，缓存自动失效。为了进一步提升 H5 页面性能，使用全局离线包，将一系列的通用资源打成一个特殊的 App 包，下发到客户端。

离线包可以分为以下类型：

- **全局离线包**：包含公共的资源，可供多个应用共同使用。
- **私有离线包**：只可以被某个应用单独使用。

使用全局离线包后，在访问 H5 的时候，都会尝试在这个包尝试读取。如果该离线包里有对应资源的时候，直接从该离线包里取，而不通过网络。因此，全局离线包的机制主要是为了解决对于通用库的使用。

由于要保证离线包的客户端覆盖率以及足够的通用性，此包一般的更新周期至少为 1 个月，并且严格控制离线包的大小。

渲染过程

当 H5 容器发出资源请求时，其访问本地资源或线上资源所使用的 URL 是一致的。

H5 容器会先截获该请求，截获请求后，发生如下情况：

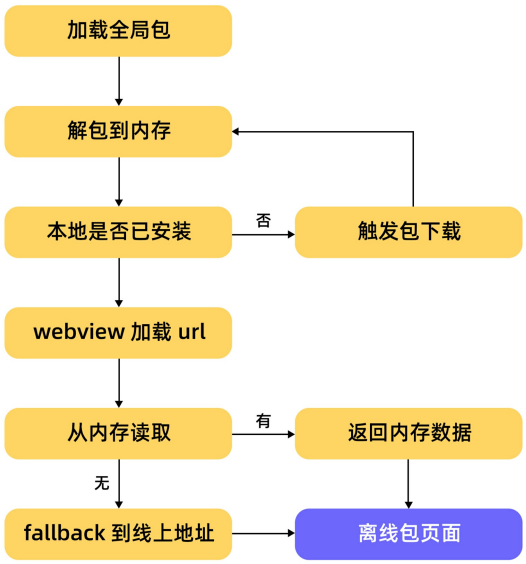
- 如果本地有资源可以满足该请求的话，H5 容器会使用本地资源。
- 如果没有可以满足请求的本地资源，H5 容器会使用线上资源。

因此，无论资源是在本地或者是线上，WebView 都是无感知的。

离线包的下载取决于创建离线包时的配置：

- 如果 **下载时机** 配置为 **仅 WiFi**，则只有在 WiFi 网络时会在后台自动下载离线包。
- 如果 **下载时机** 配置为 **所有网络都下载**，则在非 WiFi 网络时会消耗用户流量自动下载，请谨慎使用。

- 如果当前用户点击 App 时，离线包尚未下载完毕，则会跳转至 fallback 地址，显示在线页面。**fallback** 技术用于应对离线包未下载完毕的场景。每个离线包发布时，都会同步在 CDN 发布一个对应的线上版本，目录结构和离线包结构一致。fallback 地址会随离线包信息下发到本地。在离线包未下载完毕的场景下，客户端会拦截页面请求，转向对应的 CDN 地址，实现在线页面和离线页面随时切换。



离线包运行模式

要打开离线包，您需要完成以下步骤：

1. 请求包信息：从服务端请求离线包信息存储到本地数据库的过程。离线包信息包括离线包的下载地址、离线包版本号等。
2. 下载离线包：把离线包从服务端下载到手机。
3. 安装离线包：下载目录，拷贝到手机安装目录。

虚拟域名

虚拟域名是容器的独特机制，仅对离线应用有效。当页面保存在客户端之后，WebView 是通过 file Schema 从本地加载访问的。然而，用户就能在地址栏里直接看到 file 的路径，这就会导致以下问题：

- 用户体验问题：当用户看到了 file 地址，会对暴露的地址产生不安全感和不自在。
- 安全性问题：由于 file 协议上直接带上了本地路径，任何用户都可以看到这个文件所在的文件路径，会存在一定的安全隐患。

> **说明**
>
> 基于如上问题的考虑，推荐采用虚拟域名的机制而不直接使用 file 路径来访问。虚拟域名是一个符合 URL Scheme 规范的 HTTPS 域名地址，例如 `https://xxxxxxx.h5app.example.com`，虚拟域名的 **父域名 `example.com` 一定得使用自己注册的域名。**

现在标准的虚拟域名的格式如下：

```
https://{appid}.h5app.example.com。
```

相关链接

生成离线包

1.3. 接入 Android

1.3.1. SDK 升级说明

本文是关于 10.1.60 版 SDK 升级的变更说明。

MPNebula

```
public static void enableAppVerification(final String publicKey)
```

上述 API 已修改，除调用此 API 之外，仍需要开启验签开关，参见 H5 容器配置。

```
public static void disableAppVerification();
```

上述 API 已废弃，不再支持使用此方法关闭验签，如需关闭验签，参见 H5 容器配置。

H5ExtConfigProvider

已废弃，如需配置 H5 容器开关，参见 H5 容器配置。

1.3.2. 快速开始

H5 容器和离线包支持 **原生 AAR 接入** 和 **组件化接入** 两种接入方式。通过使用 H5 容器可以实现在应用内打开一个在线网页、前端调用 Native 接口、前端调用自定义 JSAPI、自定义 H5 页面的 TitleBar、使用 UC 内核等相关功能。使用 H5 离线包可以实现发布、预置、启动和更新离线包等相关功能。

前置条件

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

添加 SDK

原生 AAR 方式

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

组件化方式

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

初始化 mPaaS

如果使用 **原生 AAR 接入**，需要初始化 mPaaS。在 Application 中添加以下代码：

```
public class MyApplication extends Application {

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

详情请参考：初始化 mPaaS。

配置小程序包请求时间间隔

mPaaS 支持配置小程序包的请求时间间隔，可全局配置或单个配置。

- **全局配置**：在 Android 工程的 `assets/config` 路径下，创建 `custom_config.json` 文件，并在文件内填入以下内容：

  ```
  {
  "value": "{\"config\":{\"al\":\"3\",\"pr\":
  {\"4\":\"86400\",\"common\":\"864000\"},\"ur\":\"1800\",\"fpr\":
  {\"common\":\"3888000\"}},\"switch\":\"yes\"}",
  "key": "h5_nbmngconfig"\
  }
  ```

  其中 `\"ur\":\"1800\"` 是设置全局更新间隔的值，`1800` 为默认值，代表间隔时长，单位为秒，您可修改此值来设置您的全局离线包请求间隔，范围为 0 ~ 86400 秒（即 0 ~ 24 小时，0 代表无请求间隔限制）。

> **重要**
> 其他参数请勿随意修改。

- **单个配置**：即只对当前小程序包配置。可在控制台中前往 **添加离线包** > **扩展信息** 中填入 `{"asyncReqRate":"1800"}` 来设置请求时间间隔。详情参见 创建 H5 离线包 中的 **扩展信息**。

验证请求时间间隔配置是否生效：您可以打开一个接入 H5 离线包的工程，在 logcat 日志中过滤 `H5BaseAppProvider` 关键字，若能看到如下信息，则说明配置已经生效。

```
lastUpdateTime: xxx updateRate: xxx
```

使用 SDK

mPaaS Nebula H5 容器提供统一的接口类 `MPNebula` 来实现 H5 容器及离线包的操作。调用过程如下：

1. 启动 H5 离线包。
   - 启动一个离线包：

```
/**
* 启动离线包
*
* @param appId 离线包 ID
*/
public static void startApp(String appId);
```

   - 启动一个离线包（带启动参数）：

```
/**
* 启动离线包
*
* @param appId  离线包 ID
* @param params 启动参数
*/
public static void startApp(String appId, Bundle params);
```

2. 启动一个在线页面。
   - 启动一个在线页面：

```
/**
* 启动在线 URL
*
* @param url 在线地址
*/
public static void startUrl(String url)
```

   - 启动一个在线页面（带启动参数）：

```
/**
* 启动在线 URL
*
* @param url   在线地址
* @param param 启动参数
*/
public static void startUrl(String url, Bundle param);
```

3. 设置自定义 `UserAgent`。

i. 首先需要实现一个 UA 设置器，如下所示：

```
public class H5UaProviderImpl implements H5UaProvider {
@Override
public String getUa(String defaultUaStr) {
 //不要修改 defaultUaStr，或者返回一个不包含 defaultUaStr 的结果
 return defaultUaStr + " AlipayClient/mPaaS";
}
}
```

ii. 然后通过调用设置 UA 接口：

```
/**
* 设置 UA
*
* @param uaProvider UA 设置器，需开发者实现 getUa 方法
*/
public static void setUa(H5UaProvider uaProvider)
```

iii. 进行设置：

```
MPNebula.setUa(new H5UaProviderImpl());
```

4. 设置自定义容器 View。

容器提供方法可以设置自定义的标题栏、导航菜单、下拉刷新头部以及 WebView 的承载布局，具体实现，可参考 获取代码示例，查看 AAR 接入方式下 H5 容器和离线包的代码示例。

i. 首先，需要先实现一个 View 设置器，如下所示：

```
public class H5ViewProviderImpl implements H5ViewProvider {
@Override
public H5WebContentView createWebContentView(Context context) {
 // 此处返回自定义的 WebView 的承载布局，若返回 null 则使用默认 View
 return null;
}

@Override
public H5TitleView createTitleView(Context context) {
 // 此处返回自定义的标题栏，若返回 null 则使用默认 View
 return null;
}

@Override
public H5PullHeaderView createPullHeaderView(Context context, ViewGroup viewGroup) {
 // 此处返回自定义的下拉刷新头部，若返回 null 则使用默认 View
 return null;
}

@Override
public H5NavMenuView createNavMenu() {
 // 此处返回自定义的导航菜单，若返回 null 则使用默认 View
 return null;
}
}
```

ii. 然后通过 View 设置接口：

```
/**
* 设置容器相关的自定义 view，如标题栏、菜单栏、web layout、下拉刷新 view 等等
* @param viewProvider 自定义 view provider
*/
public static void setCustomViewProvider(H5ViewProvider viewProvider);
```

iii. 执行设置过程：

```
MPNebula.setCustomViewProvider(new H5ViewProviderImpl());
```

> **说明**
> 若要设置自定义标题栏，需要先设置 bundle name，否则可能会找不到资源。

```
// 此处必须设置标题栏资源所在的 bundle 名称，如果不设置，会导致资源无法加载导致标题栏无法生效
H5Utils.setProvider(H5ReplaceResourceProvider.class.getName(), new H5ReplaceResourceProvider() {
@Override
public String getReplaceResourcesBundleName() {
 return BuildConfig.BUNDLE_NAME;
}
});
MPNebula.setCustomViewProvider(new H5ViewProviderImpl());
```

5. 将单个容器的视图嵌入到页面。

   根据实际情况选择以下方法将 H5 页面嵌入到单个容器的视图（view）中，接口提供同步方法和异步方法。

   - **同步方法**

```
/**
* 获取 H5 容器的视图（view）
*
* @param activity 页面上下文
* @param param    启动参数，内部可包含 appid 或是 URL
* @return H5 容器的视图（view）
*/
public static View getH5View(Activity activity, Bundle param);
```

   - **异步方法**

```
/**
* 异步获取 H5 容器的视图（view）
*
* @param activity            页面上下文
* @param param               启动参数，内部可包含 appid 或是 URL
* @param h5PageReadyListener 异步回调
*/
public static void getH5ViewAsync(Activity activity, Bundle param, H5PageReadyListener h5PageReadyL
istener);
```

> **说明**
> - 在使用上述同步、异步方法嵌入容器时，需要提前获取对应离线包的信息。
> - 使用异步方法不占用主线程，不会影响性能。

1.3.3. 管理离线包

离线包的管理操作包括：预置 H5 应用、利用全局资源包、更新 H5 应用、下载 H5 应用、安装 H5 应用、获取应用信息、校验安全签名以及删除本地应用。

前置条件

- 您已完成接入配置。具体的操作步骤，查看 添加 SDK。
- 您已经生成离线包。具体的操作步骤，查看 生成离线包。

预置 H5 应用

通常情况下，第一次打开 H5 应用时，离线包可能未完成下载。此时，需要通过使用 fallback URL 的方式打开应用。

预置 H5 应用相当于在客户端发布的安装包中预先安装可用的 H5 应用。当用户第一次打开预安装应用时，可直接使用离线包资源，提高用户体验。

> **说明**
> 建议您只预装核心 H5 应用，避免预置了使用率不高的应用。

预置 H5 应用需要完成以下步骤：

1. 从 H5 应用发布后台下载 H5 应用配置的 `.json` 文件以及需要预置的离线包。
2. 将 `.json` 文件和离线包添加到工程的 `asset` 目录下。
3. 在应用启动时，调用预置代码安装应用，示例代码如下：

```
MPNebula.loadOfflineNebula("h5_json.json", new MPNebulaOfflineInfo("90000000_1.0.0.6.amr", "90000000"
, "1.0.0.6"));
```

> **说明**
> - 此方法为阻塞调用，请不要在主线程上调用内置离线包方法。
> - 此方法仅能调用一次，若多次调用，仅第一次调用有效。所以需要一次性传入所有需预置的离线包信息。
> - 如果内置多个 amr 包，要确保文件已存在，如不存在，会造成其他内置离线包失败。

利用全局资源包

Nebula 全局资源包解决多个 H5 应用使用同一资源产生的冗余问题。如 React 应用使用 ReactJS 框架代码。您可以将公共资源放入全局资源包，以降低 H5 应用体积。

通常情况下，项目需预置全局资源包，后续更新依然可以通过 H5 应用后台下发。

下方的示例代码指定应用 ID（`appId`）为 `66666692` 的离线包作为全局资源包使用，并预置 `assets/nebulaPreset/66666692` 离线包，其中：

- `getCommonResourceAppList`：用于告知 H5 容器指定 ID 的离线包将作为全局资源包使用。如果您没有配置此 ID，即使内置该离线包，也不会生效。
- `getH5PresetPkg`：用于指定内置全局资源包的路径和版本。H5 容器会从指定的 asset 资源目录加载资源包。不过如果服务端发现更高的版本，该低版本内置包将不会被加载。另外，您也可以使用上文提及的 `loadOffLineNebula` 方法来预置全局资源包，此种情况下，您无需在此方法中配置内置离线包的路径和版本。

> **说明**
> 该方法只适用于 H5 全局资源包。

- `getTinyCommonApp`：仅用于返回小程序的框架资源包 ID，如果您的全局资源包仅被 H5 使用，请不要在此方法中写入该公共资源包 ID。

参考实现类示例代码：

```
public class H5AppCenterPresetProviderImpl implements H5AppCenterPresetProvider {
    private static final String TAG = "H5AppCenterPresetProviderImpl";

    // 业务的公共资源包，尽量避开666666开头
    private static final String COMMON_BIZ_APP = "xxxxxxxx";

    // 小程序专用资源包，业务勿动
    private static final String TINY_COMMON_APP = "66666692";

    // 预置包的 asset 目录
    private final static String NEBULA_APPS_PRE_INSTALL = "nebulaPreset" + File.separator;

    // 预置包集合
    private static final Map<String, H5PresetInfo> NEBULA_LOCAL_PACKAGE_APP_IDS = new HashMap();

    static {

        H5PresetInfo h5PresetInfo2 = new H5PresetInfo();
        // 内置目录的文件名称
        h5PresetInfo2.appId = TINY_COMMON_APP;
```

```
        h5PresetInfo2.version = "1.0.0.0";
        h5PresetInfo2.downloadUrl = "";

        NEBULA_LOCAL_PACKAGE_APP_IDS.put(TINY_COMMON_APP, h5PresetInfo2);

    }

    @Override
    public Set<String> getCommonResourceAppList() {
        Set<String> appIdList = new HashSet<String>();
        appIdList.add(getTinyCommonApp());
        appIdList.add(COMMON_BIZ_APP);
        return appIdList;
    }

    @Override
    public H5PresetPkg getH5PresetPkg() {
        H5PresetPkg h5PresetPkg = new H5PresetPkg();
        h5PresetPkg.setPreSetInfo(NEBULA_LOCAL_PACKAGE_APP_IDS);
        h5PresetPkg.setPresetPath(NEBULA_APPS_PRE_INSTALL);
        return h5PresetPkg;
    }

    @Override
    public Set<String> getEnableDegradeApp() {
        return null;
    }

    @Override
    public String getTinyCommonApp() {
        return TINY_COMMON_APP;
    }

    @Override
    public InputStream getPresetAppInfo() {
        return null;
    }

    @Override
    public InputStream getPresetAppInfoObject() {
        return null;
    }
}
```

然后在应用启动时调用：

```
H5Utils.getH5ProviderManager().setProvider(H5AppCenterPresetProvider.class.getName(), new
H5AppCenterPresetProviderImpl());
```

更新 H5 应用

默认情况下，每次打开 H5 应用，Nebula 都会尝试检查是否有可更新的版本。出于服务端压力考虑，该检查有时间间隔限制，默认为 30 分钟。如果想立即检查最新的可用版本，可以调用下方的代码来请求更新。一般情况下，可以在应用启动或者用户登录后调用。

```
MPNebula.updateAllApp(new MpaasNebulaUpdateCallback(){
    @Override
    public void onResult(final boolean success, final boolean isLimit, String detailCode) {
        super.onResult(success, isLimit);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                AUToast.makeToast(NebulaAppActivity.this,
                        success ? R.string.update_success : R.string.update_failure, 2000).show();
            }
        });
    }
});
```

下载 H5 应用

MPNebula 提供了手动下载 H5 应用的接口：

```
    /**
     * 下载离线包
     *
     * @param appId                        离线包 id
     * @param mpaasNebulaDownloadCallback 下载回调
     */
    public static void downloadApp(final String appId, final MpaasNebulaDownloadCallback
mpaasNebulaDownloadCallback)
```

安装 H5 应用

MPNebula 提供了手动安装 H5 应用的接口：

```
    /**
     * 安装离线包
     *
     * @param appId                       离线包 id
     * @param mpaasNebulaInstallCallback 安装回调
     */
    public static void installApp(final String appId, final MpaasNebulaInstallCallback
mpaasNebulaInstallCallback)
```

获取应用信息

调用 H5AppProvider 的方法以获取 H5 应用的相关信息：

```
H5AppProvider provider = H5Utils.getProvider(H5AppProvider.class.getName());
AppInfo appInfo = provider.getAppInfo("10000000");                 //获取应用配置
boolean isInstalled = provider.isInstalled("10000000", "1.0.0.0"); //某版本应用是否已经安装
boolean isAvailable = provider.isAvailable("10000000", "1.0.0.0"); //某版本应用离线包是否已经下载完成
```

校验安全签名

Nebula 具有离线包签名校验机制，以防止恶意程序篡改下载到设备的离线包。通过调用 `MPNebula` 接口设置验签参数即可开启此机制。如果您使用的基线是 10.1.60 或以上版本，需要额外开启容器配置，详情参见 H5 容器配置。

- 请在第一次打开离线包前调用 `MPNebula` 接口，否则将会导致公钥初始化失败。关于公钥与私钥，参见 配置离线包 > 密钥管理。
- 无论客户端是否开启签名校验，在被判断为 root 的手机上都会强制进行签名校验。

```
    /**
     * @param publicKey 验签公钥
     */
    public static void enableAppVerification(final String publicKey)
```

删除本地应用

Nebula 提供了删除本地应用信息的接口。当本地应用信息被删除后，再次打开应用时会重新请求服务端下载更新本地应用信息。

```
public class MPNebula {
    // appId 为离线包或小程序的应用 ID
    public static boolean deleteAppInfo(String appId);
}
```

> 说明
> 此 API 支持的最低基线版本分别为 10.1.68.8 和 10.1.60.14。

1.3.4. 进阶指南

1.3.4.1. 接入 mPaaS 内核

Android mPaaS 基于 Chromium 105 版本实现内核全面升级，解决老版本的遗留问题并提升用户体验，目前处于免费试用阶段。

内核功能说明

- 基于统一内核，能解决 Android 碎片化带来的 H5 页面适配问题，并收敛低版本的系统内核导致的 Crash。
- 小程序场景下，针对 Video、Map 等组件通过同层渲染模式支持将 Native 组件渲染到同一层级上，使得这些高级组件具有原生体验。

接入

> 说明
> 接入 mPaaS 内核要求基线版本大于等于 10.2.3.55。

1. 增加新的 mvn 配置。

```
maven {
    credentials {
        username '617696fc6970513e6e677c83'
        password '2qAa=hHdfLOp'
    }
    url 'https://packages.aliyun.com/63dc774a9dee9309492b993a/maven/repo-jpkdy'
}
```

2. 在 app 的 `build.gradle` 中增加如下配置。

```
android {
  ...
   configurations {
        all*.exclude group: 'com.mpaas.mriver', module: 'mriveruc-build'
        all*.exclude module: 'nebulaucsdk-build'
        all*.exclude module: 'nebulauc-build'
   }
}

dependencies {
  ...
  api 'com.mpaas.myweb:mpaasmyweb:10.2.3.00001160@aar'
  api 'com.mpaas.myweb:mpaasmywebapi-build:10.2.3.00001151@aar'

  // 小程序容器必须添加，没使用小程序不用添加
  api 'com.mpaas.mriver:mrivermyweb-build:10.2.3.1029@aar'

  // H5 容器必须添加
  api 'com.mpaas.nebula:nebulamyweb-build:10.2.3.00001144@aar'
}
```

3. 增加混淆配置。

```
-keep class com.alipay.mywebview.** {*;}
```

4. 申请授权码 并在 `AndroidManifest.xml` 中按照如下格式进行配置。

```
<meta-data
            android:name="MPKernelAuthKey"
            android:value="xxx" />
```

> **重要**
> 申请需要提供的信息包括 **Apk 的包名** 和 **签名摘要 SHA256**。

如何获取签名摘要

方法一：通过 Android SDK 中自带的 `apksigner` 命令进行获取。

```
apksigner verify -v --print-certs test.apk | grep 'certificate SHA-256'

// 输出示例
Signer #1 certificate SHA-256 digest: 389b49f7832f53e9017923220aa85e14dfaa4886ecd7428818bf339543cf498a
```

方法二：通过 `keytool` 获取。

```
keytool -printcert -jarfile test.apk | grep SHA256

// 输出示例
 SHA256:
A0:02:3F:10:D8:B9:8F:FF:E2:57:4B:47:A6:46:30:0C:67:98:5E:BF:5A:98:BB:D5:25:32:DE:E6:F8:91:27:07
```

验证内核应用是否成功

过滤日志 webview version，如果版本号为 `0.11.xxx` 则证明内核应用成功。

针对老客户

如果是之前接入 UC 内核的 mPaaS App，需要确认以下几点：

- 是否做过内核的特定逻辑定制，如果做过内核的特定逻辑定制需要按照新的接口重新实现，具体可以通过检索代码中是否包含 `import com.uc.xxxxx` 来判断。
- 新版本内核不再支持 Nebula 小程序，如果使用了 Nebula 小程序，请升级到小程序新容器。
- 回归相关功能，主要包括：
  - 针对 H5 容器组件，包括小程序的内嵌 H5，需要回归内核相关的功能。包括 H5 页面的所有功能以及动态切换系统内核场景。
  - 针对小程序组件，回归小程序功能。包括渲染、组件、API、插件等。

1.3.4.2. 打开 URL 判断逻辑

为了更好地保障打开 URL 时 App 的安全性，可在容器调用相关 URL 之前对 URL 进行判断，如果打开的不是白名单内的 URL 则禁止调用。

建议在调用如下接口前进行 URL 判断：

```
public class MPNebula {
    /**
    * 启动在线 URL
    *
    * @param url 在线地址
    */
    public static void startUrl(String url);

    /**
    * 启动在线 URL
    *
    * @param url    在线地址
    * @param param 启动参数
    */
    public static void startUrl(String url, Bundle param);
}

// 创建page
public static final void openH5(String url) {
    if (TextUtils.isEmpty(url)) {
        return;
    }
    H5Service h5Service = LauncherApplicationAgent.getInstance().getMicroApplicationContext()
            .findServiceByInterface(H5Service.class.getName());
    H5Bundle bundle = new H5Bundle();
    Bundle param = new Bundle();
    // 打开的在线地址
    param.putString(H5Param.LONG_URL,url);
    bundle.setParams(param);
    if (h5Service != null) {
        // 同步创建api
        H5Page h5Page=h5Service.createPage(activity,bundle);

        // 异步创建api
        h5Service.createPageAsync(activity, bundle, h5PageReadyListener);
    }
}
```

> **重要**
>
> URL 要进行精准匹配，至少要匹配到 URI 类的 scheme 和 host 信息，慎用或不用正则匹配，严格避免使用 `contains`、`startsWith`、`endsWith`、`indexOf` 等不精准函数。

1.3.4.3. JSAPI 鉴权设置方法

mPaaS 建议所有的 JSAPI 访问都需要添加访问控制，目前可以通过设置 provider 的方式添加访问控制。

1. 设置自定义权限控制 provider。

```
public class H5JSApiPermissionProviderImpl implements H5JSApiPermissionProvider {
    @Override
    public boolean hasDomainPermission(String jsapi, String url) {
        // 在该方法里，对所有url的jsapi请求进行校验，安全的url才可以放行，true表示jsapi可以调用，false表示不可以调
用
        // 注意：以下代码仅供参考，请根据您实际的业务需要，对url和jsapi进行校验
        //  另外还需对 jsapi、url、uri 等参数进行判空操作，防止发生 Nullpointer 异常
        Uri uri = Uri.parse(url);
        String domain = uri.getHost();
        String scheme = uri.getScheme();
        if (!TextUtils.isEmpty(domain) && domain.equals("www.example.com") && "https".equals(scheme))
{
            return true;
        } else {
            return false;
        }
    }

    @Override
    public boolean hasThisPermission(String jsapi, String url) {
        // 默认返回 false 即可
        return false;
    }
}
```

> **重要**
>
> URL 要进行精准匹配，至少要匹配到 URI 类的 scheme 和 host 信息，慎用或不用正则匹配，严格避免使用 contains、startsWith、endsWith、indexOf 等不精准函数。

2. 在 mPaaS 初始化之后，调用容器之前，设置 provider。

```
H5Utils.setProvider(H5JSApiPermissionProvider.class.getName(), new H5JSApiPermissionProviderImpl());
```

1.3.4.4. 添加 UC SDK

在 Android 应用中接入 UC SDK 能够有效解决各种厂商浏览器的兼容性问题，保持比系统浏览器更低的闪退率并且性能更卓越。UC SDK 提供安全支持，可及时解决安全隐患。本文介绍了添加 UC SDK 的流程。

前置条件

- 若采用原生 AAR 方式接入，需先完成 前提条件 和后续相关步骤。
- 若采用 mPaaS Inside 方式接入，需先完成 接入流程。
- 若采用组件化方式接入，需先完成 接入流程。

添加 SDK

原生 AAR 方式

参考 管理组件依赖（原生 AAR），通过 **组件管理（AAR）** 在工程中安装 **UC 内核** 组件。

mPaaS Inside 方式

在工程中通过 **组件管理** 安装 **UC 内核** 组件。更多信息，参考 管理组件依赖。

组件化方式

在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **UC 内核** 组件。更多信息，参考 接入流程。

申请 UC 内核

> **重要**
>
> 由于产品策略变更，UC 不再全面开放申请，UC Key 的发放采用人工审核机制。请填写表格提交相关信息，工作人员会进行审核并反馈申请结果。

操作步骤

1. 获取 SHA1。
   - **使用生成 UC Key 签名信息工具获取**

     自 V2.20062211 起，Android Studio mPaaS 插件提供了 **生成 UC Key 签名信息** 工具，该工具能够帮助获取申请 UC Key 的签名信息。如果您使用的 mPaaS 插件版本为 V2.20062211 或更新版本，可以使用此工具加速 UC SDK Key 申请流程。详细信息请参见 使用 mPaaS 插件。
   - **使用命令行申请**

     根据开发环境，执行相应命令获得应用签名证书指纹的 SHA1 值。

     > **说明**
     >
     > - 确认已在工程中添加 UC 内核组件依赖。
     > - 提供应用的应用 ID（Application ID）。

     - Windows

       ```
       keytool -v -list -keystore keystore 的绝对路径
       ```

     - macOS

       ```
       keytool -list -v -keystore keystore 的绝对路径
       ```

2. 填写 UC key 申请表 并提交。
3. 将获取的 Key 填入项目的 `AndroidManifest.xml` 文件中：

   ```
   <meta-data android:name="UCSDKAppKey" android:value="您申请获得的 key"/>
   ```

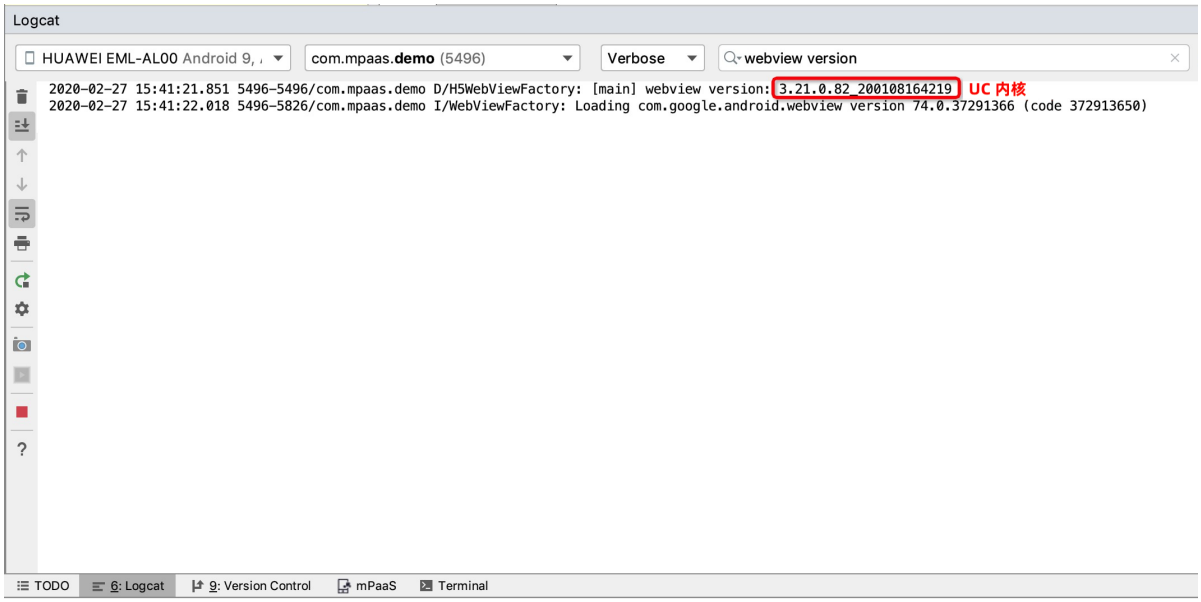
   > **说明**
   >
   > UC SDK 的授权信息与 APK 的 **包名** 以及 **签名** 绑定。因此，如果 UCWebView 没有生效，检查签名和包名与申请时使用的信息是否一致。

检测 UC 内核是否生效

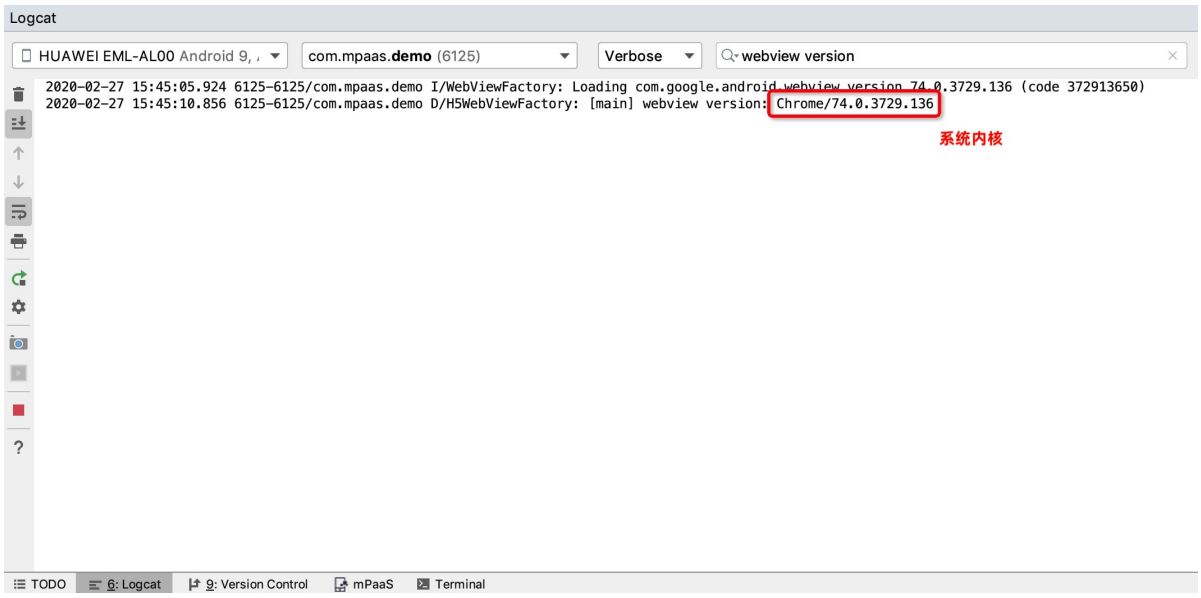
重新生成 debug 包并安装后，需要验证 UC 内核是否已经生效。

打开任意 H5 页面，在 logcat 日志中过滤 `webview version`。如果能够过滤到 UC 内核版本号，则说明 UC 内核已生效。通常情况下，版本号（version）以 2 或 3 开头的为 UC 内核版本，以 Chrome 开头的为系统内核。如下图所示：

- UC 内核



- 系统内核



1.3.4.5. 扩展 UC 内核

在使用 UC 内核时，您可以选择使用系统键盘（推荐）或 UC 键盘进行输入。

> **重要**
>
> 此文档仅用于已有 UC SDK 用户调试及支持使用，由于产品策略变更，UC SDK 不再开放申请。

选择输入键盘

通过为 H5 容器设置 `mp_h5_uc_number_input_use_system` 开关的配置，来控制在 UC 内核下是否使用系统键盘。其中，`YES` 表示使用系统键盘，`NO` 表示使用 UC 键盘。默认值为 `NO`，即使用 UC 内核时，默认使用 UC 键盘进行输入。关于如何设置 H5 容器配置，请您参考 H5 容器配置。

> **重要**
>
> UC 键盘不支持折叠屏手机，折叠屏手机只能使用系统键盘。

1.3.4.6. 开启 UC 内核 inspect 模式

从 10.1.68.14 开始，mPaaS 正式基线中的 UC 内核不再支持 inspect 模式。如果您要在开发阶段需要使用 inspect 模式调试，则需要添加 UC 内核调试包。

inspect 的开启方式

1. 下载对应版本的 debuguc 内核 so。
2. 把对应的 debuguc 内核 so，存放到 `/sdcard/slm` 目录下，完整路径为 `/sdcard/slm/libWebViewCore_ri_7z_uc.so` 。

> **重要**
> - 路径和命名要保持完全一致。
> - 复制指定 CPU 架构的 so 到 SD 卡中。
> - 开启读取 SD 卡的权限。
> - 该模式仅支持 debug 包。
> - inspect 开启成功后，会把 `/sdcard/slm` 中的内容剪切到 App 的内部存储中，所以双清 App 缓存或卸载 App 时需要再次拷贝 debuguc 内核 so 到 `/sdcard/slm` 中。

UC 内核调试包版本

详情请参见 UC 内核版本列表。

1.3.4.7. 抓取 UC 内核闪退

UC 内核是 C 层原生代码，抓取 UC 内核时需要提供 UCCrashSDK。安装 UC 内核组件后，该 SDK 会自动安装。

> **重要**
> 此文档仅用于已有 UC SDK 用户调试及支持使用，由于产品策略变更，UC SDK 不再开放申请。

将 C 层闪退上报至移动分析后台还需要在 `Manifest` 文件中加入对应的 `receiver` 。

```
<!--上报 native 闪退-->
  <receiver
    android:name="com.alipay.mobile.common.logging.process.LogReceiverInToolsProcess"
    android:enabled="true"
    android:exported="false"
    android:process=":tools">
    <intent-filter>
      <action android:name="${applicationId}.monitor.command"/>
    </intent-filter>
  </receiver>

  <receiver
    android:name="com.alipay.mobile.logmonitor.ClientMonitorWakeupReceiver"
    android:enabled="true"
    android:exported="false"
    android:process=":push">
    <intent-filter>
      <action android:name="${applicationId}.push.action.CHECK" />
      <action android:name="${applicationId}.monitor.command" />
    </intent-filter>
  </receiver>
```

> **重要**
> 使用 UC 内核闪退 SDK，会有网络请求向 UC 进行鉴权。更多内容参见 引用第三方服务数据说明。

1.3.4.8. 指定 UC 内核版本或 inspect 调试版本

> **重要**
> 此文档仅用于已有 UC SDK 用户调试及支持使用，由于产品策略变更，UC SDK 不再开放申请。

本文介绍了指定 UC 内核版本或 inspect 调试版本的方法。

使用 UC 内核时，如果需要指定 UC 内核版本，可参照如下步骤进行操作：

1. 移除当前 UC 依赖
2. 接入指定 UC 版本

移除当前 UC 依赖

若您使用 mPaaS AAR 方式接入您的工程，则无需移除当前 UC 依赖，直接 接入指定 UC 版本。

接入指定 UC 版本

在 Application Module 的 `build.gradle` 下的 `dependencies` 中，加入如下代码，强制依赖特定版本。其中，`version` 替换为指定版本，参考版本列表中 GAV 的值。如果需要使用调试版本，请参考 版本列表 中的调试 GAV 信息。

> **说明**
> 在组件化接入方式下，请把代码添加在 Portal 工程 Application Module 里 `build.gradle` 下的 `dependencies` 中。

```
dependencies {
    ...
    implementation ('com.alipay.android.phone.wallet:nebulaucsdk-build:version@aar'){
        force = true
    }
    ...
}
```

版本列表

UC 版本	GAV	调试 GAV	备注
3.22.2.66.230817192043	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.22.2.66.230817192043	com.alipay.android.phone.wallet:nebulaucsdk-build:888.3.22.2.66.230817192043	适配 Android 14。
3.22.2.46.220614210535	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.22.2.46.220614210535	com.alipay.android.phone.wallet:nebulaucsdk-build:888.3.22.2.46.220614210535	适配 Android 13。
3.22.2.30.211011154625	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.22.2.30.211011154625	com.alipay.android.phone.wallet:nebulaucsdk-build:888.3.22.2.30.211011154625	基于 3.22.2.18 修复了低概率偶现的 JS 通道缓存异常问题。

3.22.2.18.210803145558	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.22.2.18.210803145558	无。	Beta 版。基于 3.22.2.17 版本，修复了富媒体的输入问题。 **说明** 该版本为 beta 版本，不推荐使用，请优先选择其他版本。
3.22.2.17.210719105414	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.22.2.17.210719105414	无。	Beta 版。当前支付宝的最新版本。 **说明** 该版本为 beta 版本，不推荐使用，请优先选择其他版本。
3.21.0.184.210105191337	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.21.0.184.210105191337	com.alipay.android.phone.wallet:nebulaucsdk-build:888.3.21.0.184.210105191337	基于 3.21.0.174 版本，修复了低概率偶现的 JS 通道缓存异常问题。
3.21.0.174.200825145737	com.alipay.android.phone.wallet:nebulaucsdk-build:999.3.21.0.174.200825145737	com.alipay.android.phone.wallet:nebulaucsdk-build:888.3.21.0.174.200825145737	当前 10.1.68 的默认基线。

1.3.4.9. 自定义 JSAPI

JavaScript API（JSAPI）是为 H5 应用提供原生能力的接口，您可以利用这些接口使用更多的原生能力和操控能力，提高 H5 应用的用户体验。

H5 容器组件提供以下能力：

- 丰富的内置 JSAPI，实现例如页面 push、pop、标题设置等功能。更多信息查看 内置 JSAPI。
- 支持用户自定义 JSAPI 和插件功能，扩展业务需求。

本文介绍如何实现自定义 JSAPI 和插件功能。

关于此任务

自定义 JSAPI 插件具备以下特点：

- 为了让各个业务更加灵活地接入 H5 容器，H5 容器提供一种可以给外部业务注册插件配置的机制。
- 业务方使用这种方式注册自定义的外部插件，代码可以放在自己的 bundle 中，全过程不需要 H5 容器介入。使用插件配置的方式注册插件时，H5 容器只在页面调用的时候才初始化对象，并不会立即生成对象。
- 实现插件和注册插件可以不在同一个 bundle 中，实现依赖解耦。您需要通过 `H5Service` 动态将插件注入容器。

> **说明**
> - 必须在页面调用 JS 之前注入 JS，一般在静态链接的 Pipeline 中注入。如果过早注入，会出现容器的 bundle 还没加载好（h5service==null）的问题。
> - 如果注册的 bundle 不是静态链接，而是懒加载的 bundle 的话，会存在 metainfo 还没加载，JS 就要调用的问题。

前端页面调用 Native

当前端页面调用 Native 时，可以通过以下步骤分别设置客户端和前端代码。

1. 通过 MPNebula 的注册 JSAPI 接口，注册 JSAPI。

> 说明
> `MyJSApiPlugin` 的源码可以前往 获取代码示例 中查看。

- 注册 JSAPI 接口如下：

```
/**
* 注册自定义 H5 插件（JSAPI）
*
* @param className  插件类名，需要全路径（package + class）
* @param bundleName bundle 名（bundle 名可在主 module/build/intermediates/bundle/META-INF/BUNDLE.MF 中查看，如果该插件写在 portal 工程中，则 bundleName 需填空字符串 ""）
* @param scope      范围，通常为 page
* @param events     注册的事件
*/
public static void registerH5Plugin(String className, String bundleName, String scope, String[] events)
```

- 注册示例如下：

```
MPNebula.registerH5Plugin(
  MyJSApiPlugin.class.getName(),
  BuildConfig.BUNDLE_NAME,
  "page",
  new String[]{"myapi1","myapi2",H5Plugin.CommonEvents.H5_PAGE_SHOULD_LOAD_URL}
);
```

2. 前端调用。

   通过以下方法使得前端调用自定义 JSAPI：将 event 参数修改成以上插件注册的事件。插件将通过 `event.getParam()` 获取 JS 的传值，并从中解析出数据。

```
AlipayJSBridge.call('myapi2', {
 param2: 'World'
},
function(result) {
 console.log(result);
});
```

Native 调用前端页面

H5 容器也可以是 Native 主动调用前端页面。以网络变化 JSAPI 为例，页面监听该事件，前端代码和客户端代码如下所示：

1. 前端注册监听。

```
document.addEventListener('h5NetworkChange',
function(e) {
 alert("网络环境发生变化，可调用 getNetworkType 接口获取详细信息");
},
false);
```

2. 客户端监听到网络发生变化时，向页面发送调用事件。

```
JSONObject param = new JSONObject();
// param 设置自定义的参数
param.put("data", param);
H5Page h5Page = h5Service.getTopH5Page();
if (h5Page != null) {
 h5Page.getBridge().sendDataWarpToWeb("h5NetworkChange", param, null);
}
```

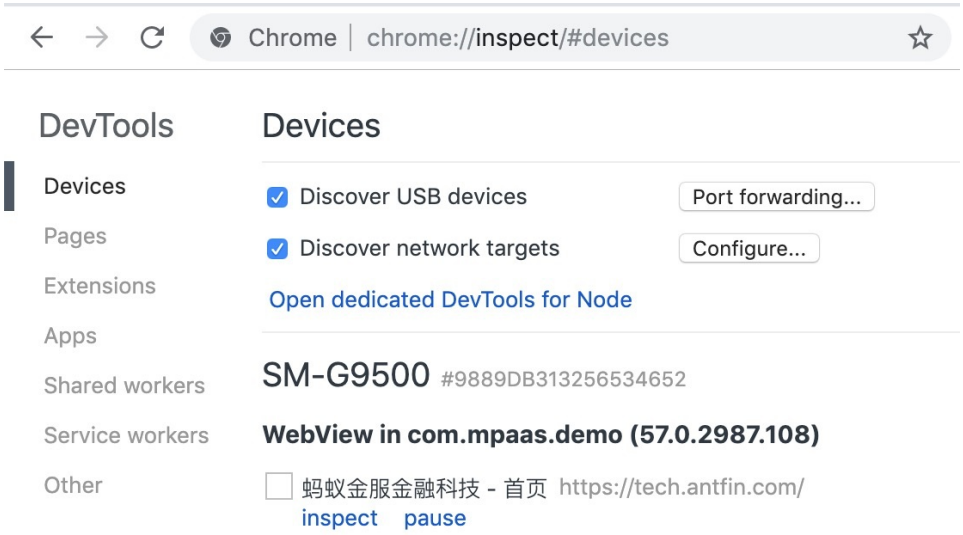
使用 Inspect

使用 Chrome 的 inspect 工具，检查调用自定义的 JSAPI 是否有效：

1. 将手机连接电脑，在电脑中打开 Chrome 浏览器，输入 `chrome://inspect` ，进入调试页面。

2. 使用 mPaaS demo 打开蚂蚁金服金融科技首页。此时，Chrome 的 inspect 页面会变成如下所示：

> **说明**
> 如果电脑打开 `chrome://inspect` 会出现白屏，请升级到 Chrome 最新版本。



3. 单击图中的 **inspect**，出现如下页面：

4. 单击页面工具栏上的 **Console** 进入页面调试模式。接下来，您就可以如图例所示调用自己的 API 接口了。

相关链接

- 获取代码示例
- 常见问题

1.3.4.10. 自定义导航栏（10.1.68）

Nebula 容器可支持自定义导航栏，您可以制定导航栏的样式，例如标题的位置、返回按钮的样式等。本文将向您介绍如何在 10.1.68 基线中自定义导航栏。

前提条件

在完整阅读此指南前，需提前知晓以下四点：

- 由于小程序和 H5 共用导航栏的实现，在进行自定义导航栏的开发时应将 H5 和小程序使用导航栏的情况都考虑在内，除非确定使用场景不包含小程序或 H5。
- 自定义导航栏 **必须符合容器调用的标准流程** ，请仔细阅读本文档并按照要求进行开发。
- 小程序导航栏默认使用内置导航栏，如需开启自定义导航栏，详情请参考 容器配置。
- 由于导航栏的颜色可以动态设置，为达到最佳体验效果，您应当准备 **两套主题配置** 并根据不同场景进行切换。

接入步骤

1. 继承 **AbsTitleView** 抽象类并实现自定义导航栏。
2. 实现 `H5ViewProvider` ，在 `createTitleView` 方法中创建并返回自定义导航栏实例。

```
public class H5ViewProviderImpl implements H5ViewProvider {
    @Override
    public H5TitleView createTitleView(Context context) {
        return new NewH5TitleViewImpl(context);
    }

    @Override
    public H5NavMenuView createNavMenu() {
        return null;
    }

    @Override
    public H5PullHeaderView createPullHeaderView(Context context, ViewGroup viewGroup) {
        return null;
    }

    @Override
    public H5WebContentView createWebContentView(Context context) {
        return null;
    }
}
```

3. 在合适的地方，比如应用启动时，设置 `H5ViewProvider` 至容器。

```
MPNebula.setCustomViewProvider(new H5ViewProviderImpl());
```

4. 如果工程是基于 Portal&Bundle 架构，需额外设置。

```
H5Utils.setProvider(H5ReplaceResourceProvider.class.getName(), new H5ReplaceResourceProvider() {
    @Override
    public String getReplaceResourcesBundleName() {
        return BuildConfig.BUNDLE_NAME;
    }
});
```

更多扩展

背景色

resetTitle 一般在前端页面调用 setTitleColor 的 JSAPI 时被触发。这种情况下，建议您将导航栏的背景以及其他后续提及组件的色值等显示元素重置为默认元素。

```
    /**
     * 返回导航栏背景颜色值
     * @return
     */
    public abstract int getBackgroundColor();

    /**
     * 设置导航栏透明度
     * @param alpha
     */
    public abstract void setBackgroundAlphaValue(int alpha);

    /**
     * 设置导航栏背景色，不含 alpha 值
     * @param color
     */
    public abstract void setBackgroundColor(int color);

    /**
     * 重置导航栏
     */
    public abstract void resetTitle();
```

标题

副标题当前仅在 H5 场景中支持，如应用不需要副标题，可以不提供副标题实现。

为了使 H5 页面能够监听 点击标题栏事件，导航栏应该在合适的地方调用 `invokeTitleClickEvent` 方法；监听 点击副标题栏事件，则调用 `invokeSubTitleClickEvent` 方法。

```
    /**
     * 返回主标题文本
     * @return
     */
    public abstract String getTitle();

    /**
     * 设置主标题文本
     * @param title
     */
    public abstract void setTitle(String title);

    /**
     * 设置副标题文本
     * @param subTitle
     */
    public abstract void setSubTitle(String subTitle);

    /**
     * 返回主标题视图
     * @return
     */
    public abstract TextView getMainTitleView();

    /**
     * 返回副标题视图
     * @return
     */
    public abstract TextView getSubTitleView();
```

左侧控制区

```
    /**
     * 设置关闭按钮可见性
     * @param visible
     */
    public abstract void showCloseButton(boolean visible);

    /**
     * 设置返回按钮可见性
     * @param visible
     */
    public abstract void showBackButton(boolean visible);

    /**
     * 设置去首页按钮可见性
     * @param visible
     */
    public abstract void showBackHome(boolean visible);

    /**
     * 设置标题栏加载提示符可见性
     * @param visible
     */
    public abstract void showTitleLoading(boolean visible);
```

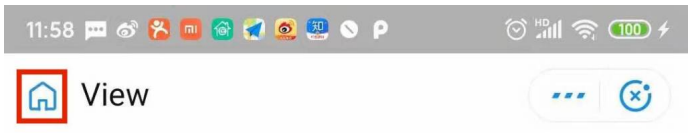
关闭按钮

如上图红框区域中所显示，关闭按钮仅出现在 H5 场景中，当在线页面的历史（即浏览器历史页面数）大于 1 时，容器会调用 `showCloseButton` 方法来控制其可见性。此按钮被点击时必须调用 `invokePageCloseEvent` 方法以符合容器行为规范。

返回按钮

上图红框区域中的返回按钮是 **必须** 要在自定义导航栏上实现的控件，容器调用 `showBackButton` 方法控制其可见性。响应返回按钮点击时，开发者必须调用 `invokePageBackEvent` 方法以符合容器行为规范。

去首页按钮



去首页按钮仅在小程序场景下使用，当跳转小程序的非首页页面时，容器会调用 `showBackHome` 方法控制其可见性。响应按钮点击事件时，开发者必须调用 `invokeHomeClickEvent` 方法以符合容器行为规范。

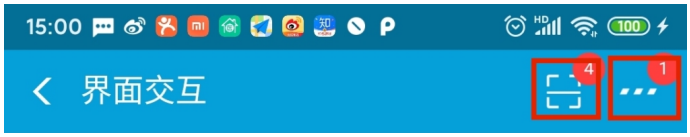
加载提示符

当 H5 页面或者小程序调用导航栏加载动画 API 时，容器会调用 `showTitleLoading` 方法控制其可见性。

右侧控制区

右侧控制区，也可称为 `OptionMenu` 控制区，主要用于提供更多操作给用户。

- H5 容器：



- 小程序：

由上图可见，开发者需要为 `OptionMenu` 控制区预留两个视图区域，容器操作时会按照索引来操作这两块区域，**从右至左，从 0 开始**。

```
    public abstract void showOptionMenu(boolean visible);

    public abstract View getOptionMenuContainer(int index);

    public abstract void setOptionMenu(boolean reset, boolean override, boolean isTinyApp,
List<MenuData> menus);
```

容器通过调用 `showOptionMenu` 方法来控制 `OptionMenu` 区域可见性，在某些情况也需要获取该区域的视图进行操作，开发者需要正确实现 `getOptionMenuContainer` 方法。

实现 `setOptionMenu` 方法，请参考 设置右上角按钮 API 入参进行合理的实现。`icontype` 为内置按钮样式，仅在 H5 场景中可用，如果接入场景中不会用到则可忽略，否则需要为不同类型配置相应的按钮。`redDot` 与 `icontype` 类似，可以选择性接入。

为了使 H5 页面或小程序监听右上角按钮点击事件，开发者需要调用 `invokeOptionClickEvent` 方法。

> **重要**
> 在小程序场景下：
> - `setOptionMenu` 方法的 `isTinyApp` 参数值必定为 `true`，以此区分 H5 和小程序场景。
> - 必须从 **最右侧倒数第二个** 按钮开始设置按钮。

小程序右上角控制区

当处于小程序场景时，最右侧区域需要特殊实现，步骤如下：

1. 继承 `AbsTinyOptionMenuView` 抽象类实现自定义控制区。
2. 在合适的地方，比如应用启动时，设置 `TinyOptionMenuViewProvider` 至容器。

```
H5Utils.setProvider(TinyOptionMenuViewProvider.class.getName(), new TinyOptionMenuViewProvider() {
 @Override
 public AbsTinyOptionMenuView createView(Context context) {
     return new TinyOptionMenuView(context);
 }
});
```

按照容器规范要求，开发者必须实现和配置 **更多** 和 **关闭** 按钮视图。

```
    public abstract void setOptionMenuOnClickListener(View.OnClickListener listener);

    public abstract void setCloseButtonOnClickListener(View.OnClickListener listener);

    public abstract void setCloseButtonOnLongClickListener(View.OnLongClickListener listener);

    public abstract void onStateChanged(TinyAppActionState currentState);

    public abstract View getView();
```

容器会调用上方代码中的前三个方法来设置合理的响应回调，开发者必须让指定的视图设置该响应回调。

`onStateChanged` **方法在定位、蓝牙场景中会被调用，举例来说，当小程序正在使用定位服务的时，容器会调用此方法，开发者可作出响应，下面是样式效果。**

简单示例代码：

```
public class TinyOptionMenuView extends AbsTinyOptionMenuView {

    private View container;

    private ImageView ivMore;

    private View ivClose;

    private Context context;

    public TinyOptionMenuView(Context context) {
        this.context = context;
        ViewGroup parent = null;
        if (context instanceof Activity) {
            parent = (ViewGroup) ((Activity) context).findViewById(android.R.id.content);
        }
        container = LayoutInflater.from(context).inflate(R.layout.layout_tiny_right, parent, false);
        ivClose = container.findViewById(R.id.close);
        ivMore = (ImageView) container.findViewById(R.id.more);

    }

    @Override
    public View getView() {
        return container;
    }

    @Override
    public void setOptionMenuOnClickListener(View.OnClickListener onClickListener) {
        ivMore.setOnClickListener(onClickListener);
    }

    @Override
    public void setCloseButtonOnClickListener(View.OnClickListener onClickListener) {
        ivClose.setOnClickListener(onClickListener);
    }

    @Override
    public void setCloseButtonOnLongClickListener(View.OnLongClickListener onLongClickListener) {
        ivClose.setOnLongClickListener(onLongClickListener);
    }

    @Override
    public void onStateChanged(TinyAppActionState state) {
        if (state == null) {
            ivMore.setImageDrawable(context.getResources().getDrawable(R.drawable.icon_more));
        } else if (state.getAction().equals(TinyAppActionState.ACTION_LOCATION)) {

ivMore.setImageDrawable(context.getResources().getDrawable(R.drawable.icon_miniprogram_location));
        }
    }
}
```

主题变更

不同的小程序或 H5 应用可能会设置不同的导航栏背景色，考虑到用户体验，在导航栏背景色发生变化的时候，也需要对导航栏上的其他元素进行调整，比如右上角控制区。

在导航栏的扩展上，右上角控制区和导航栏分别为不同的组件，所以我们提供接口来方便开发者能够让右上角控制区能够及时响应导航栏的变化。

- `AbsTinyOptionMenuView` 提供 `onTitleChanged` 方法供 override 来响应导航栏变化。当此方法被调用时，可通过传入H5TitleView 对象来获取导航栏的信息，如背景色。也可以使用 `AbsTinyOptionMenuView` 提供的 `getTitleBar` 方法直接获取导航栏对象。另外，您可以将 H5TitleView 转换成自己实现的导航栏对象，因为 `AbsTitleView` 实现了 `H5TitleView` 接口类，这样能够获得更多的导航栏信息。

```
protected void onTitleChange(H5TitleView title)
```

- 您需要主动调用 `AbsTitleView` 提供的 `notifyTitleBarChanged` 方法触发 `onTitleChange` 被调用，比如在对导航栏对象设置背景色时。举例说明：

```
package com.mpaas.demo.nebula;

public class NewH5TitleViewImpl extends AbsTitleView {

    @Override
    public void setBackgroundAlphaValue(int i) {
        content.getContentBgView().setAlpha(i);
        notifyTitleBarChanged();
    }

    @Override
    public void setBackgroundColor(int i) {
        content.getContentBgView().setColor(i);
        notifyTitleBarChanged();
    }

    @Override
    public void resetTitle() {

content.getContentBgView().setColor(context.getResources().getColor(R.color.h5_default_titlebar_color))

        notifyTitleBarChanged();
    }
}
```

上述示例代码调用 `notifyTitleBarChange` 方法时，`AbsTinyOptionMenuView` 的子类对象可能并没有初始化完毕，因此建议覆写 `setH5Page` 方法，获取导航栏信息并判定当前主题，示例如下：

```
public class TinyOptionMenuView extends AbsTinyOptionMenuView {

    @Override
    public void setH5Page(H5Page h5Page) {
        super.setH5Page(h5Page);
        // title becomes available from here.
        if (getTitleBar().getBackgroundColor() == -1) {
            bgView.setBackgroundColor(Color.RED);
        }
    }
}
```

1.3.4.11. 自定义导航栏（10.1.60）

在 10.1.60 基线下，如果有自定义 H5 容器标题栏的需求，可参考下方示例代码：

H5TitleViewImpl.java

```
package com.mpaas.demo.nebula;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
```

```
import android.text.TextUtils;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.h5container.api.H5Page;
import com.alipay.mobile.h5container.api.H5Param;
import com.alipay.mobile.h5container.api.H5Plugin;
import com.alipay.mobile.nebula.util.H5Log;
import com.alipay.mobile.nebula.util.H5StatusBarUtils;
import com.alipay.mobile.nebula.util.H5Utils;
import com.alipay.mobile.nebula.view.H5TitleBarFrameLayout;
import com.alipay.mobile.nebula.view.H5TitleView;
import com.alipay.mobile.nebula.view.IH5TinyPopMenu;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by omg on 2018/7/23.
 */

public class H5TitleViewImpl implements H5TitleView, View.OnClickListener {

    private static final String TAG = "H5TitleViewImpl";

    private Context mContext;

    private H5TitleBarFrameLayout contentView;

    private String title;

    // 此处定义标题栏的基本控件
    private TextView mTitleView;

    private TextView mSubTitleView;

    private View mCloseButton;

    private View mBackButton;

    private View vDivider;

    private View hDivider;

    private View statusBarAdjustView;

    /**
     * ==== OptionMenu的各种View Start ====
     */
    // OptionMenu的Container
    public View h5NavOptions;
    public View h5NavOptions1;
    public List<View> h5NavOptionsList = new ArrayList<View>();

    // ---- OptionMenu 三种形态 Start ---- //
    // 1. OptionType.MENU(默认) -默认的 Option 按钮
    public TextView btMenu;
    public TextView btMenu1;
```

```
    public TextView btMenu1;
    public List<TextView> btMenuList = new ArrayList<TextView>();

    // 2. OptionType.ICON - 通过 setOptionMenu 手动设置的 icon
    public ImageButton btIcon;
    public ImageButton btIcon1;
    public List<ImageButton> btIconList = new ArrayList<ImageButton>();

    // 3. OptionType.TEXT - 文字
    public TextView btText;
    public TextView btText1;
    public List<TextView> btTextList = new ArrayList<TextView>();
    // ---- OptionMenu 三种形态 Over ---- //

    // Web 页面的实例接口类，可用于控制 web 行为
    private H5Page h5Page;

    private int visibleOptionNum = 0;
    private IH5TinyPopMenu h5TinyPopMenu;

    /**
     * H5 容器标题栏构造方法
     * 注意：标题栏布局 xml 文件必须以 H5TitleBarFrameLayout 作为根节点
     * @param context
     */
    public H5TitleViewImpl(Context context) {
        mContext = context;
        ViewGroup parent = null;
        if (context instanceof Activity) {
            parent = (ViewGroup) ((Activity) mContext).findViewById(android.R.id.content);
        }
        contentView = (H5TitleBarFrameLayout)
LayoutInflater.from(context).inflate(R.layout.h5_navigation_bar, parent, false);

contentView.getContentBgView().setColor(context.getResources().getColor(R.color.h5_default_titlebar_color

        mTitleView = (TextView) contentView.findViewById(R.id.h5_tv_title);
        mTitleView.setOnClickListener(this);
        mSubTitleView = (TextView) contentView.findViewById(R.id.h5_tv_subtitle);
        mSubTitleView.setOnClickListener(this);
        mCloseButton = contentView.findViewById(R.id.h5_nav_close);
        mCloseButton.setOnClickListener(this);
        mBackButton = contentView.findViewById(R.id.h5_tv_nav_back);
        mBackButton.setOnClickListener(this);
        vDivider = contentView.findViewById(R.id.h5_v_divider);
        hDivider = contentView.findViewById(R.id.h5_h_divider_intitle);
        h5NavOptions = contentView.findViewById(R.id.h5_nav_options);
        h5NavOptions1 = contentView.findViewById(R.id.h5_nav_options1);
        statusBarAdjustView = contentView.findViewById(R.id.h5_status_bar_adjust_view);

        btIcon = (ImageButton) contentView.findViewById(R.id.h5_bt_image);
        btText = (TextView) contentView.findViewById(R.id.h5_bt_text);
        btMenu = (TextView) contentView.findViewById(R.id.h5_bt_options);

        btIcon1 = (ImageButton) contentView.findViewById(R.id.h5_bt_image1);
        btText1 = (TextView) contentView.findViewById(R.id.h5_bt_text1);
        btMenu1 = (TextView) contentView.findViewById(R.id.h5_bt_options1);

        //add view to list
        h5NavOptionsList.add(h5NavOptions);
        h5NavOptionsList.add(h5NavOptions1);

        btIconList.add(btIcon);
        btIconList.add(btIcon1);
```

```
        btTextList.add(btText);
        btTextList.add(btText1);

        btMenuList.add(btMenu);
        btMenuList.add(btMenu1);

        btText.setOnClickListener(this);
        btIcon.setOnClickListener(this);
        btText1.setOnClickListener(this);
        btIcon1.setOnClickListener(this);
        btMenu.setOnClickListener(this);
        btMenu1.setOnClickListener(this);

    }

    /**
     * 容器调用此方法获得主标题内容
     */
    @Override
    public String getTitle() {
        return title;
    }

    /**
     * 容器调用此方法设置主标题内容
     */
    @Override
    public void setTitle(String s) {
        title = s;
        mTitleView.setText(s);
    }

    /**
     * 容器调用此方法设置副标题内容
     */
    @Override
    public void setSubTitle(String s) {
        mSubTitleView.setVisibility(View.VISIBLE);
        mSubTitleView.setText(s);
    }

    /**
     * 暂时忽略，可不实现
     */
    @Override
    public void setImgTitle(Bitmap bitmap) {

    }

    /**
     * 暂时忽略，可不实现
     */
    @Override
    public void setImgTitle(Bitmap bitmap, String s) {

    }

    /**
     * 容器设置是否显示关闭按钮
     */
    @Override
    public void showCloseButton(boolean b) {
        mCloseButton.setVisibility(b ? View.VISIBLE : View.GONE);
```

```
    }

    /**
     * 容器获取标题栏 View
     */
    @Override
    public View getContentView() {
        return contentView;
    }

    /**
     * 容器获取标题栏背景用于设置背景色
     */
    @Override
    public ColorDrawable getContentBgView() {
        return contentView.getContentBgView();
    }

    /**
     * 容器获取主标题 View
     * 不可为空
     */
    @Override
    public TextView getMainTitleView() {
        return mTitleView;
    }

    /**
     * 容器获取副标题 View
     * 不可为空
     */
    @Override
    public TextView getSubTitleView() {
        return mSubTitleView;
    }

    /**
     * 设置是否展示返回按钮
     */
    @Override
    public void showBackButton(boolean b) {
        mBackButton.setVisibility(b ? View.VISIBLE : View.GONE);
    }

    /**
     * 设置是否显示右上角菜单项
     */
    @Override
    public void showOptionMenu(boolean isShow) {
        if (isShow) {
            switch (visibleOptionNum) {
                case 1:
                    h5NavOptions.setVisibility(View.VISIBLE);
                    break;
                case 2:
                    h5NavOptions.setVisibility(View.VISIBLE);
                    h5NavOptions1.setVisibility(View.VISIBLE);
                    break;
            }
        } else {
            h5NavOptions.setVisibility(View.GONE);
            h5NavOptions1.setVisibility(View.GONE);
        }
```

```
    }

    /**
     * 设置右上角菜单项展示类型，可为图标、文字
     */
    @Override
    public void setOptionType(H5Param.OptionType optionType) {
        setOptionType(optionType, 0, true);
    }

    /**
     * 设置右上角菜单项展示类型，可为图标、文字
     * @param byIndex 是否只对某个菜单项设置展示类型
     */
    @Override
    public void setOptionType(H5Param.OptionType type, int num, boolean byIndex) {
        boolean icon = false;
        boolean text = false;
        boolean menu = false;
        if (type == H5Param.OptionType.ICON) {
            icon = true;
        } else if (type == H5Param.OptionType.TEXT) {
            text = true;
        } else if (type == H5Param.OptionType.MENU) {
            menu = true;
        }
        ctrlbtText(num, text ? View.VISIBLE : View.GONE, byIndex);
        ctrlbtIcon(num, icon ? View.VISIBLE : View.INVISIBLE, byIndex);
        ctrlbtMenu(num, menu ? View.VISIBLE : View.INVISIBLE, byIndex);
    }

    //view visible control
    private boolean isOutOfBound(int num, int length) {
        if (length == 0) {
            return true;
        }
        return length < num;
    }

    private void ctrlbtText(int num, int visible, boolean byIndex) {
        if (isOutOfBound(num, btTextList.size())) {
            return;
        }
        if (byIndex) {
            btTextList.get(num).setVisibility(visible);
        } else {
            for (int i = 0; i < num; i++) {
                btTextList.get(i).setVisibility(visible);
            }
        }
    }

    private void ctrlbtIcon(int num, int visible, boolean byIndex) {
        if (isOutOfBound(num, btIconList.size())) {
            return;
        }
        if (byIndex) {
            btIconList.get(num).setVisibility(visible);
        } else {
            for (int i = 0; i < num; i++) {
                btIconList.get(i).setVisibility(visible);
            }
        }
    }
```

```
    }

    private void ctrlbtMenu(int num, int visible, boolean byIndex) {
        if (isOutOfBound(num, btMenuList.size())) {
            return;
        }
        if (byIndex) {
            btMenuList.get(num).setVisibility(visible);
        } else {
            for (int i = 0; i < num; i++) {
                btMenuList.get(i).setVisibility(visible);
            }
        }
    }

    /**
     * 设置是否在标题栏上显示加载状态，可选择实现方式
     */
    @Override
    public void showTitleLoading(boolean b) {

    }

    /**
     * 可忽略
     */
    @Override
    public void showTitleDisclaimer(boolean b) {

    }

    // 设置右上角按钮图标
    @Override
    public void setBtIcon(Bitmap btIcon, int index) {
        if (isOutOfBound(index, btIconList.size())) {
            return;
        }
        btIconList.get(index).setImageBitmap(btIcon);
    }

    @Override
    public void setH5Page(H5Page h5Page) {
        this.h5Page = h5Page;
    }

    /**
     * 根据 JS 传递过来的参数设置右上角菜单
     */
    @Override
    public void setOptionMenu(JSONObject params) {
        boolean reset = H5Utils.getBoolean(params, "reset", false);
        boolean override = H5Utils.getBoolean(params, "override", false);
        JSONArray menus = H5Utils.getJSONArray(params, "menus", null);
        if (reset) {
            h5NavOptions1.setVisibility(View.GONE);
            setOptionType(H5Param.OptionType.MENU, 0, true);
            visibleOptionNum = 1;
            return;
        }
        if (menus != null && !menus.isEmpty()) {
            visibleOptionNum = 0;
            if (override) {
                int menuSize = menus.size() > 2 ? 2 : menus.size();
                for (int i = 0; i < menuSize; i++) {
```

```
//                    h5NavOptionsList.get(i).setVisibility(View.VISIBLE);
                    JSONObject menusItem = menus.getJSONObject(i);
                    setOptionMenuInternal(menusItem, i);
                    visibleOptionNum++;
                }
            } else {
                visibleOptionNum = 2;
//                h5NavOptionsList.get(1).setVisibility(View.VISIBLE);
                JSONObject menusItem = menus.getJSONObject(0);
                setOptionMenuInternal(menusItem, 1);
            }
        } else {
            setOptionMenuInternal(params, 0);
            visibleOptionNum = 1;
        }
    }

    private void setOptionMenuInternal(JSONObject params, int index) {
        String title = H5Utils.getString(params, "title");
        String icon = H5Utils.getString(params, "icon");
        String icontype = H5Utils.getString(params, "icontype");
        String contentDesc = H5Utils.getString(params, "contentDesc");
        String colorText = H5Utils.getString(params, "color");

        // default white color
        int color = 0xFF108ee9;
        if (!TextUtils.isEmpty(colorText)) {
            try {
                color = Color.parseColor(colorText);
            } catch (Throwable ignore) {
                //can not find logutil
            }
            color = 0xFF000000 | color;
            btTextList.get(index).setTextColor(color);
        } else {
            int currentColor = mTitleView.getCurrentTextColor();
            currentColor = 0xFF000000 | currentColor;
            H5Log.d(TAG, "setOptionMenuInternal currentColor is " + currentColor);
            if (currentColor != 0xFF111111) {
                btText.setTextColor(0xFFFFFFFF);
                btText1.setTextColor(0xFFFFFFFF);
            } else {
                btText.setTextColor(0xFF108ee9);
                btText1.setTextColor(0xFF108ee9);
            }
        }

        if (!TextUtils.isEmpty(title)) {
            title = title.trim();
            btTextList.get(index).setText(title);
            setOptionType(H5Param.OptionType.TEXT, index, true);
            btTextList.get(index).setContentDescription(title);
        } else if (!TextUtils.isEmpty(icon) || !TextUtils.isEmpty(icontype)) {
            if (!TextUtils.isEmpty(contentDesc)) {
                btIconList.get(index).setContentDescription(contentDesc);
            }
        }
    }

    /**
     * 容器获取返回按钮和标题内容之间的分割线
     * 可返回空
     */
    @Override
```

```
    public View getDivider() {
        return vDivider;
    }

    /**
     * 容器获取标题栏和 Web 页面间的分割线
     * 不可为空
     */
    @Override
    public View getHdividerInTitle() {
        return hDivider;
    }

    /**
     * 容器获取下拉菜单弹出位置的锚点 View
     */
    @Override
    public View getPopAnchor() {
        return btMenu;
    }

    /**
     * 容器重置标题栏背景色
     */
    @Override
    public void resetTitleColor(int color) {

    }

    /**
     * 暂时忽略
     */
    @Override
    public void switchToWhiteTheme() {

    }

    /**
     * 暂时忽略
     */
    @Override
    public void switchToBlueTheme() {

    }

    /**
     * 容器页面销毁时触发，此处可释放被引用的 View
     */
    @Override
    public void releaseViewList() {
        if (h5NavOptionsList != null) {
            h5NavOptionsList.clear();
        }
        if (btIconList != null) {
            btIconList.clear();
        }
        if (btTextList != null) {
            btTextList.clear();
        }
        if (btMenuList != null) {
            btMenuList.clear();
        }
    }
```

```
/**
 * 容器设置沉浸式标题栏颜色
 */
@Override
public void openTranslucentStatusBarSupport(int color) {
    if (H5StatusBarUtils.isSupport()) {
        int statusBarHeight = H5StatusBarUtils.getStatusBarHeight(mContext);

        if (statusBarHeight == 0) { //保护，万一有 rom 没办法拿到状态栏高度的话，则在这里不生效。
            return;
        }
        LinearLayout.LayoutParams layoutParams =
                (LinearLayout.LayoutParams) statusBarAdjustView.getLayoutParams();
        layoutParams.height = statusBarHeight;
        statusBarAdjustView.setLayoutParams(layoutParams);
        statusBarAdjustView.setVisibility(View.VISIBLE);

        try {
            H5StatusBarUtils.setTransparentColor((Activity) mContext, color);
        } catch (Exception e) {
            H5Log.e(TAG, e);
        }
    }
}

/**
 * 暂时忽略
 */
@Override
public void switchToTitleBar() {

}

/**
 * 暂时忽略
 */
@Override
public View setTitleBarSearch(Bundle bundle) {
    return null;
}

/**
 * 暂时忽略
 */
@Override
public void setBackCloseBtnImage(String s) {

}

/**
 * 设置标题栏字体颜色
 */
@Override
public void setTitleTxtColor(int i) {
    mTitleView.setTextColor(i);
    mSubTitleView.setTextColor(i);
}

/**
 * 容器获取右上角菜单 View，必须是 ViewGroup 及子类
 */
@Override
```

```
public View getOptionMenuContainer() {
    return h5NavOptions;
}

/**
 * 容器根据位置获取右上角菜单 View，必须是 ViewGroup 及子类
 */
@Override
public View getOptionMenuContainer(int index) {
    switch (index) {
        case 0:
            return h5NavOptions;
        case 1:
            return h5NavOptions1;
        default:
            return getOptionMenuContainer();
    }
}

/**
 * 暂时忽略
 */
@Override
public void setIH5TinyPopMenu(IH5TinyPopMenu tinyPopMenu) {
    this.h5TinyPopMenu = tinyPopMenu;
}

/**
 * 暂时忽略
 */
@Override
public IH5TinyPopMenu getH5TinyPopMenu() {
    return null;
}

/**
 * 暂时忽略
 */
@Override
public void setTitleView(View view) {

}

/**
 * 暂时忽略
 */
@Override
public void initTitleSegControl(JSONObject jsonObject) {

}

/**
 * 暂时忽略
 */
@Override
public void enableTitleSegControl(boolean b) {

}

/**
 * 暂时忽略
 */
@Override
```

```
    public void enableBackButtonBackground(boolean b) {

    }

    /**
     * 处理标题栏上不同控件的点击事件
     * 如果 JS 需要收到事件，需按照该代码中呈现的方式发送事件给 JS
     * 比如点击返回按钮，则发送 H5Plugin.CommonEvents.H5_TOOLBAR_BACK 事件
     */
    @Override
    public void onClick(View view) {
        if (h5Page == null) {
            return ;
        }

        String eventName = null;
        JSONObject data = null;

        if (view == mBackButton) {
            eventName = H5Plugin.CommonEvents.H5_TOOLBAR_BACK; // 发送后退事件
        } else if (view == mCloseButton) {
            eventName = H5Plugin.CommonEvents.H5_TOOLBAR_CLOSE; // 发送关闭页面事件
        }else if (view.equals(btIcon) || view.equals(btText)) {
            eventName = H5Plugin.CommonEvents.H5_TITLEBAR_OPTIONS;
            data = new JSONObject();
            data.put("index", 0);
        } else if (view.equals(btIcon1) || view.equals(btText1)) {
            eventName = H5Plugin.CommonEvents.H5_TITLEBAR_OPTIONS;
            data = new JSONObject();
            data.put("index", 1);
        } else if (view.equals(btMenu) || view.equals(btMenu1)) {
            eventName = H5Plugin.CommonEvents.H5_TITLEBAR_OPTIONS;
            data = new JSONObject();
            data.put("fromMenu", true);
            data.put("index", view.equals(btMenu) ? 0 : 1);
        } else if (view.equals(mTitleView)) {
            eventName = H5Plugin.CommonEvents.H5_TITLEBAR_TITLE;
        } else if (view.equals(mSubTitleView)) {
            eventName = H5Plugin.CommonEvents.H5_TITLEBAR_SUBTITLE;
        }

        if (!TextUtils.isEmpty(eventName)) {
            h5Page.sendEvent(eventName, data);
        }
    }
}
```

h5_navigation_bar.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.alipay.mobile.nebula.view.H5TitleBarFrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/h5_title_bar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@android:color/transparent">

    <LinearLayout
        android:id="@+id/h5_rl_title_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/transparent"
```

```
        android:orientation="vertical">

        <View
            android:id="@+id/h5_status_bar_adjust_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:visibility="gone"/>

        <RelativeLayout
            android:id="@+id/h5_title_bar_layout"
            android:layout_width="match_parent"
            android:layout_height="48dp">

            <ImageButton
                android:id="@+id/h5_tv_nav_back"
                android:layout_width="48dp"
                android:layout_height="match_parent"
                android:layout_alignParentLeft="true"
                android:layout_centerVertical="true"
                android:background="@android:color/transparent"
                android:scaleType="centerInside"
                android:padding="12dp"
                android:src="@drawable/back"/>

            <ImageButton
                android:id="@+id/h5_nav_close"
                android:layout_width="48dp"
                android:layout_height="match_parent"
                android:layout_centerVertical="true"
                android:padding="12dp"
                android:layout_marginLeft="-6dp"
                android:layout_toRightOf="@+id/h5_tv_nav_back"
                android:background="@android:color/transparent"
                android:clickable="true"
                android:scaleType="centerInside"
                android:src="@drawable/close"/>

            <View
                android:id="@+id/h5_v_divider"
                android:layout_width="0.7dp"
                android:layout_height="24dp"
                android:layout_centerVertical="true"
                android:layout_gravity="center"
                android:layout_marginRight="12dp"
                android:layout_toRightOf="@+id/h5_nav_close"
                tools:ignore="ContentDescription"/>

            <RelativeLayout
                android:id="@+id/h5_rl_title"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerVertical="true"
                android:gravity="center">


                <LinearLayout
                    android:id="@+id/h5_ll_title"
                    android:layout_width="wrap_content"
                    android:layout_height="match_parent"
                    android:gravity="center"
                    android:orientation="vertical">

                    <FrameLayout
                        android:layout_width="wrap_content"
```

```
                    android:layout_height="wrap_content">

                    <TextView
                        android:id="@+id/h5_tv_title"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:ellipsize="end"
                        android:singleLine="true"
                        android:textColor="@android:color/white"
                        android:textSize="16dp" />

                    <ImageView
                        android:id="@+id/h5_tv_title_img"
                        android:layout_width="wrap_content"
                        android:layout_height="36dp"
                        android:scaleType="centerInside"
                        android:visibility="gone"/>
                </FrameLayout>

                <TextView
                    android:id="@+id/h5_tv_subtitle"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:singleLine="true"
                    android:textColor="@android:color/white"
                    android:textSize="12dp"
                    android:visibility="gone"
                    tools:visibility="visible" />
            </LinearLayout>

        </RelativeLayout>

        <!-- optionmenu0-->
        <FrameLayout
            android:id="@+id/h5_nav_options"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_alignParentRight="true"
            android:layout_centerVertical="true"
            android:layout_marginLeft="6dp">

            <ImageButton
                android:id="@+id/h5_bt_image"
                android:layout_width="32dp"
                android:layout_height="32dp"
                android:layout_gravity="center|right"
                android:layout_marginRight="12dp"
                android:background="@android:color/transparent"
                android:padding="4dp"
                android:scaleType="fitCenter"
                android:visibility="gone"/>

            <TextView
                android:id="@+id/h5_bt_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center|right"
                android:layout_marginRight="12dp"
                android:background="@null"
                android:ellipsize="end"
                android:maxLength="8"
                android:singleLine="true"
                android:textColor="#ffffff"
```

```
                    android:textSize="16dp"/>

                <TextView
                    android:id="@+id/h5_bt_options"
                    android:layout_width="48dp"
                    android:layout_height="match_parent"
                    android:background="@null"
                    android:gravity="center"
                    android:textSize="23dp"
                    android:visibility="gone"
                    tools:visibility="gone" />
            </FrameLayout>

            <!-- optionmenu1-->
            <FrameLayout
                android:id="@+id/h5_nav_options1"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_centerVertical="true"
                android:layout_marginLeft="6dp"
                android:layout_toLeftOf="@id/h5_nav_options"
                android:visibility="gone"
                tools:visibility="visible">

                <ImageButton
                    android:id="@+id/h5_bt_image1"
                    android:layout_width="32dp"
                    android:layout_height="32dp"
                    android:layout_gravity="center|right"
                    android:layout_marginRight="12dp"
                    android:background="@android:color/transparent"
                    android:padding="4dp"
                    android:scaleType="fitCenter"
                    android:visibility="gone"/>

                <TextView
                    android:id="@+id/h5_bt_text1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center|right"
                    android:layout_marginRight="12dp"
                    android:background="@null"
                    android:ellipsize="end"
                    android:maxLength="8"
                    android:singleLine="true"
                    android:textColor="#108ee9"
                    android:textSize="16dp"/>

            </FrameLayout>
        </RelativeLayout>

        <View
            android:id="@+id/h5_h_divider_intitle"
            android:layout_width="match_parent"
            android:layout_height="1px"
            android:background="@android:color/white"
            android:visibility="gone"/>

    </LinearLayout>

</com.alipay.mobile.nebula.view.H5TitleBarFrameLayout>
```

1.3.4.12. 管理 H5 页面

打开 H5 离线包后，您可以选择同步方法或异步方法将单个容器的视图（View）嵌入到原生页面中。

> **说明**
> 使用异步方法不占用主线程，不会影响性能。

- 使用同步方法嵌入单个容器视图到原生页面的方法如下：

```
public static final void openH5(String url) {
    if (TextUtils.isEmpty(url)) {
        return;
    }
    H5Service h5Service = LauncherApplicationAgent.getInstance().getMicroApplicationContext()
            .findServiceByInterface(H5Service.class.getName());
    H5Bundle bundle = new H5Bundle();
    Bundle param = new Bundle();
    // 要打开的离线包 appId
    param.putString(H5Param.APP_ID, appId);
    // 要打开的离线包内的 URL /www/index.html，必须加 /
    // 如果不传 URL，容器将默认打开离线包默认配置的 URL
    param.putString(H5Param.LONG_URL,url);
    bundle.setParams(param);
    if (h5Service != null) {
        H5Page h5Page=h5Service.createPage(activity,bundle);
      View view=h5Page.getContentView(),
     // view 最后添加到自己的页面中就行
    }
}
```

- 使用异步方法嵌入单个容器视图到原生页面的方法如下：

```
H5Service h5Service = LauncherApplicationAgent.getInstance().getMicroApplicationContext()
                .findServiceByInterface(H5Service.class.getName());
H5Bundle bundle = new H5Bundle();
Bundle param = new Bundle();
     param.putString(H5Param.APP_ID, appId);
     param.putString(H5Param.LONG_URL, url);
     bundle.setParams(param);
     if (h5Service != null) {
            h5Service.createPageAsync(activity, bundle, h5PageReadyListener);
     }
```

1.3.4.13. 配置 H5 容器

H5 容器内有许多开关配置，通过修改开关配置，能够改变容器的特定行为，比如可以通过验签配置来开启或关闭离线包签名校验。

修改开关配置有以下三种方式：

- 内置 `custom_config.json` 至 portal 工程或应用主工程 `assets` 目录下的 `config` 文件夹，该方法仅适用于 **10.1.60** 及以上版本。`custom_config.json` 的文件格式如下：

```
[
{
  "value": "NO",
  "key": "h5_shouldverifyapp"
},
{
  "value": "0",
  "key": "TSBS"
}
]
```

- 使用 `H5ExtConfigProvider` 在代码中配置开关，该方法仅适用于 **10.1.60** 以下版本。`H5ExtConfigProvider` 使用说明如下：

```
public class H5ExtConfigProviderImpl implements H5ExtConfigProvider {
  @Override
  public String getConfig(String key) {
      if ("h5_shouldverifyapp".equalsIgnoreCase(key)) {
          return "YES";
      } else if ("TSBS".equalsIgnoreCase(key)) {
          return "0";
      }
      return null;
  }
}
// 建议在启动时调用，全局只生效一个 H5ExtConfigProvider 的实例，以最后设置的实例为准
H5Utils.setProvider(H5ExtConfigProvider.class.getName(), new H5ExtConfigProviderImpl());
```

- 通过 MDS 平台下发开关配置，参见 开关配置管理。

容器开关列表

您可通过下表中的开关，来自定义是否使用对应功能。

开关名称	用途	说明	默认值
`h5_shouldverifyapp`	开启或关闭验签。建议线上开启，当手机被认为是 root 手机时，会强制开启验签，此时开关配置不生效。	YES 表示开启，NO 表示关闭。	YES
`TSBS`	是否使用沉浸式标题栏，仅适用于 Android。	“1”表示使用，“0”表示不使用。 **重要** 此处的“1”和“0”为字符串形式。	1
`h5_remote_debug_host`	真机调试远程服务器地址。	• 如果配置，表示开启了远程真机调试。调试用服务器地址需要配置给 `h5_remote_debug_host` 。 • 如果不配置，即在代码中不体现，表示不使用远程真机进行调试，也无默认值。	-
`androidFallbackNetwork`	是否采用 mPaaS 网络库方式加载 fallback 资源。	YES 表示使用 mpaas 网络库加载 fallback 资源，NO 表示使用系统网络库加载 fallback 资源。	YES
`mp_h5_push_window_use_activity`	调用 pushWindow 时是否强制启动新的 Activity。	YES 表示启用， 其他值表示不启用。	NO

`mp_ta_showOptionMenu`	是否显示小程序右上角选项菜单。 **说明** 此配置仅在发布小程序时选择是否显示右上角菜单时有效。	YES 表示显示，其他值表示不显示。	NO
`mp_ta_showShareMenuItem`	是否显示小程序右上角选项菜单中的分享选项。	YES 表示显示，其他值表示不显示。	NO
`mp_ta_use_orginal_mini_nagivationbar`	小程序是否使用内置导航栏。	YES 表示使用，其他值表示不使用。	YES
`h5_CORSWhiteList`	域名白名单，该域名下的离线资源可被跨域访问。 **说明** 对于在线请求的资源，仍需资源服务端开启正确的跨域设置。	内容格式为 JSON 数组，特殊字符要求转义，示例：`{ "value": "[\"oss-cn-hangzhou.aliyuncs.com\"]", "key": "h5_CORSWhiteList" }`	空
`mp_h5_allow_mix_content`	允许 Mixed Content 模式，开启此模式有安全风险，请慎重考虑先。 **说明** 此开关仅在 10.1.60 版本中支持。	YES 表示允许， NO 表示不允许。	NO

1.3.4.14. 扩展 H5 容器

H5 容器组件提供丰富的扩展功能，为方便用户在更多的场景下使用 H5 容器，本文档通过示例来介绍以下 H5 容器扩展功能。

- H5Plugin 获取 Activity 返回的结果
- 自定义 H5 错误页
- 开启沉浸式状态栏
- 添加第三方 JavaScriptInterface
- 为 H5 容器添加过场动画
- 为 H5 容器的 JSAPI 配置黑名单

H5Plugin 获取 Activity 返回的结果

在刷脸、识别等场景中，都需要启动一个新的 Activity 以获取 Activity 返回的结果，但是在这种场景中，JSAPI 无法直接通过重写 H5Activity 来获取结果。因此在使用 H5 容器时，您需要通过以下方式来获取 Activity 返回的结果：

1. 在自定义的 H5Plugin 中注册 `OnH5ActivityResult` 回调，示例如下：

```
H5ActivityResultManager.getInstance().put(onH5ActivityResult);
```

说明

- `put` 方法不检查重复注册，开发者需自己处理防止重复注册。
- 使用后，需要调用 `remove` 方法移除回调，一般建议在 H5Plugin 的 `onRelease` 方法中移除回调。示例如下：

```
H5ActivityResultManager.getInstance().remove(onH5ActivityResult);
```

2. 用 `startActivityForResult` 的方式启动目的 `Activity`，如可以在自定义的 `H5Plugin` 的 handleEvent 方法中启动，示例如下：

```
public boolean handleEvent(H5Event event, H5BridgeContext context) {
 if ("CustomJSAPI".equals(event.getAction())) {
     if (event.getActivity()!=null){
         Intent intent = new Intent(event.getActivity(), yourDestinationActivity.class);
         event.getActivity().startActivityForResult(intent,requestCode,bundle);
     }
     return true;
 }
 return false;
}
```

> **说明**
> 仅对 H5Activity 的 result 进行回调。

3. 在 `OnH5ActivityResult` 的回调方法中，将结果通过 `H5BridgeContext` 对象传递给前端。

```
public interface OnH5ActivityResult {
     void onGetResult(int requestCode, int resultCode, Intent intent);
}
```

自定义 H5 错误页

当您需要自定义 H5 错误页的时候，请按照以下步骤进行操作：

1. 新建一个 HTML 格式的自定义错误页。

```
 <!doctype html>
 <html lang="zh-cn">

 <head>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width,maximum-scale=1.0,minimum-scale=1.0,user-scalabl
e=no" />
   <meta name="format-detection" content="telephone=no" />
   <title>自定义错误</title>
 </head>

 <body>
     <p>这个页面是一个自定义错误页</p>
 </body>

 </html>
```

2. 实现 `H5ErrorPageView` 。在 `APWebView` 中设置刚才创建的错误页。

```
 public class H5ErrorPageViewImpl implements H5ErrorPageView {
     @Override
     public boolean enableShowErrorPage() {
         // true 表示启动自定义错误页
         return true;
     }
     @Override
     public void errorPageCallback(H5Page h5Page, APWebView view, String errorUrl, int statusCode, St
 ring errorMsg, String subErrorMsg) {
         // 获取错误页的 html，demo 中放到了 raw 中，也可以放在其他地方
         String html = H5ResourceManager.readRawFromResource(R.raw.custom_error,
 LauncherApplicationAgent.getInstance().getApplicationContext().getResources());
         // 将错误页设置给 webview
         view.loadDataWithBaseURL(errorUrl, html, "text/html", "utf-8", errorUrl);
     }
 }
```

3. 注册 `H5ErrorPageView` 。在打开 H5 容器之前，将自定义的 `H5ErrorPageView` 注册给容器。

```
H5Utils.setProvider(H5ErrorPageView.class.getName(),new H5ErrorPageViewImpl());
```

> **说明**
>
> 10.1.68.7 及以上版本的基线支持了新的 `MPH5ErrorPageView` ，方法名和使用方式与 `H5ErrorPageView` 一致，但是方法参数有扩展。

```
/**
 * 自定义网络错误页面接口
 */
public interface MPH5ErrorPageView {
    /**
     * @param h5Page      page 对象
     * @param view        webview 对象
     * @param errorUrl    错误地址
     * @param statusCode  错误码
     * @param errorMsg    错误描述
     * @param subErrorMsg 错误描述 sub
     * @param extInfo     扩展信息，请注意判空
     * @param extObj      扩展类，请注意判空
     * @return true 表示需要展示自定义页面，会走到下面 errorPageCallback 方法
     */
    boolean enableShowErrorPage(H5Page h5Page, APWebView view, String errorUrl, int statusCode, String
errorMsg, String subErrorMsg, Bundle extInfo, Object extObj);
    /**
     * @param h5Page      page对象
     * @param view        webview 对象
     * @param errorUrl    错误地址
     * @param statusCode  错误码
     * @param errorMsg    错误描述
     * @param subErrorMsg 错误描述 sub
     * @param extInfo     扩展信息，请注意判空
     * @param extObj      扩展类，请注意判空
     */
    void errorPageCallback(H5Page h5Page, APWebView view, String errorUrl, int statusCode, String error
Msg, String subErrorMsg, Bundle extInfo, Object extObj);
}
```

开启沉浸式状态栏

若需开启沉浸式状态栏，可根据如下步骤实现：

> **说明**
>
> - 此功能仅在 10.1.60 及以上基线版本中支持。
> - 此方法将设置所有 H5 容器打开的 H5 页面的状态栏颜色，如果对状态栏颜色设置有更复杂的需求，可实现 H5 容器 自定义标题栏。
> - 状态栏颜色设置可在容器标题栏接口的 `openTranslucentStatusBarSupport` 方法中处理，也可在其他地方处理。

1. 在 H5 容器配置 中开启 `TSBS` 。
2. 对于使用内置标题栏的开发者，可实现 `H5TransStatusBarColorProvider` 接口，并通过 `H5Utils.setProvider` 方法将实例设置给 H5 容器，代码示例如下：

```
package com.mpaas.demo.nebula;

import android.graphics.Color;

import com.alipay.mobile.nebula.provider.H5TransStatusBarColorProvider;

public class H5TransStatusBarColorProviderImpl implements H5TransStatusBarColorProvider {
    @Override
    public int getColor() {
        return Color.argb(70, 255, 255, 255);
    }
}
```

添加第三方 JavaScriptInterface

接入方通常会遇到接入第三方页面必须要使用 `JavaScriptInterface` 的问题，可按照以下步骤来支持此场景：

1. 实现插件以拦截三方页面加载事件。
2. 获取 WebView 并注入 JavaScript 对象。

代码示例如下：

```
package com.mpaas.demo.nebula;

import android.text.TextUtils;

import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5Param;
import com.alipay.mobile.h5container.api.H5SimplePlugin;

public class TechFinSitePlugin extends H5SimplePlugin {

    @Override
    public void onPrepare(H5EventFilter filter) {
        super.onPrepare(filter);
        filter.addAction(CommonEvents.H5_PAGE_SHOULD_LOAD_URL);
    }

    @Override
    public boolean interceptEvent(H5Event event, H5BridgeContext context) {
        String action = event.getAction();
        if (CommonEvents.H5_PAGE_SHOULD_LOAD_URL.equals(action)) {
            JSONObject params = event.getParam();
            String url = params.getString(H5Param.LONG_URL);
            if (!TextUtils.isEmpty(url) && url.contains("tech.antfin.com")) {
                event.getH5page().getWebView().addJavascriptInterface(new TechFinJavaScriptInterface(),
"techFinBridge");
            }
        }

        return false;
    }
}
```

> **说明**
>
> 切勿在 `interceptEvent` 方法中返回 `true` ，否则将影响容器加载页面。

```
package com.mpaas.demo.nebula;

import android.webkit.JavascriptInterface;

public class TechFinJavaScriptInterface {

    @JavascriptInterface
    @com.uc.webview.export.JavascriptInterface
    public String whoAmI() {
        return "It is tech fin.";
    }
}
```

> **说明**
>
> 系统内核和 UC 内核使用的注解类不同，必须要兼容这两个注解类。

为 H5 容器添加过场动画

要为 H5 容器添加过场动画，只需在项目的 `res/anim` 文件夹下添加动画资源即可。操作步骤如下：

1. 在项目的 `res` 文件夹下新建 `anim` 文件夹，如有可跳过。
2. 将过程动画的资源文件添加到 `anim` 文件夹。H5 容器根据资源文件的文件名自动识别资源文件，因此资源文件的文件名只能为 `h5_slide_out_right.xml` 、 `h5_slide_out_left.xml` 、 `h5_slide_in_right.xml` 或 `h5_slide_in_left.xml` 。您可以参考以下示例，创建您自己的资源文件。
   - h5_slide_out_right.xml

     ```
     <?xml version="1.0" encoding="utf-8"?>
     <set xmlns:android="http://schemas.android.com/apk/res/android">
     <translate
       android:fromXDelta="0%"
       android:toXDelta="100%"
       android:duration="300" />
     </set>
     ```

   - h5_slide_out_left.xml

     ```
     <?xml version="1.0" encoding="utf-8"?>
     <set xmlns:android="http://schemas.android.com/apk/res/android">
     <translate
       android:fromXDelta="0%"
       android:toXDelta="-100%"
       android:duration="300" />
     </set>
     ```

   - h5_slide_in_right.xml

     ```
     <?xml version="1.0" encoding="utf-8"?>
     <set xmlns:android="http://schemas.android.com/apk/res/android">
     <translate
       android:fromXDelta="100%"
       android:toXDelta="0"
       android:duration="300" />
     </set>
     ```

   - h5_slide_in_left.xml

     ```
     <?xml version="1.0" encoding="utf-8"?>
     <set xmlns:android="http://schemas.android.com/apk/res/android">
     <translate
       android:fromXDelta="-100%"
       android:toXDelta="0%"
       android:duration="300" />
     </set>
     ```

为 H5 容器的 JSAPI 配置黑名单

指定域名调用容器 JSAPI 做权限管控时，可以通过配置黑名单的方式处理。具体步骤如下：

1. 继承 `H5JSApiPermissionProvider` 类，重写 `hasDomainPermission` 方法。 `hasDomainPermission` 方法两个入参分别为 `action` 和 `url` 。 `action` 表示自定义 JSAPI 的事件名称， `url` 表示当前页面访问的域名地址。返回值为 `boolean` 类型。 `true` 表示可以处理该事件， `false` 表示无权限处理该事件。以下为 Demo 代码，仅供参考。

```
public class H5JSApiPermissionProviderImpl implements H5JSApiPermissionProvider {
     private static final List blackList = new ArrayList<String>();
     static {
     // 在黑名单列表中的 url 将无权限执行 JSAPI 等相关事件
     blackList.add("https://mcube-prod.cn-hangzhou.oss.aliyuncs.com/ONEX4B905F1032156-
MUAT/20210728/0.0.0.1_all/nebula/fallback/www/index.html");
     }

     @override
     public boolean hasDomainPermission(String action, String url) {
         // 接入者可以根据 action 名称和 url 来判断是否有权限执行当前的 action。
         // action 就是定义的 JSAPI 事件，返回 true 代表可以处理该事件，返回 false 代表无权限处理该事件
         if (blackList.contains(url)) {
             return false;
         }
         return true;
     }
     @override
     public boolean hasThisPermission(String permission, String url) {
         return true;
     }
}
```

2. 在框架初始化完成之后设置 `Provider` 。

```
H5Utils.setProvider(H5JSApiPermissionProvider.class.getName(), new H5JSApiPermissionProviderImpl());
```

1.3.4.15. H5 容器拦截物理按键

使用 H5 容器拦截物理按键功能，请将 mPaaS 基线版本升级至 10.1.68.33 及以上。通过如下代码设置物理返回键拦截的 Provider。

```
public interface MPH5OnKeyDownProvider {
    boolean needIntercept(H5Page page, int keyCode, KeyEvent intent);
    boolean onKeyDown(H5Page page, int keyCode, KeyEvent intent);
}
```

若需要执行拦截操作，将 `needIntercept` 返回 `true` ，自动执行 `onKeyDown` 方法，处理按键逻辑，不再执行 mPaaS 物理返回键逻辑。若 `needIntercept` 返回 `false` ，表示将物理返回键事件交还给原有的 mPaaS 逻辑处理，不再执行 `onKeyDown` 方法。

> **重要**
> 执行 `needIntercept` 方法和 `onKeyDown` 方法时，需要对其参数值做判空操作。

1.3.4.16. H5 容器资源拦截

App 页面里图片展示过多，加载速度缓慢，需要优化 H5 容器的加载速度，争取实现 H5 页面秒开。通过拦截替换 H5 容器中加载的资源文件，替换成下载好的本地文件，无需网上加载，即可大大提升 H5 页面的打开效率。以下方法实现为 Demo 代码，仅供参考。

1. 继承 H5 容器提供的 `H5ResProvider` 类，并重写 `contains` 和 `getResource` 方法。

```
//return true 拦截(取本地资源),false 不拦截(网络加载)
 @Override
 public boolean contains(String sourceUrl) {
     if (isCache(sourceUrl)) {
         if (ResourceCache.contains(sourceUrl)) {
             LoggerFactory.getTraceLogger().debug(TAG, "contains: " + sourceUrl);
             //不拦截
             return true;
         } else {
             ResourceCache.download(sourceUrl);
             return false;
         }
     }
     return false;
 }
```

```
@Override
 public InputStream getResource(String sourceUrl) {
     //从本地缓存中获取资源
     if (isCache(sourceUrl)) {
         if (ResourceCache.contains(sourceUrl)) {
             try {
                 InputStream inputStream = ResourceCache.getResource(sourceUrl);
                 if (null == inputStream) {
                     LoggerFactory.getTraceLogger().debug(TAG, "File null: " + sourceUrl);
                     return new URL(sourceUrl).openStream();
                 }
                 LoggerFactory.getTraceLogger().debug(TAG, "getResource: " + sourceUrl);
                 return inputStream;
             } catch (Exception e) {
             }
         }
     } else {
         //从网络链接获资源
         try {
             return new URL(sourceUrl).openStream();
         } catch (IOException e) {
             e.printStackTrace();
         }
     }
     return null;
 }
```

2. 注册 `H5ResProvider` 。

```
public static void register() {
     H5Utils.setProvider(H5ResProvider.class.getName(), new GapResProvider());
}
```

通过自定义 `H5ResProvider` ，用户可以决定是否拦截加载的资源和资源获取方式（本地读取、网络链接读取），用户可自定义实现自己的业务场景。

1.3.4.17. MPH5OpenFileChooserProvider 的使用

使用场景

在 H5 离线包中使用 `<input type="file"/>` 选择文件或者图片时，若不满足需求，可以使用自定义图片选择功能。如选择图片样式、可选多张图片、动态权限的处理等，推荐使用 `MPH5OpenFileChooserProvider` 来实现。

实现方式

1. 在 mPaaS 初始化完成后的回调中注册 `MPH5OpenFileChooserProvider` 。

```
H5Utils.setProvider(MPH5OpenFileChooserProvider.class.getName(),
                    new MPH5OpenFileChooserProvider() {
                        @Override
                        public boolean needIntercept(Activity activity, ValueCallback valueCallback, b
oolean b, APFileChooserParams apFileChooserParams) {
                            //不需要自定义图片选择功能时，不拦截 H5 容器，return false；默认不拦截 H5 容器
                            //需要自定义图片选择功能时，拦截 H5 容器，return true
                            return true;
                        }

                        @Override
                        public void openFileChooser(Activity activity, ValueCallback valueCallback, bo
olean b, APFileChooserParams apFileChooserParams) {
                            // 拿到合法的 uri 给到前端
                            //必须调用该方法，若不调用，下次单击回调不执行
                            //1.通过intent打开一个Activity，调用系统相册或者拍照
                            //2.拿到选择后的图片
                            //3.关闭页面通过valueCallback，传递给前端
                            valueCallback.onReceiveValue(xx);
                        }
                    });
```

> **重要**
>
> `valueCallback.onReceiveValue(xx)` 函数必须调用，若不调用，下次单击回调不执行。

2. 在回调函数 `openFileChooser` 中选择图片进行处理。
   i. 通过 `intent` 打开一个 `Activity`，调用系统相册或者拍照。
   ii. 获取选择后的图片。
   iii. 关闭打开的 `Activity`，通过 `valueCallback` 传递图片给前端。

参考方案

方案一

1. 在 `openFileChooser` 的回调方法中注册广播接收者。
2. 在 `openFileChooser` 方法中通过 `Intent` 跳转到自己创建的 `Activity`。
3. 调用系统的拍照/图片选择 SDK（可以进行扩展如图片裁剪等功能）。
4. 通过图片选择 `Activity` 获取到最终的选择图片， 发送广播到 `openFileChooser` 回调的广播监听者，关闭 `Activity` 页面。
5. 通过 `WebView` 系统的 `valueCallback.onReceiveValue(xx)` 传递给前端。

方案二

1. 在 `openFileChooser` 方法中通过 `Intent` 跳转到一个 `Activity`，并将 `valueCallback` 传递给 `Activity`。
2. 调用系统的拍照/图片选择 SDK（可以进行扩展如图片裁剪等功能）。
3. 通过 `WebView` 系统的 `valueCallback.onReceiveValue(xx)` 传递给前端。
4. 关闭 `Activity` 页面。

1.3.5. API 参考

1.3.5.1. 10.1.68 基线

本文档介绍 10.1.68 基线下关于 H5 容器和离线包的 Android SDK 的接口详情。

公共函数

H5TitleView

getTitle

- 声明：`String getTitle();`
- 说明：获取主标题文本。
- 参数：无。
- 返回值：String，主标题。

setTitle

- 声明：`void setTitle(String title);`
- 说明：设置主标题文本。
- 参数：

参数	类型	说明
title	String	标题文本

- 返回值：无。

setSubTitle

- 声明：`void setSubTitle(String subTitle);`
- 说明：设置副标题。
- 参数：

参数	类型	说明
subTitle	String	副标题文本

- 返回值：无。

setImgTitle

- 声明：`void setImgTitle(Bitmap imgTitle);`
- 说明：设置图像图标。
- 参数：

参数	类型	说明
imgTitle	Bitmap	图像信息

- 返回值：无。

setImgTitle

- 声明：`void setImgTitle(Bitmap imgTitle,String contentDescription);`
- 说明：设置图像图标和 contentDescription。
- 参数：

参数	类型	说明
imgTitle	Bitmap	图像信息
contentDescription	String	contentDescription 的内容

- 返回值：无。

showCloseButton

- 声明：`void showCloseButton(boolean visible);`
- 说明：是否显示关闭按钮。
- 参数：

参数	类型	说明
visible	boolean	是否显示关闭按钮： 为 true 时，显示关闭按钮 为 false 时，不显示关闭按钮

- 返回值：无。

getContentView

- 声明：`View getContentView();`
- 说明：容器获取标题栏 View。
- 参数：无。
- 返回值：View，标题栏 View。

getContentBgView

- 声明：`ColorDrawable getContentBgView();`
- 说明：容器获取标题栏背景。
- 参数：无。
- 返回值：ColorDrawable，标题栏背景。

getMainTitleView

- 声明：`TextView getMainTitleView();`
- 说明：容器获取主标题 View。
- 参数：无。
- 返回值：TextView，主标题 View。

getSubTitleView

- 声明：`TextView getSubTitleView();`
- 说明：容器获取副标题 View。
- 参数：无。
- 返回值：TextView，副标题 View。

showBackButton

- 声明：`void showBackButton(boolean visible);`
- 说明：设置是否展示返回按钮。
- 参数：

参数	类型	说明
visible	boolean	是否展示返回按钮： 为 true 时，显示返回按钮 为 false 时，不显示返回按钮

- 返回值：无。

showBackHome

- 声明：`void showBackHome(boolean visible);`
- 说明：设置首页按钮可见性。
- 参数：

参数	类型	说明
visible	boolean	是否显示首页按钮： 为 true 时，显示返回首页按钮 为 false 时，不显示返回首页按钮

- 返回值：无。

showOptionMenu

- 声明：`void showOptionMenu(boolean visible);`
- 说明：设置是否显示右上角菜单项。
- 参数：

参数	类型	说明
visible	boolean	是否显示右上角菜单项： 为 true 时，显示菜单 为 false 时，不显示菜单

- 返回值：无。

setOptionType

- 声明：`void setOptionType(H5Param.OptionType type);`
- 说明：设置右上角菜单项展示类型。
- 参数：

参数	类型	说明
type	H5Param.OptionType	菜单展示类型

- 返回值：无。

setOptionType

- 声明：`void setOptionType(H5Param.OptionType type, int num, boolean byIndex);`
- 说明：设置右上角菜单项展示类型。
- 参数：

参数	类型	说明
type	H5Param.OptionType	菜单展示类型
num	int	表示从右往左的第几个 icon，从 0 开始数
byIndex	boolean	是否只对某个菜单项设置展示类型

- 返回值：无。

showTitleLoading

- 声明：`void showTitleLoading(boolean visible);`
- 说明：设置是否在标题栏上显示加载状态，可根据需要选择实现方式。
- 参数：

参数	类型	说明
visible	boolean	是否在标题栏上显示加载状态

- 返回值：无。

setBtIcon

- 声明：`void setBtIcon(Bitmap btIcon, int index);`
- 说明：设置右上角按钮图标。
- 参数：

参数	类型	说明
btIcon	Bitmap	图标
index	int	表示从右往左的第几个 icon，从 0 开始数

- 返回值：无。

setH5Page

- 声明：`void setH5Page(H5Page h5Page);`
- 说明：设置容器的 H5 页面。
- 参数：

参数	类型	说明
h5Page	H5Page	H5 页面

- 返回值：无。

setOptionMenu

- 声明：`void setOptionMenu(JSONObject params);`
- 说明：根据 JS 传递过来的参数设置右上角菜单。
- 参数：

参数	类型	说明
params	JSONObject	JS 参数

- 返回值：无。

getDivider

- 声明：`View getDivider();`
- 说明：容器获取返回按钮和标题内容之间的分割线，可返回空。

- 参数：无。
- 返回值：View，分割线 View。

getHdividerInTitle

- 声明：`View getHdividerInTitle();`
- 说明：容器获取标题栏和 Web 页面间的分割线，不可为空。
- 参数：无。
- 返回值：View，分割线 View。

getPopAnchor

- 声明：`View getPopAnchor();`
- 说明：容器获取下拉菜单弹出位置的锚点 View。
- 参数：无。
- 返回值：View，下拉菜单弹出位置的锚点 View。

resetTitleColor

- 声明：`void resetTitleColor(int color);`
- 说明：容器重置标题栏背景色。
- 参数：

参数	类型	说明
color	int	颜色值

- 返回值：无。

releaseViewList

- 声明：`void releaseViewList();`
- 说明：释放被引用的 View，容器页面销毁时触发此方法。
- 参数：无。
- 返回值：无。

openTranslucentStatusBarSupport

- 声明：`void openTranslucentStatusBarSupport(int color);`
- 说明：容器设置沉浸式标题栏颜色。
- 参数：

参数	类型	说明
color	int	颜色值

- 返回值：无。

setTitleTxtColor

- 声明：`void setTitleTxtColor(int color);`
- 说明：设置标题栏字体颜色。
- 参数：

参数	类型	说明

color	int	颜色值

- 返回值：无。

getOptionMenuContainer

- 声明：`View getOptionMenuContainer();`
- 说明：容器获取右上角菜单 View, 必须是 ViewGroup 及子类。
- 参数：无。
- 返回值：View，右上角菜单 View。

getOptionMenuContainer

- 声明：`View getOptionMenuContainer(int index);`
- 说明：容器根据位置获取右上角菜单 View，必须是 ViewGroup 及子类。
- 参数：

参数	类型	说明
index	int	从右向左的第几个 icon 的位置，从 0 开始数

- 返回值：View，右上角菜单 View。

setBackgroundAlphaValue

- 声明：`void setBackgroundAlphaValue(int alpha);`
- 说明：设置背景透明度。
- 参数：

参数	类型	说明
alpha	int	透明度值

- 返回值：无。

setBackgroundColor

- 声明：`void setBackgroundColor(int color);`
- 说明：设置背景色。
- 参数：

参数	类型	说明
color	int	背景色的值

- 返回值：无。

H5AppCenterPresetProvider

getCommonResourceAppList

- 声明：`Set<String> getCommonResourceAppList();`
- 说明：获取全局资源包。
- 参数：无。
- 返回值：`Set<String>`，全局资源包集合。

getH5PresetPkg

- 声明：`H5PresetPkg getH5PresetPkg();`
- 说明：获取预置资源包。
- 参数：无。
- 返回值：H5PresetPkg，H5 预置资源包。

getTinyCommonApp

- 声明：`String getTinyCommonApp();`
- 说明：获取 tinyApp 的公告资源包的 appId。
- 参数：无。
- 返回值：String，tinyApp 的公告资源包的 appId。

H5Plugin

onPrepare

- 声明：`void onPrepare(H5EventFilter filter);`
- 说明：预准备阶段注册 H5 插件过滤器。
- 参数：

参数	类型	说明
filter	H5EventFilter	H5 插件过滤器

- 返回值：无。

interceptEvent

- 声明：`boolean interceptEvent(final H5Event event, final H5BridgeContext context);`
- 说明：拦截事件。
- 参数：

参数	类型	说明
event	H5Event	H5 事件
context	H5BridgeContext	上下文

- 返回值：boolean，若成功返回 true，否则返回 false。

handleEvent

- 声明：`boolean handleEvent(final H5Event event, final H5BridgeContext context);`
- 说明：是否处理事件。
- 参数：

参数	类型	说明
event	H5Event	可处理的只读事件
context	H5BridgeContext	用于处理一些 JSAPI 相关事件的桥接上下文

- 返回值：boolean，true 表示插件已经处理了它，否则返回 false。

MPNebula

downloadApp

- 声明：`public static void downloadApp(final String appId, final MpaasNebulaDownloadCallback mpaasNebulaDownloadCallback)`
- 说明：下载离线包。
- 参数：

参数	类型	说明
appId	String	离线包 ID
mpaasNebulaDownloadCallback	MpaasNebulaDownloadCallback	下载回调

- 返回值：无。

installApp

- 声明：`public static void installApp(final String appId, final MpaasNebulaInstallCallback mpaasNebulaInstallCallback)`
- 说明：安装离线包。
- 参数：

参数	类型	说明
appId	String	离线包 ID
mpaasNebulaInstallCallback	MpaasNebulaInstallCallback	安装回调

- 返回值：无。

loadOfflineNebula

- 声明：`public static void loadOfflineNebula(String jsonFileName, MPNebulaOfflineInfo… mpNebulaOfflineInfos)`
- 说明：加载预置离线包。
- 参数：

参数	类型	说明
jsonFileName	String	预置离线包 JSON，可从控制台下载
mpNebulaOfflineInfos	MPNebulaOfflineInfo	预置离线包信息

- 返回值：无。

registerH5Plugin

- 声明：`public static void registerH5Plugin(String className, String bundleName, String scope, String[] events)`
- 说明：注册自定义 H5 插件（JSAPI）。
- 参数：

参数	类型	说明
className	String	插件类名，需要全路径（package + class）
bundleName	String	bundle 名（bundle 名可在主 module/build/intermediates/bundle/META-INF/BUNDLE.MF 中查看）
scope	String	范围，通常为 “page”
events	String[]	注册的事件

- 返回值：无。

enableAppVerification

- 声明：`public static void enableAppVerification(final String publicKey)`
- 说明：开启离线包验签。必须在首次打开离线包之前开启离线包验签，否则无法正确设置公钥。
- 参数：

参数	类型	说明
publicKey	String	验签公钥

- 返回值：无。

setCustomViewProvider

- 声明：`public static void setCustomViewProvider(H5ViewProvider viewProvider)`
- 说明：设置容器相关的自定义 View，如标题栏、菜单栏、Web Layout、下拉刷新 View 等。
- 参数：

参数	类型	说明
viewProvider	H5ViewProvider	自定义 view provider

- 返回值：无。

getH5View

- 声明：`public static View getH5View(Activity activity, Bundle param)`
- 说明：获取 H5 容器的视图（view）。
- 参数：

参数	类型	说明
activity	Activity	页面上下文
param	Bundle	启动参数，内部可包含 appid 或是 URL

- 返回值View，H5 容器的视图（view）

getH5ViewAsync

- 声明：`public static void getH5ViewAsync(Activity activity, Bundle param, H5PageReadyListener h5PageReadyListener)`
- 说明：异步获取 H5 容器的视图（view）。
- 参数：

参数	类型	说明
activity	Activity	页面上下文
param	Bundle	启动参数，内部可包含 appid 或是 URL
h5PageReadyListener	H5PageReadyListener	异步回调

- 返回值：无。

AbsTitleView

resetTitle

- 声明：`public abstract void resetTitle();`
- 说明：重置导航栏。
- 参数：无。
- 返回值：无。

H5ViewProvider

全屏 H5 自定义 View 切点。

createTitleView

- 声明：`H5TitleView createTitleView(Context context);`
- 说明：创建自定义标题栏的标题。
- 参数：

参数	类型	说明
context	Context	上下文

- 返回值：H5TitleView，自定义标题栏的标题。

createNavMenu

- 声明：`public H5NavMenuView createNavMenu()`
- 说明：创建自定义导航菜单。
- 参数：无。
- 返回值：H5NavMenuView，自定义的导航菜单。

createPullHeaderView

- 声明：`H5PullHeaderView createPullHeaderView(Context context, ViewGroup viewGroup);`
- 说明：创建自定义下拉刷新的头部样式。
- 参数：

参数	类型	说明
context	Context	上下文

viewGroup	ViewGroup	ViewGroup 控件

- 返回值：H5PullHeaderView，自定义下拉刷新的头部

createWebContentView

- 声明：`H5WebContentView createWebContentView(Context context);`
- 说明：创建 WebView 的承载布局。
- 参数：

参数	类型	说明
context	Context	上下文

- 返回值：H5WebContentView，自定义 WebView 的承载布局。

ITinyOptionMenuView

getView

- 声明：`View getView();`
- 说明：获取当前 View。
- 参数：无。
- 返回值：View，当前界面 View。

setH5Page

- 声明：`void setH5Page(H5Page h5Page);`
- 说明：设置容器的 H5 页面。
- 参数：

参数	类型	说明
h5Page	H5Page	H5 页面

- 返回值：无。

setOptionMenuOnClickListener

- 声明：`void setOptionMenuOnClickListener(View.OnClickListener listener);`
- 说明：设置选项菜单的监听事件。
- 参数：

参数	类型	说明
listener	View.OnClickListener	View 的监听事件

- 返回值：无。

setCloseButtonOnClickListener

- 声明：`void setCloseButtonOnClickListener(View.OnClickListener listener);`
- 说明：设置关闭按钮的监听事件。
- 参数：

参数	类型	说明
listener	View.OnClickListener	View 的监听事件

- 返回值：无。

setCloseButtonOnLongClickListener

- 声明：`void setCloseButtonOnLongClickListener(View.OnLongClickListener listener);`
- 说明：长按关闭按钮的监听事件。
- 参数：

参数	类型	说明
listener	View.OnLongClickListener	View 的监听事件

- 返回值：无。

H5Utils

setProvider

- 声明：`void setProvider(String name, Object provider);`
- 说明：设置 Provider。
- 参数：

参数	类型	说明
name	String	Provider 的名称
provider	Object	Provider 对象

- 返回值：无。

AbsTinyOptionMenuView

onTitleChange

- 声明：`void onTitleChange(H5TitleView title);`
- 说明：响应标题栏变化。
- 参数：

参数	类型	说明
title	H5TitleView	H5 标题栏

- 返回值：无。

H5ReplaceResourceProvider

自定义资源动态加载切点

getReplaceResourcesBundleName

- 声明：`String getReplaceResourcesBundleName();`
- 说明：获取标题栏资源的 bundle 名称。
- 参数：无。

- 返回值：String，标题栏资源的 bundle 名称。

H5ErrorPageView

自定义网络错误页接口

enableShowErrorPage

- 声明：`boolean enableShowErrorPage(H5Page h5Page, APWebView view, String errorUrl, int statusCode, String errorMsg, String subErrorMsg, Bundle extInfo, Object extObj);`
- 说明：是否展示自定义错误页面。
- 参数：

参数	类型	说明
h5Page	H5Page	Page 对象
view	APWebView	WebView 对象
errorUrl	String	错误地址
statusCode	int	错误码
errorMsg	String	错误描述
subErrorMsg	String	错误描述 sub
extInfo	Bundle	扩展信息
extObj	Object	扩展类

- 返回值：boolean 类型，为 true 时表示需要展示自定义页面，会执行 errorPageCallback 方法。

H5BridgeContext

H5 插件返回结果给 Jsapi 请求接口类

sendBridgeResult

- 声明：`boolean sendBridgeResult(JSONObject data);`
- 说明：给 JS 层返回结果。返回一次事件就被消费掉，如需多次使用一个 BridgeContext 返回，则需调用 sendBridgeResultWithCallbackKept。
- 参数：

参数	类型	说明
data	JSONObject	返回给 JS 层的数据

- 返回值：boolean 类型，若成功则返回 true，否则返回 false。

sendToWeb

- 声明：`void sendToWeb(String action, JSONObject param, H5CallBack callback);`
- 说明：给 JS 层返回结果，支持添加回调参数。
- 参数：

参数	类型	说明
action	String	传递数据时的 action
param	JSONObject	需要传递给 JS 层的数据
callback	H5CallBack	-

- 返回值：无。

sendError

- 声明：`boolean sendError(H5Event event, Error code);`
- 说明：发送错误信息。
- 参数：

参数	类型	说明
event	H5Event	H5 事件
code	Error	错误信息

- 返回值：boolean 类型，为 true 时表示传递成功，否则为 false。

sendSuccess

- 声明：`void sendSuccess();`
- 说明：用于无返回值的 H5 回调。
- 参数：无。
- 返回值：无。

sendError

- 声明：`void sendError(int error, String errorMessage);`
- 说明：自定义返回错误码和返回文案。
- 参数：

参数	类型	说明
error	int	返回错误码
errorMessage	String	返回文案

- 返回值：无。

回调函数

errorPageCallback

- 声明：`void errorPageCallback(H5Page h5Page, APWebView view, String errorUrl, int statusCode, String errorMsg, String subErrorMsg, Bundle extInfo, Object extObj)`
- 说明：错误页回调。
- 参数：

参数	类型	说明
h5Page	H5Page	Page 对象
view	APWebView	WebView 对象
errorUrl	String	错误地址
statusCode	int	错误码
errorMsg	String	错误描述
subErrorMsg	String	错误描述 sub
extInfo	Bundle	扩展信息
extObj	Object	扩展类

- 返回值：无。

1.3.5.2. 10.1.60 基线

本文档为您介绍 10.1.60 基线下关于 H5 容器和离线包的 Android SDK 的接口详情。

公共函数

H5TitleView

getTitle

- 声明：`String getTitle();`
- 说明：获取主标题文本。
- 参数：无。
- 返回值：String，主标题。

setTitle

- 声明：`void setTitle(String title);`
- 说明：设置主标题文本。
- 参数：

参数	类型	说明
title	String	标题文本

- 返回值：无。

setSubTitle

- 声明：`void setSubTitle(String subTitle);`
- 说明：设置副标题。
- 参数：

参数	类型	说明
subTitle	String	副标题文本

- 返回值：无。

setImgTitle

- 声明：`void setImgTitle(Bitmap imgTitle);`
- 说明：设置图像图标。
- 参数：

参数	类型	说明
imgTitle	Bitmap	图像信息

- 返回值：无。

setImgTitle

- 声明：`void setImgTitle(Bitmap imgTitle,String contentDescription);`
- 说明：设置图像图标和 contentDescription。
- 参数：

参数	类型	说明
imgTitle	Bitmap	图像信息
contentDescription	String	contentDescription 的内容

- 返回值：无。

showCloseButton

- 声明：`void showCloseButton(boolean visible);`
- 说明：是否显示关闭按钮。
- 参数：

参数	类型	说明
visible	boolean	为 true 时，显示关闭按钮 为 false 时，不显示关闭按钮

- 返回值：无。

getContentView

- 声明：`View getContentView();`
- 说明：容器获取标题栏 View。
- 参数：无。
- 返回值：View，标题栏 View。

getContentBgView

- 声明：`ColorDrawable getContentBgView();`

- 说明：容器获取标题栏背景。
- 参数：无。
- 返回值：ColorDrawable，标题栏背景。

getMainTitleView

- 声明：`TextView getMainTitleView();`
- 说明：容器获取主标题 View。
- 参数：无。
- 返回值：TextView，主标题 View。

getSubTitleView

- 声明：`TextView getSubTitleView();`
- 说明：容器获取副标题 View。
- 参数：无。
- 返回值：TextView，副标题 View。

showBackButton

- 声明：`void showBackButton(boolean visible);`
- 说明：设置是否展示返回按钮。
- 参数：

参数	类型	说明
visible	boolean	为 true 时，显示返回按钮 为 false 时，不显示返回按钮

- 返回值：无。

showBackHome

- 声明：`void showBackHome(boolean visible);`
- 说明：设置首页按钮可见性。
- 参数：

参数	类型	说明
visible	boolean	为 true 时，显示返回按钮 为 false 时，不显示返回按钮

- 返回值：无。

showOptionMenu

- 声明：`void showOptionMenu(boolean visible);`
- 说明：设置是否显示右上角菜单项。
- 参数：

参数	类型	说明
visible	boolean	为 true 时，显示菜单 为 false 时，不显示菜单

- 返回值：无。

setOptionType

- 声明：`void setOptionType(H5Param.OptionType type);`
- 说明：设置右上角菜单项展示类型。
- 参数：

参数	类型	说明
type	H5Param.OptionType	菜单展示类型

- 返回值：无。

setOptionType

- 声明：`void setOptionType(H5Param.OptionType type, int num, boolean byIndex);`
- 说明：设置右上角菜单项展示类型。
- 参数：

参数	类型	说明
type	H5Param.OptionType	菜单展示类型
num	int	从右往左第几个 icon，从 0 开始
byIndex	boolean	是否只对某个菜单项设置展示类型

- 返回值：无。

showTitleLoading

- 声明：`void showTitleLoading(boolean visible);`
- 说明：设置是否在标题栏上显示加载状态，可根据需要选择实现方式。
- 参数：

参数	类型	说明
visible	boolean	是否在标题栏上显示加载状态

- 返回值：无。

setBtIcon

- 声明：`void setBtIcon(Bitmap btIcon, int index);`
- 说明：设置右上角按钮图标。
- 参数：

参数	类型	说明
btIcon	Bitmap	图标
index	int	从右往左第几个 icon，从 0 开始

- 返回值：无。

setH5Page

- 声明：`void setH5Page(H5Page h5Page);`
- 说明：设置容器的 H5 页面。
- 参数：

参数	类型	说明
h5Page	H5Page	H5 页面

- 返回值：无。

setOptionMenu

- 声明：`void setOptionMenu(JSONObject params);`
- 说明：根据 JS 传递过来的参数设置右上角菜单。
- 参数：

参数	类型	说明
params	JSONObject	JS 参数

- 返回值：无。

getDivider

- 声明：`View getDivider();`
- 说明：容器获取返回按钮和标题内容之间的分割线，可返回空。
- 参数：无。
- 返回值：View，分割线 View。

getHdividerInTitle

- 声明：`View getHdividerInTitle();`
- 说明：容器获取标题栏和 Web 页面间的分割线，不可为空。
- 参数：无。
- 返回值：View，分割线 View。

getPopAnchor

- 声明：`View getPopAnchor();`
- 说明：容器获取下拉菜单弹出位置的锚点 View。
- 参数：无。
- 返回值：View，下拉菜单弹出位置的锚点 View。

resetTitleColor

- 声明：`void resetTitleColor(int color);`
- 说明：容器重置标题栏背景色。
- 参数：

参数	类型	说明
color	int	颜色值

- 返回值：无。

releaseViewList

- 声明：`void releaseViewList();`
- 说明：释放被引用的 View，容器页面销毁时触发此方法。
- 参数：无。
- 返回值：无。

openTranslucentStatusBarSupport

- 声明：`void openTranslucentStatusBarSupport(int color);`
- 说明：容器设置沉浸式标题栏颜色。
- 参数：

参数	类型	说明
color	int	颜色值

- 返回值：无。

setTitleTxtColor

- 声明：`void setTitleTxtColor(int color);`
- 说明：设置标题栏字体颜色。
- 参数：

参数	类型	说明
color	int	颜色值

- 返回值：无。

getOptionMenuContainer

- 声明：`View getOptionMenuContainer();`
- 说明：容器获取右上角菜单 View，必须是 ViewGroup 及其子类。
- 参数：无。
- 返回值：View，右上角菜单 View。

getOptionMenuContainer

- 声明：`View getOptionMenuContainer(int index);`
- 说明：容器根据位置获取右上角菜单 View，必须是 ViewGroup 及其子类。
- 参数：

参数	类型	说明
index	int	从右向左 icon 的位置，从 0 开始

- 返回值：View，右上角菜单 View。

setBackgroundAlphaValue

- 声明：`void setBackgroundAlphaValue(int alpha);`
- 说明：设置背景透明度。
- 参数：

参数	类型	说明
alpha	int	透明度值

- 返回值：无。

setBackgroundColor

- 声明：`void setBackgroundColor(int color);`
- 说明：设置背景色。
- 参数：

参数	类型	说明
color	int	背景色的值

- 返回值：无。

H5AppCenterPresetProvider

getCommonResourceAppList

- 声明：`Set<String> getCommonResourceAppList();`
- 说明：获取全局资源包。
- 参数：无。
- 返回值：`Set<String>`，全局资源包集合。

getH5PresetPkg

- 声明：`H5PresetPkg getH5PresetPkg();`
- 说明：获取预置资源包。
- 参数：无。
- 返回值：H5PresetPkg，H5 预置资源包。

getTinyCommonApp

- 声明：`String getTinyCommonApp();`
- 说明：获取 tinyApp 的公告资源包的 appId。
- 参数：无。
- 返回值：String，tinyApp 的公告资源包的 appId。

H5Plugin

onPrepare

- 声明：`void onPrepare(H5EventFilter filter);`
- 说明：预准备阶段注册 H5 插件过滤器。
- 参数：

参数	类型	说明
filter	H5EventFilter	H5 插件过滤器

- 返回值：无。

interceptEvent

- 声明：`boolean interceptEvent(final H5Event event, final H5BridgeContext context);`

- 说明：拦截事件。
- 参数：

参数	类型	说明
event	H5Event	H5 事件
context	H5BridgeContext	上下文

- 返回值：boolean，若成功返回 true，否则返回 false。

handleEvent

- 声明：`boolean handleEvent(final H5Event event, final H5BridgeContext context);`
- 说明：是否发送事件。
- 参数：

参数	类型	说明
event	H5Event	可处理的只读事件
context	H5BridgeContext	用于处理一些 JSAPI 相关事件的桥接上下文

- 返回值：boolean，`true` 表示插件已经处理了它，否则返回 `false`。

H5ReplaceResourceProvider

自定义资源动态加载切点

getReplaceResourcesBundleName

- 声明：`String getReplaceResourcesBundleName();`
- 说明：获取标题栏资源的 bundle 名称。
- 参数：无。
- 返回值：String，标题栏资源的 bundle 名称。

H5ErrorPageView

说明：自定义网络错误页接口

enableShowErrorPage

- 声明：`boolean enableShowErrorPage(H5Page h5Page, APWebView view, String errorUrl, int statusCode, String errorMsg, String subErrorMsg, Bundle extInfo, Object extObj);`
- 说明：是否展示自定义错误页。
- 参数：

参数	类型	说明
h5Page	H5Page	Page 对象
view	APWebView	WebView 对象
errorUrl	String	错误地址

statusCode	int	错误码
errorMsg	String	错误描述
subErrorMsg	String	错误描述 sub
extInfo	Bundle	扩展信息
extObj	Object	扩展类

- 返回值：boolean，true 表示需要展示自定义页面，会执行 errorPageCallback 方法。

H5BridgeContext

H5 插件返回结果给 JSAPI 请求接口类

sendBridgeResult

- 声明：`boolean sendBridgeResult(JSONObject data);`
- 说明：给 js 层返回结果。每次返回一个事件即被消费掉，如需多次使用一个 BridgeContext 返回，则需调用 sendBridgeResultWithCallbackKept。
- 参数：

参数	类型	说明
data	JSONObject	返回给 js 层的数据

- 返回值：boolean，若成功返回 true，否则返回 false。

sendToWeb

- 声明：`void sendToWeb(String action, JSONObject param, H5CallBack callback);`
- 说明：给 js 层返回结果，支持添加回调参数。
- 参数：

参数	类型	说明
action	String	传递数据时的 action
param	JSONObject	需要传递给 js 层的数据
callback	H5CallBack	-

- 返回值：无。

sendError

- 声明：`boolean sendError(H5Event event, Error code);`
- 说明：发送错误信息。
- 参数：

参数	类型	说明

event	H5Event	H5 事件
code	Error	错误信息

- 返回值：boolean，true 表示传递成功，否则为 false。

sendSuccess

- 声明：`void sendSuccess();`
- 说明：用于无返回值的 H5 回调。
- 参数：无。
- 返回值：无。

sendError

- 声明：`void sendError(int error, String errorMessage);`
- 说明：自定义返回错误码和返回文案。
- 参数：

参数	类型	说明
error	int	返回错误码
errorMessage	String	返回文案

- 返回值：无。

回调函数

errorPageCallback

- 声明：`void errorPageCallback(H5Page h5Page, APWebView view, String errorUrl, int statusCode, String errorMsg, String subErrorMsg, Bundle extInfo, Object extObj)`
- 说明：错误页回调。
- 参数：

参数	类型	说明
h5Page	H5Page	Page 对象
view	APWebView	WebView 对象
errorUrl	String	错误地址
statusCode	int	错误码
errorMsg	String	错误描述
subErrorMsg	String	错误描述 sub
extInfo	Bundle	扩展信息

extObj	Object	扩展类

- 返回值：无。

1.4. 接入 iOS

1.4.1. 快速开始

本文介绍如何将 H5 容器和离线包组件接入到 iOS 客户端。H5 容器和离线包支持 **基于 mPaaS 框架接入**、**基于已有工程且使用 mPaaS 插件接入** 和 **基于已有工程且使用 CocoaPods 接入** 三种接入方式。通过使用 H5 容器和离线包可以实现初始化容器、唤起容器、实现 H5 与 Native 的双向通信、加载和使用离线包、集成 Nebula 容器自动化埋点能力及查看埋点数据等功能。

前置条件

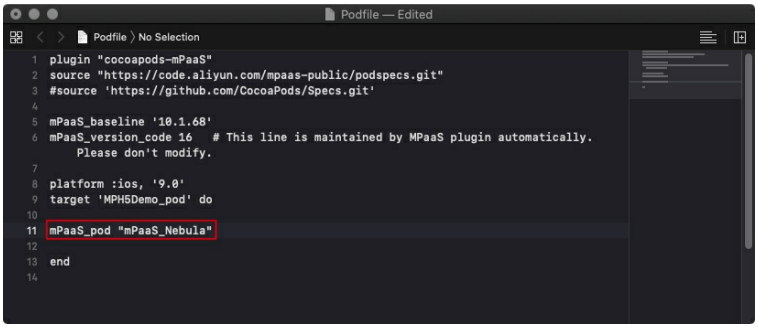
您已经接入工程到 mPaaS。更多信息，请参见以下内容：

- 基于 mPaaS 框架接入
- 基于已有工程且使用 mPaaS 插件接入
- 基于已有工程且使用 CocoaPods 接入

添加 SDK

根据您采用的接入方式，请选择相应的添加方式。

- 使用 mPaaS Xcode Extension。此方式适用于采用了 **基于 mPaaS 框架接入** 或 **基于已有工程且使用 mPaaS 插件接入** 的接入方式。
  i. 点击 Xcode 菜单项 **Editor** > **mPaaS** > **编辑工程**，打开编辑工程页面。
  ii. 选择 **H5 容器&离线包**，保存后点击 **开始编辑**，即可完成添加。
- 使用 cocoapods-mPaaS 插件。此方式适用于采用了 **基于已有工程且使用 CocoaPods 接入** 的接入方式。
  i. 在 Podfile 文件中，使用 `mPaaS_pod "mPaaS_Nebula"` 添加 H5 容器组件依赖。



  ii. 执行 `pod install` 即可完成接入。

使用 SDK

本文将结合 H5 容器和离线包 官方 Demo 介绍如何在 10.1.60 及以上版本的基线中使用 H5 容器 SDK。

H5 容器 SDK 的整个使用过程主要分为以下 5 步：

1. 初始化配置
2. 唤起容器
3. 实现 H5 与 Native 双向通信
4. 加载离线包
5. H5 容器埋点

1 初始化配置

1.1 初始化容器

- 为了使用 Nebula 容器，您需要在程序启动完成后调用 SDK 接口，对容器进行初始化。初始化必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中进行。在 mPaaS 框架提供的分类 `DTFrameworkInterface +（项目工程名）` 中重写此方法。

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 初始化容器
    [MPNebulaAdapterInterface initNebula];
}
```

- 若您需要使用 **预置离线包**、**自定义 JSAPI** 和 **Plugin** 等功能，请将上方代码中的 `initNebula` 接口替换为下方代码中的 `initNebulaWith` 接口，传入对应参数对容器进行初始化。
  - presetApplistPath：自定义的预置离线包的包信息路径。
  - appPackagePath：自定义的预置离线包的包路径。
  - pluginsJsapisPath：自定义 JSAPI 和 Plugin 文件的存储路径。

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 初始化容器
    NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"MPCustomPresetApps.bundle/h5_json.json"] ofType:nil];
    NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"MPCustomPresetApps.bundle"] ofType:nil];
    NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
    [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:pluginsJsapisPath];
}
```

> **说明**
> `initNebula` 和 `initNebulaWithCustomPresetApplistPath` 是两个并列的方法，不支持同时调用。

- 配置小程序包请求时间间隔，mPaaS 支持配置小程序包的请求时间间隔，可全局配置或单个配置。
  - **全局配置**：您可以在初始化容器时通过如下代码设置离线包或小程序的更新频率。

```
[MPNebulaAdapterInterface shareInstance].nebulaUpdateReqRate = 7200;
```

其中 7200 是设置全局更新间隔的值，7200 为默认值，代表间隔时长，单位为秒，您可修改此值来设置您的全局离线包请求间隔，范围为 0 ~ 86400 秒（即 0 ~ 24 小时，0 代表无请求间隔限制）。

  - **单个配置**：即只对当前小程序包配置。可在控制台中前往 **单个配置** > **添加离线包** > **扩展信息** 中填入 `{"asyncReqRate":"1800"}` 来设置请求时间间隔。详情参见 创建 H5 离线包 中的 **扩展信息**。

1.2 定制容器

- 如有需要，您可以通过设置 `MPNebulaAdapterInterface` 的属性值来定制容器配置。必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中设置，否则会被容器默认配置覆盖。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 定制容器
    [MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
    [MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
    [MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
    [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];
}
```

- 属性含义如下：

属性	含义	备注
nebulaVeiwControllerClass	H5 页面的基类	默认为 H5WebViewController。若需指定所有 H5 页面的基类，可直接设置此接口。 **重要** 基类必须继承自 H5WebViewController。
nebulaWebViewClass	设置 WebView 的基类	> 10.1.60：默认为 H5WKWebView。自定义的 WebView 必须继承 H5WKWebView。 = 10.1.60：不支持自定义。
nebulaUseWKArbitrary	设置是否使用 WKWebView 加载离线包页面	> 10.1.60：默认为 YES。 = 10.1.60：默认为 NO。
nebulaUserAgent	设置应用的 UserAgent	设置的 UserAgent 会作为后缀添加到容器默认的 UA 上。
nebulaNeedVerify	是否验签，默认为 YES	若 配置离线包 时未上传私钥文件，此值需设为 NO，否则离线包加载失败。
nebulaPublicKeyPath	离线包验签的公钥路径	与 配置离线包 时上传的私钥对应的公钥路径。
nebulaCommonResourceAppList	公共资源包的 appId 列表	-
errorHtmlPath	当 H5 页面加载失败时展示的 HTML 错误页路径	默认读取 `MPNebulaAdapter.bundle/error.html`。
configDelegate	设置自定义开关 delegate	提供全局修改容器默认开关值的能力。

1.3 更新离线包

启动完成后，全量请求所有离线包信息，检查服务端是否有更新包。为了不影响应用启动速度，建议在 `(void)application:(UIApplication \*)application afterDidFinishLaunchingWithOptions:(NSDictionary \*)launchOptions` 之后调用。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)lau
nchOptions
{
    // 定制容器
    [MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
    [MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
    [MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
    [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];

    // 全量更新离线包
    [[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error
) {
        NSLog(@"");
    }];
}
```

初始化完成后，效果如下：

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 初始化 rpc
    [MPRpcInterface initRpc];

    // 初始化容器
    //    [MPNebulaAdapterInterface initNebula];

    // 自定义jsapi路径和预置离线包信息
    NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString
        stringWithFormat:@"DemoCustomPresetApps.bundle/NebulaApplist.plist"] ofType:nil];
    NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle"]
        ofType:nil];
    NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString
        stringWithFormat:@"DemoCustomPlugins.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
    [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath customPresetAppPackagePath:appPackagePath
        customPluginsJsapisPath:pluginsJsapisPath];
}

- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 定制容器
    [MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class]; //设置H5容器基类
    [MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";//设置H5容器UserAgent
    [MPNebulaAdapterInterface shareInstance].nebulaUseWKArbitrary = YES; //开启 WKWebview
    [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];// 设置全局资源包
    [MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO; // 关闭离线包验签，正式版本请开启验签

    // 更新离线包
    [[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {
        NSLog(@"[mpaas] nebula rpc data :%@", data);
    }];
}

@end
```

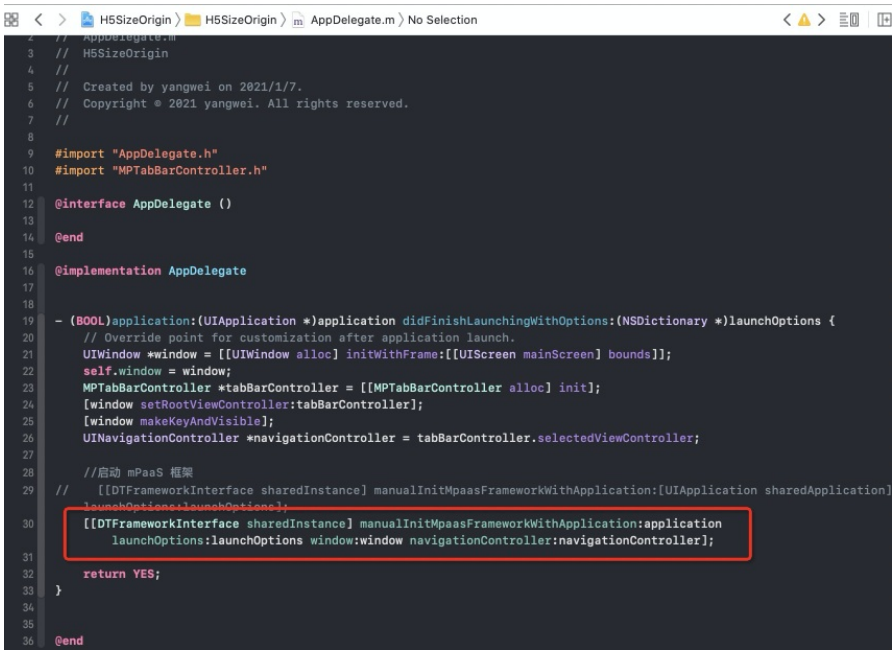
1.4 非框架托管配置

非框架托管是指 App 的生命周期并没有交给 mPaaS 框架托管，而是指定为自定义的 delegate。如果采用了非框架托管的方式（如下图所示），则需进行额外配置，请参照本文提供的方法初始化 mPaaS 框架。如果采用了 mPaaS 框架托管 App 的生命周期，则无需进行额外配置。

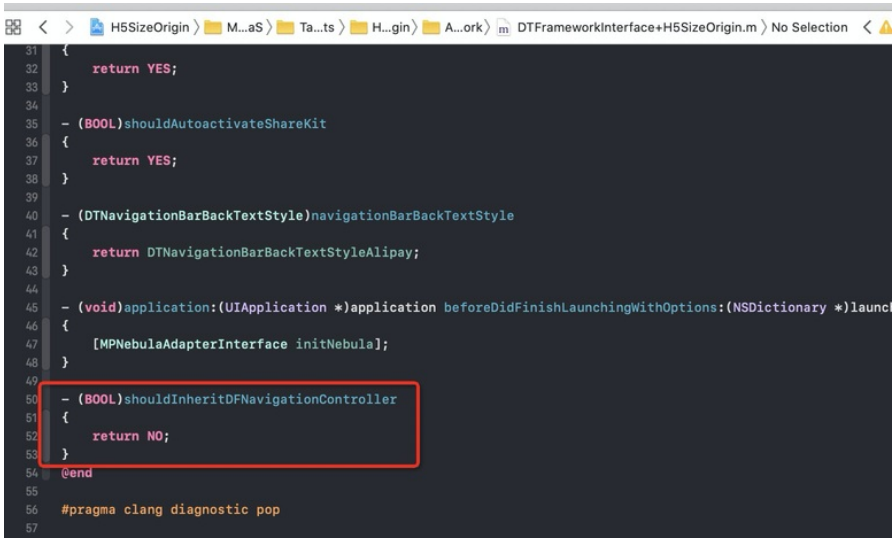
```
//
//  main.m
//  MPH5Demo_pod
//
//  Created by yangwei on 2019/3/26.
//  Copyright © 2019 yangwei. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "AppDelegate.h"

int main(int argc, char * argv[]) {
    [MPAnalysisHelper enableCrashReporterService]; // USE MPAAS CRASH REPORTER
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
//        return UIApplicationMain(argc, argv, @"DFApplication", @"DFClientDelegate"); // NOW USE MPAAS FRAMEWORK
    }
}
```

> **说明**
>
> 如果您的基线版本 ＜10.1.68.25，建议您升级、选用新版本基线。

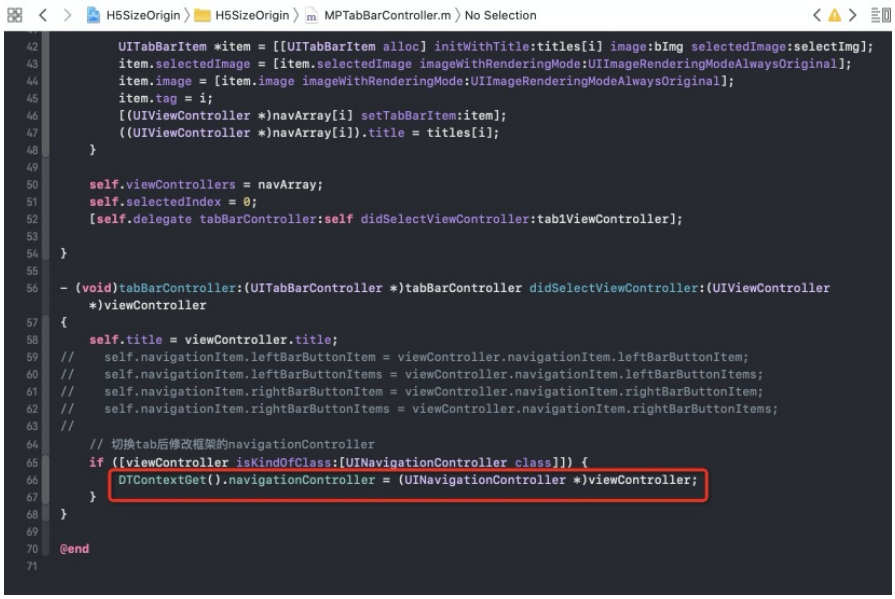
- 只需在应用的 window 及 navigationController 创建完成后，调用以下方法即可，不再需要创建 bootloader、隐藏框架 window 等操作。



- 支持不继承 DFNavigationController。



- 若 App 有多个导航栏，且需要在不同导航栏中打开不同离线包，在切换导航栏后需重新设置容器的导航栏。



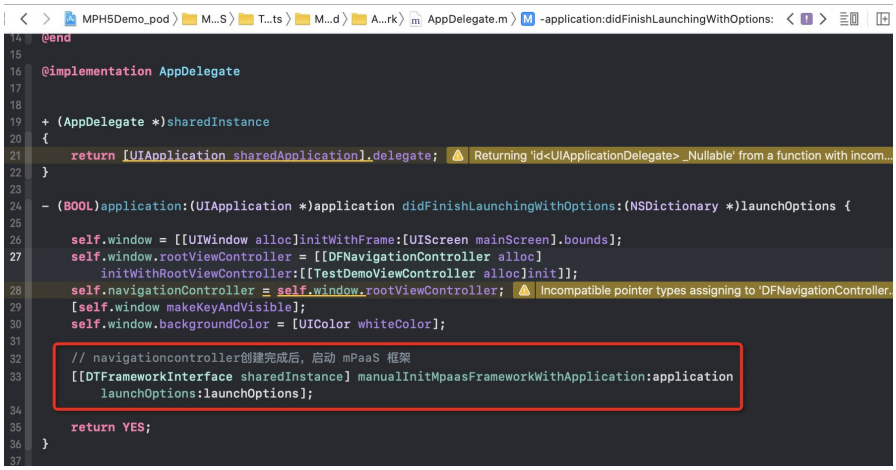
1.5 非框架托管配置（10.1.68.25 以下版本）

非框架托管是指 App 的生命周期并没有交给 mPaaS 框架托管，而是指定为自定义的 delegate。如果采用了非框架托管 App 生命周期的方式（如下图所示），需进行额外配置，请参照本文提供的方法初始化 mPaaS 框架。如果采用了 mPaaS 框架托管 App 的生命周期，则无需进行额外配置。

> **说明**
>
> 如果您的基线版本 ≥10.1.68.25，请参考 **1.4 非框架托管配置** 进行配置。对于 10.1.68.25 及以上版本的基线，**1.4 非框架托管配置** 提供了快速、简洁的初始化框架方法，如果您使用的基线版本仍低于 10.1.68.25，建议您升级、选用 10.1.68.25 或以上版本基线。

1.5.1 启动 mPaaS 框架

在当前应用的 `didFinishLaunchingWithOptions` 方法中调用 `[[DTFrameworkInterface sharedInstance] manualInitMpaasFrameworkWithApplication:application launchOptions:launchOptions];` 来启动 mPaaS 框架。

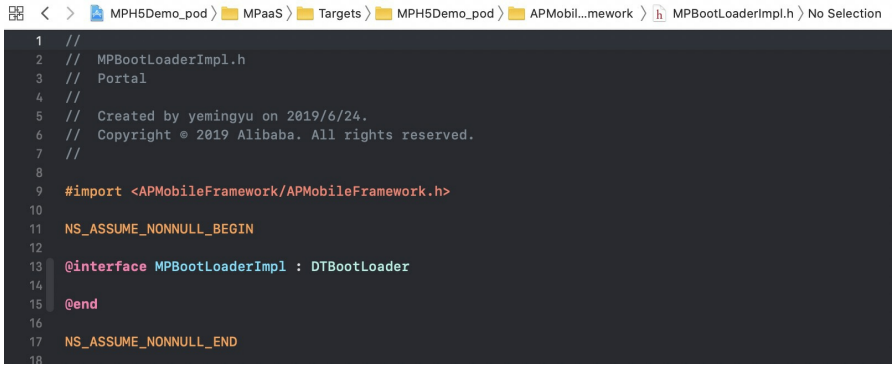


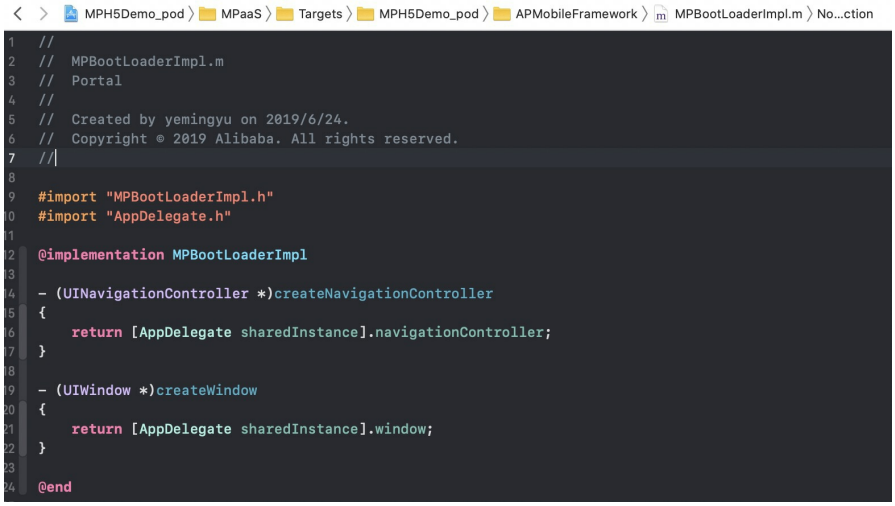
> **说明**
>
> 启动框架必须在当前应用 window 和 navigation 初始化完成后调用，否则无法生效。

1.5.2 创建应用启动器

创建继承 DTBootLoader 的子类，重写 `createWindow` 和 `createNavigationController` 方法，返回当前应用自己的 window 和 navigationControlle。

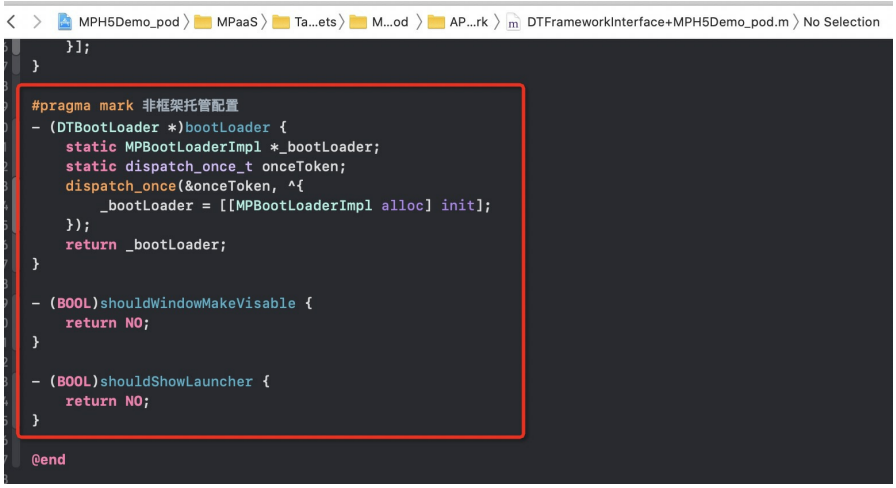
- 设置 window：当前应用的 keyWindow。
- 设置 navigationController：当前应用 keyWindow 的 rootviewcontroller，必须继承 DFNavigationController。





1.5.3 指定应用启动器

在 DTFrameworkInterface 的 category 中重写方法，指定当前应用自己的 bootloader，并隐藏 mPaaS 框架默认的 window 和 launcher 应用。



2 唤起容器

容器初始化完成后，就可以唤起一个 H5 容器。分为以下三种情况。

- 基于在线 URL 或本地 HTML 文件，创建一个 H5 容器。 示例代码如下：

```
// 打开在线 URL
[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url":
@"https://tech.antfin.com/products/MPAAS"}];
// 打开本地 HTML 页面
NSString *path = [[NSBundle mainBundle].bundlePath stringByAppendingFormat:@"/%@/%@",
@"MPH5Demo.bundle", @"H52Native.html"];
if ([path length] > 0) {
[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": path}];
}
```

- 基于传入的离线包信息，创建一个 H5 容器，并自动 push 打开。示例代码如下：

```
  [[MPNebulaAdapterInterface shareInstance]
startH5ViewControllerWithNebulaApp:@{@"appId":@"90000000"}];
```

- 基于传入的离线包信息，创建一个 H5 容器，并返回创建的 H5 容器实例（一般用在首页 tab 页面）。 示例代码如下：

```
  [[MPNebulaAdapterInterface shareInstance]
createH5ViewControllerWithNebulaApp:@{@"appId":@"90000000"}];
```

3 实现 H5 与 Native 双向通信

您可以通过调用 JSAPI 和监听特定事件来实现 H5 与 Native 双向通信。

3.1 在 H5 页面调用 Native 功能

您可以通过调用 JSAPI 来实现 H5 到 Native 的通信。

Nebula 容器支持的 JSAPI 及相关参数说明，请参见 内置 JSAPI。

示例

调用 JSAPI 接口 `pushWindow` ，实现“在 H5 页面点击某个按钮时，加载一个新页面”的需求：

```
AlipayJSBridge.call('pushWindow', {
    url: 'https://tech.antfin.com',
    param: {
    readTitle: true,
    defaultTitle: true,
    // ...
    }
}, function(data) {alert('调用结果'+JSON.stringify(data)); });
```

AlipayJSBridge 说明

AlipayJSBridge 是 Nebula 容器自动注入的 JSBridge。在 `Window.onload` 以后，容器会生成一个全局变量 AlipayJSBridge，然后触发 AlipayJSBridgeReady 事件。AlipayJSBridge 注入是一个异步过程，因此需要先监听 AlipayJSBridgeReady 事件再调用接口。

示例代码如下：

```
<h1>bridge 使用方法</h1>

<script>
function ready(callback) {
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}

ready(function(){
  alert('bridge ready');
});
</script>
```

3.2 在 Native 页面调用 H5 功能

您可以通过监听特定事件来实现 Native 到 H5 的通信。Nebula 容器支持的事件，参见 事件扩展列表。

```
document.addEventListener('back', function (e) {
    if(confirm('back已拦截,是否确定返回？')) {
        // do something;
    }
}, false);
```

除了 Nebula 容器默认支持的事件外，您还可以在 Native 端通过以下方式自定义事件让前端来监听。

```
// self: 当前 H5 页面所在的 VC
// data：native 传递给前端的参数
// callBack：前端收到事件后的回调
[self callHandler:@"customEvent" data:@{@"key":@"value"} responseCallback:^(id responseData) {
        NSLog(@"前端收到事件后的回调: %@", responseData);
    }];
```

3.3 扩展 Nebula 容器能力

如果 Nebula 容器提供的基础 H5 页面双向通信能力无法满足需求，您可以对 Nebula 进行扩展开发：

- **JSAPI**：如果您要从 H5 页面发起 Native 功能调用（如显示一个 ActionSheet，或显示联系人对话框），那么您需要扩展 JSAPI。JSAPI 可以通过 handler 方法，让您很方便地增加 H5 页面的 Native 功能调用来实现特定功能。具体的自定义方法，请参见 自定义 JSAPI。
- **Plugin**：如果您需要在某个时机（如进入页面、收到请求等）做某些事情（如记录埋点、修改返回数据等），那么您需要开发一个插件 (Plugin)。在插件中订阅相应的事件后，就可以在 handler 中对事件所携带的数据进行加工处理。具体的自定义方法，请参见 自定义插件。

4 加载离线包

传统的在线 H5 技术容易受到网络环境影响，从而影响 H5 页面的性能。为了最大程度摆脱网络对 H5 页面加载的影响，您可以将不同的业务封装打包成为一个离线包，通过发布平台下发到客户端，对客户端资源进行更新。更多信息，请参见 离线包简介 和 使用离线包。

5 H5 容器埋点

在 H5 页面加载时，Nebula 容器会自动监控加载性能，并捕获相关的行为数据和异常报错数据。更多信息，请参见 H5 容器埋点。

1.4.2. 管理离线包

传统的在线 H5 技术容易受到网络环境影响，从而影响 H5 页面的性能。为此，您可以将不同的业务封装打包成一个离线包，通过发布平台下发对客户端资源进行更新。

本文将引导您管理离线包：

- 生成离线包
- 加载离线包
- 利用全局资源包
- 动态更新离线包

前置条件

您需要确保完成 SDK 添加后，客户端工程已集成 `NebulamPaaSBiz.framework` 。

生成离线包

为了生成 `.amr` 离线包，您需要构建前端 `.zip` 包并在线生成 `.amr` 包。具体的操作步骤，请参见文档 生成离线包。

加载离线包

根据是否在客户端预置离线包，分为以下两种方式：

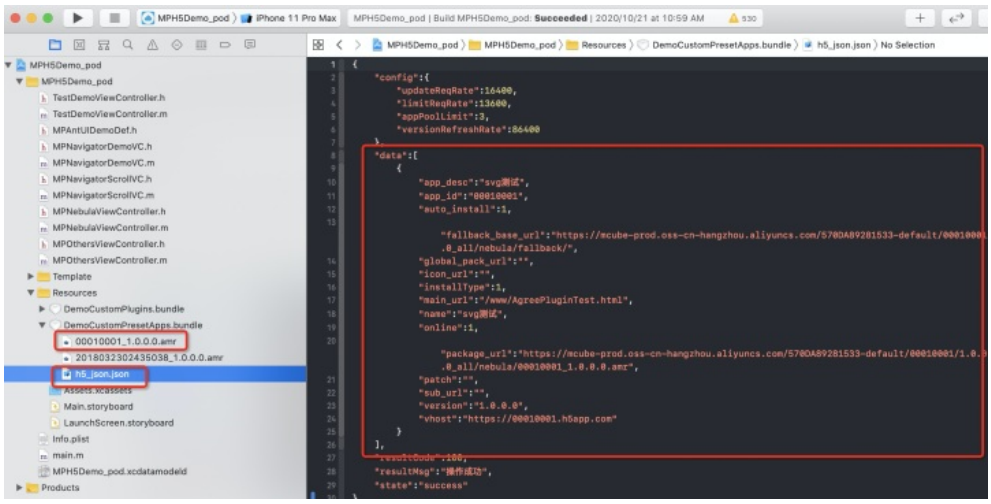
- 预置离线包
- 加载远程离线包

预置离线包

不管网络情况如何，首页或登录页等页面都需快速加载。因此，可以将这部分资源打包为离线包、提前预置在工程中，以保证在离线情况下资源也能快速加载。

具体步骤如下：

1. 新建一个独立的 bundle，如 `DemoCustomPresetApps.bundle` ，将从发布平台下载的 `.amr` 离线包和 `h5_json.json` 文件添加到此 bundle 中。



> **说明**
>
> 目前发布平台仅支持下载单个离线包的 `h5_json.json` 配置文件。当预置多个离线包时，需要手动合并到 JSON 文件中的 data 数组。

2. 在初始化容器时，调用 `initNebulaWithCustomPresetApplistPath` 接口，设置预置离线包路径为上一步中创建的 bundle。

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 初始化 rpc
    [MPRpcInterface initRpc];

    // 初始化容器
    // [MPNebulaAdapterInterface initNebula];

    // 自定义 jsapi 路径和预置离线包信息
    NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle/h5_json.json"] ofType:nil];
    NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle"] ofType:nil];
    NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPlugins.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
    [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:pluginsJsapisPath];
}
```

3. 与加载非预置离线包类似，进入对应的页面时，调用 Nebula 容器提供的接口方法加载离线包。

```
- (void)openPresetPackage {
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithNebulaApp:@{@"appId":@"20180910"}];
}
```

加载远程离线包

除将离线包预置在客户端外，您还可以通过发布平台动态发布一个离线包。这样，客户端就可以直接加载远程离线包，避免了预置大量离线包导致客户端包体积过大的问题。

具体步骤如下：

1. 在应用启动完成后，可以预加载获取包信息并下载离线包，防止打开离线包时白屏。
   - 代码示例

```
[[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {
    NSLog(@"[mpaas] nebula rpc data :%@", data);
}];
```

   - 接口方法

```
@interface MPNebulaAdapterInterface : NSObject

/**
 *  全量更新本地离线包信息
 *
 *  @param finish 完成回调
 *
 */
- (void)requestAllNebulaApps:(NAMRequestFinish)finish;

/**
 *  单个应用请求
 *
 *  @param params   请求列表，格式为{appid:version}。可传多个 appid，版本号最长 4 位，如 1.0.0.1；不指定 version 时传空，默认取最高版本；支持版本号模糊匹配，如： '*' 匹配最高版本号、'1.*' 匹配以 1 开头的版本号中最高的版本号
 *  @param finish   完成回调
 */
- (void)requestNebulaAppsWithParams:(NSDictionary *)params finish:(NAMRequestFinish)finish;

@end
```

2. 客户端配置完成后，您可以通过发布平台下发一个离线包，具体的操作步骤，请参见 实时发布 > 离线包管理 > 发布离线包 。
3. 进入对应的页面时，调用 Nebula 容器提供的接口方法加载离线包，即可看到您在发布平台下发的离线包。
   - 代码示例

```
  - (void)openPresetPackage {
  [[MPNebulaAdapterInterface shareInstance]
startH5ViewControllerWithNebulaApp:@{@"appId":@"20180910"}];
  }
```

   - 接口方法

```
  @interface MPNebulaAdapterInterface : NSObject

  /**
   *  基于传入的离线包信息，创建一个 H5 容器，并自动 push 打开
   *
   *  @param params H5 容器的启动参数，必填参数：appId，其他可选参数参考文档
https://tech.antfin.com/docs/2/85001
   *
   */
  - (void)startH5ViewControllerWithNebulaApp:(NSDictionary *)params;

  @end
```

利用全局资源包

Nebula 全局资源包解决多个 H5 应用使用同一资源产生的冗余问题，如 React 应用使用 ReactJS 框架代码。您可以将公共资源放入全局资源包，以降低 H5 应用体积。您可以在 `afterDidFinishLaunchingWithOptions` 方法中配置全局离线包，如下文示例代码，其中 `77777777` 为全局资源包的 appId。

`nebulaCommonResourceAppList` 用于告知 H5 容器指定 ID 的离线包将作为全局资源包使用。如果您没有配置此 ID，即使内置该离线包，也不会生效。

```
```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)lau
nchOptions
{
    [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];// 设置全局资源
包
}
```
```

通常情况下为了提高页面的加载速度，建议预置全局资源包，后续依然可以通过 H5 应用后台下发更新。

动态更新离线包

mPaaS 提供强大的动态更新能力，您可以通过发布平台下发一个新版本的离线包，来更新客户端的对应页面。具体的操作步骤，请参见 实时发布 > 离线包管理 > 发布离线包。

相关链接

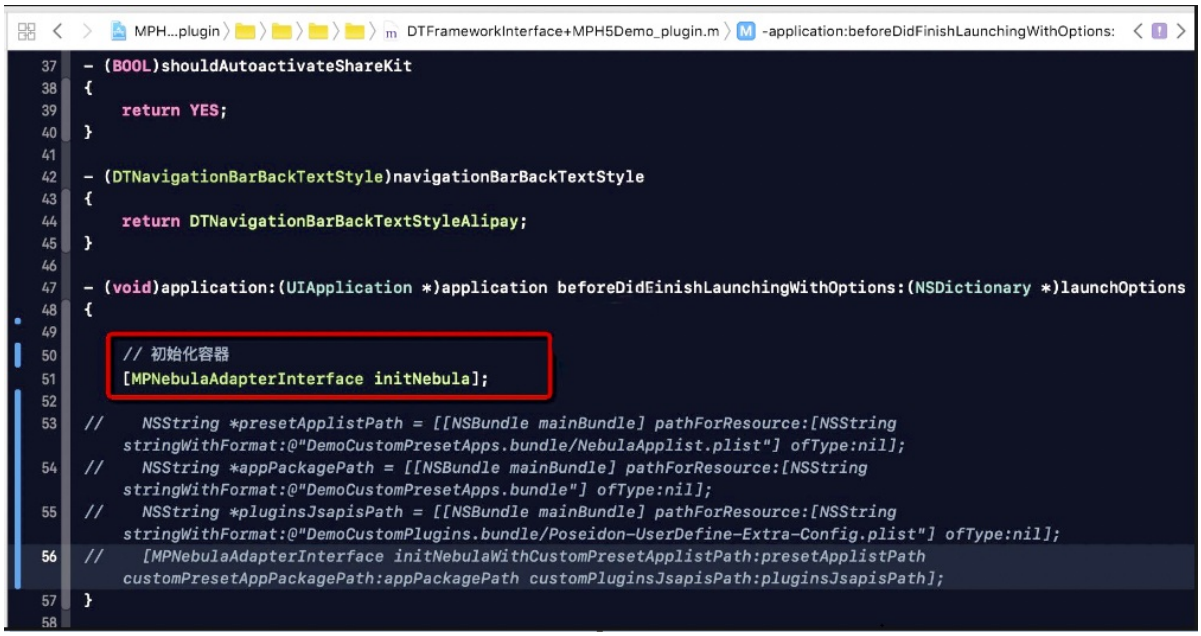
- 离线包介绍
- 代码示例

1.4.3. 10.1.60 升级指南

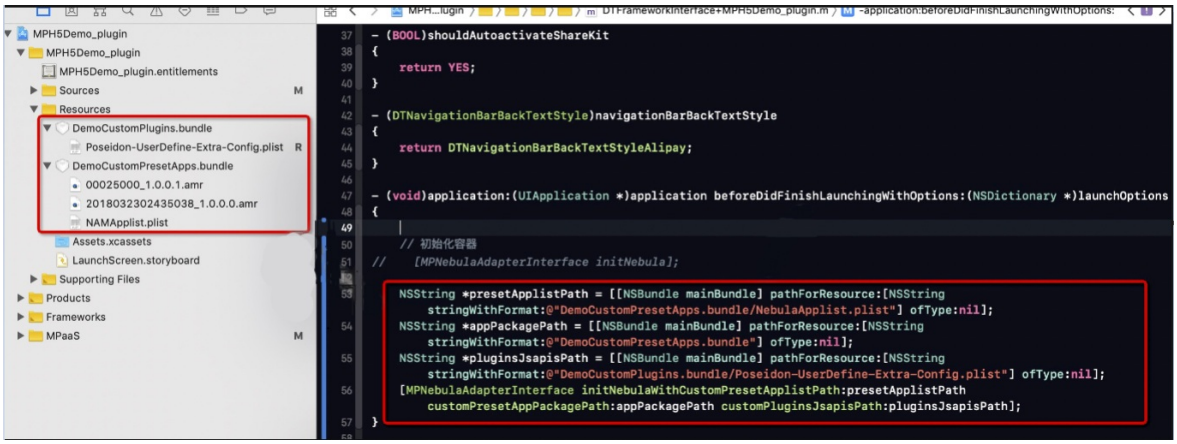
目前，10.1.32 基线已经停止维护。若已有工程为 10.1.32 基线，请根据如下操作升级到 10.1.60 基线。

初始化容器

- 初始化时机：在框架加载之前且必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中调用。



- 若已有工程基线为 10.1.32：
  - 需修改自定义 JSAPI 路径、预置离线包及包信息路径：必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中调用 `initNebulaWithCustomPresetApplistPath`。同时，需要将 `afterDidFinishLaunchingWithOptions` 替换为 `beforeDidFinishLaunchingWithOptions`。

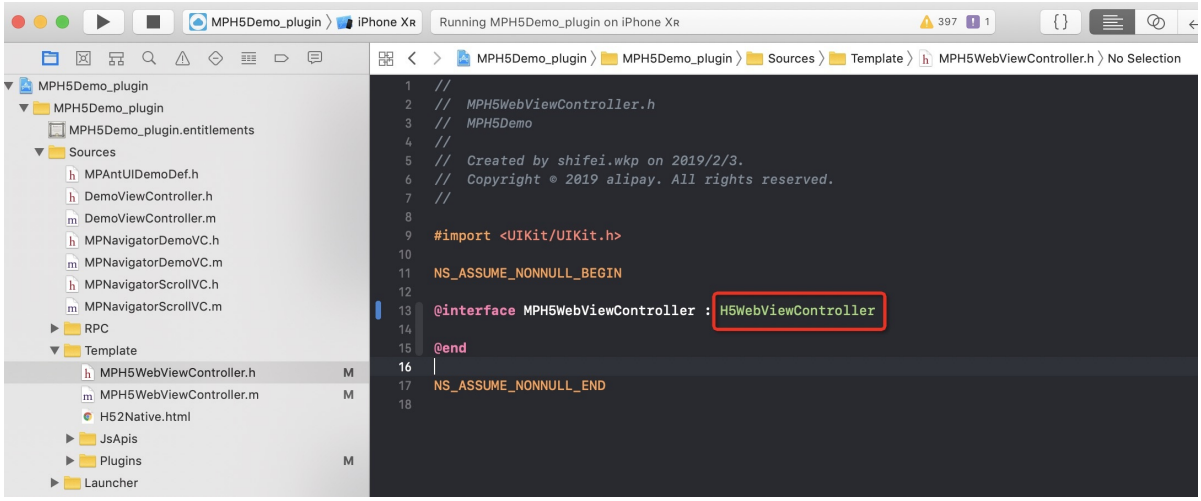


- 需指定所有 H5 页面的基类、全局资源包、UA、是否验签等配置：需在容器初始化之后调用，必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中设置，否则会被容器默认配置覆盖。

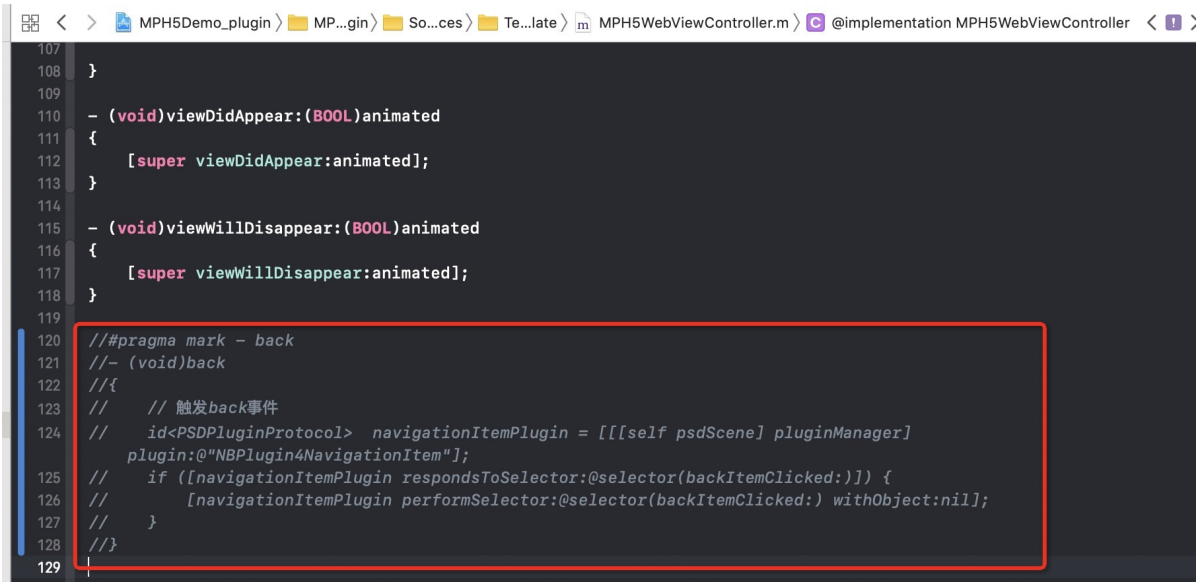
```
78
79      // 启动netctrl
80  //    [[MASSAccess shareInstance] dispatchTokenSvrInfo:nil uploadInfo:nil downloadInfo:nil downloadInfoCrypto:nil
        configStorage:nil];
81
82      // 小程序初始化
83  //    [MPTinyAppAdapterInterface initTinyApp];
84
85      // Nebula容器初始化
86      NSString *customPresetApplistPath = [[NSBundle mainBundle].bundlePath stringByAppendingFormat:@"/%@/%@",
            @"MPH5Demo.bundle", @"h5_json.json"];
87      NSString *customPresetAppPackagePath = [[NSBundle mainBundle].bundlePath stringByAppendingFormat:@"/%@",
            @"MPH5Demo.bundle"];
88      NSString *customPluginsJsapisPath = [[NSBundle mainBundle].bundlePath stringByAppendingFormat:@"/%@/%@",
            @"MPH5Demo.bundle", @"Poseidon-Custom-Config.plist"];
89      [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:customPresetApplistPath
            customPresetAppPackagePath:customPresetAppPackagePath customPluginsJsapisPath:customPluginsJsapisPath];
90  }
91
92  - (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions  Categ...
93  {
94      [MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = NSClassFromString(@"MPH5WebViewController");
95      [MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
96      [MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
97      [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];
98
99      // 全量更新
00      [[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {
01          |
02      }];
03  }
04
```

容器基类

- 自定义的所有 H5 页面的基类，必须是 `H5WebViewController` 的子类。

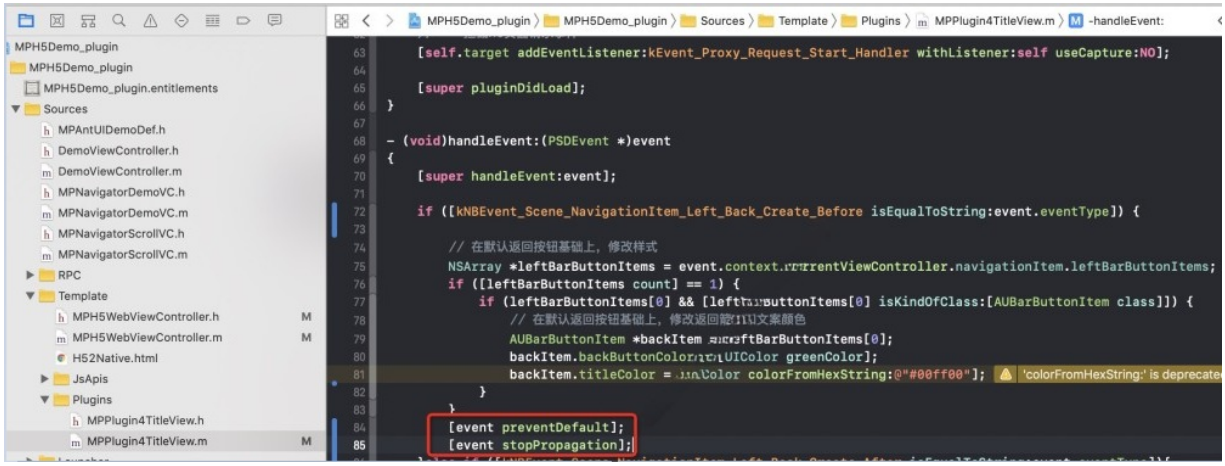


- 若原有 H5 基类中实现了 `back` 方法，需删除。



自定义导航栏

返回按钮需监听 `kNBEvent_Scene_NavigationItem_Left_Back_Create_Before` 事件，修改默认导航栏样式。



1.4.4. 使用容器

1.4.4.1. 使用 SDK（版本 = 10.1.32）

本文将向您介绍在 10.1.32 基线下 H5 容器 SDK 的使用。

> **重要**
> 自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请参考使用 SDK（版本 ≥ 10.1.60）使用 10.1.60 或 10.1.68 基线。

初始化容器

启动容器

- 为了使用 Nebula 容器，您需要在程序启动完成后调用 SDK 接口并对容器进行初始化。初始化必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中进行。

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 初始化容器
    [MPNebulaAdapterInterface initNebula];
}
```

- 若您需要使用 **预置离线包**、**自定义 JSAPI** 和 **Plugin** 等功能，需将上述功能的相关信息分别存储在以下默认的 bundle 中，否则功能将无法生效。

名称	含义
MPCustomPlugins.bundle	自定义的 JSAPI 和 Plugin 路径。
MPCustomPresetApps.bundle	预置的离线包和包信息路径。

定制容器

- 如有需要，您可以通过设置 `MPNebulaAdapterInterface` 的属性值来定制容器配置。必须在 `DTFrameworkInterface` 的 `- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions` 中设置，否则会被容器默认配置覆盖。

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

    // 定制容器
    [MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
    [MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
    [MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
    [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];
}
```

- 属性含义如下：

名称	含义	备注
nebulaVeiwControllerClass	H5 页面的基类	默认为 UIViewController。若需指定所有 H5 页面的基类，可直接设置此接口。
nebulaUserAgent	设置应用的 UserAgent	设置的 UserAgent 会作为后缀添加到容器默认的 UA 上。
nebulaNeedVerify	是否验签，默认为 YES	若 配置离线包 时未上传私钥文件，此值需设为 NO，否则离线包加载失败。
nebulaPublicKeyPath	离线包验签的公钥路径	与 配置离线包 时上传的私钥相对应的公钥路径。
nebulaCommonResourceAppList	公共资源包的 appId 列表	-
errorHtmlPath	当 H5 页面加载失败时展示的 HTML 错误页路径	默认读取 `MPNebulaAdapter.bundle/error.html`。

更新离线包

启动完成后，全量请求所有离线包信息，检查服务端是否有更新包。为了不影响应用启动速度，建议在 `(void)application:(UIApplication \*)application afterDidFinishLaunchingWithOptions:(NSDictionary \*)launchOptions` 之后调用。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)lau
nchOptions
{
    // 初始化容器
    [MPNebulaAdapterInterface initNebula];

    // 定制容器
    [MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
    [MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
    [MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
    [MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];

    // 全量更新离线包
    [[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error
) {
        NSLog(@"");
    }];
}
```

初始化完成后，效果如下：

唤起容器

容器初始化完成后，就可以唤起一个 H5 容器。分为以下三种情况：

- 基于在线 URL 或本地 HTML 文件，创建一个 H5 容器。 示例代码如下：

```
// 打开在线 URL
[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url":
@"https://tech.antfin.com/products/MPAAS"}];
// 打开本地 HTML 页面
NSString *path = [[NSBundle mainBundle].bundlePath stringByAppendingFormat:@"/%@/%@",
@"MPH5Demo.bundle", @"H52Native.html"];
if ([path length] > 0) {
[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": path}];
}
```

- 基于传入的离线包信息，创建一个 H5 容器，并自动 push 打开。示例代码如下：

```
  [[MPNebulaAdapterInterface shareInstance]
startH5ViewControllerWithNebulaApp:@{@"appId":@"90000000"}];
```

- 基于传入的离线包信息，创建一个 H5 容器，并返回创建的 H5 容器实例（一般用在首页 tab 页面）。示例代码如下：

```
  [[MPNebulaAdapterInterface shareInstance]
createH5ViewControllerWithNebulaApp:@{@"appId":@"90000000"}];
```

实现 H5 与 Native 双向通信

您可以通过调用 JSAPI 和监听特定事件来实现 H5 与 Native 双向通信。

在 H5 页面调用 Native 功能

您可以通过调用 JSAPI 来实现 H5 到 Native 的通信。

Nebula 容器支持的 JSAPI 及相关参数说明，请参见 内置 JSAPI。

示例

例如，您可以调用 JSAPI 接口 `pushWindow`，以实现“在 H5 页面单击某个按钮时，加载一个新页面”的需求：

```
AlipayJSBridge.call('pushWindow', {
    url: 'https://tech.antfin.com',
    param: {
    readTitle: true,
    defaultTitle: true,
    // ...
    }
}, function(data) {alert('调用结果'+JSON.stringify(data)); });
```

AlipayJSBridge 说明

`AlipayJSBridge` 是 Nebula 容器自动注入的 JSBridge。在 `Window.onload` 以后，容器会生成一个全局变量 `AlipayJSBridge`，然后触发 `AlipayJSBridgeReady` 事件。`AlipayJSBridge` 注入是一个异步过程，因此需要先监听 `AlipayJSBridgeReady` 事件再调用接口。

示例代码如下：

```
<h1>bridge 使用方法</h1>

<script>
function ready(callback) {
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}

ready(function(){
  alert('bridge ready');
});
</script>
```

在 Native 页面调用 H5 功能

您可以通过监听特定事件来实现 Native 到 H5 的通信。Nebula 容器支持的事件列表，参见 事件扩展。

```
document.addEventListener('back', function (e) {
    if(confirm('back已拦截，是否确定返回？')) {
        // do something;
    }
}, false);
```

除了 Nebula 容器默认支持的事件外，您还可以在 Native 端通过以下方式自定义事件让前端来监听：

```
// self: 当前 H5 页面所在的 VC
// data：native 传递给前端的参数
// callBack：前端收到事件后的回调
[self callHandler:@"customEvent" data:@{@"key":@"value"} responseCallback:^(id responseData) {
        NSLog(@"前端收到事件后的回调: %@", responseData);
    }];
```

扩展 Nebula 容器能力

如果 Nebula 容器提供的基础 H5 页面双向通信能力无法满足需求，您可以对 Nebula 进行扩展开发：

- **JSAPI**：如果要从 H5 页面发起 Native 功能调用（如显示一个 ActionSheet 或显示联系人对话框），那么需要扩展 JSAPI。JSAPI 可以通过 handler 方法，很方便地增加 H5 页面的 Native 功能调用来实现特定功能。具体的自定义方法，请参见 自定义 JSAPI。
- **Plugin**：如果您需要在某个时机（如进入页面、收到请求等）做某些事情（如记录埋点、修改返回数据等），那么需要开发一个插件 (Plugin)。在插件中订阅相应的事件后，即可在 handler 中对事件所携带的数据进行加工处理。具体的自定义方法参见 自定义插件。

加载离线包

传统的在线 H5 技术容易受到网络环境影响，从而影响 H5 页面的性能。为了最大程度摆脱网络对 H5 页面加载的影响，您可以将不同的业务封装打包成为一个离线包，通过发布平台下发到客户端，对客户端资源进行更新。更多信息，参见 离线包简介 和 使用离线包。

H5 容器埋点

在 H5 页面加载时，Nebula 容器会自动监控加载性能，并捕获相关的行为数据和异常报错数据。更多信息，参见 H5 容器埋点。

1.4.5. 进阶指南

1.4.5.1. JSAPI 鉴权方法设置

mPaaS 建议所有的 JSAPI 访问都需要添加访问控制，目前可以通过设置 Plugin 的方式添加访问控制。

1. 设置自定义权限控制 Plugin。
   i. 自定义 Plugin，监听 JSAPI 调用的事件，进行拦截处理。
   ii. 在 Plugin 中拦截到事件后，获取当前页面的 URL，建议根据 host、scheme 进行字符串匹配校验。

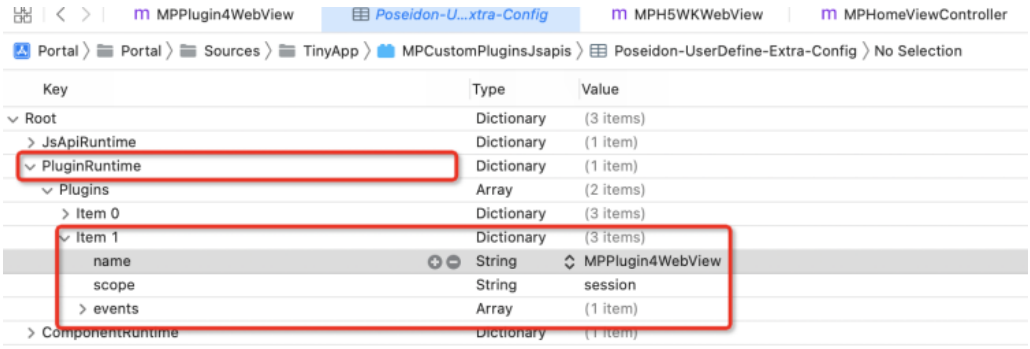
```
@interface MPPlugin4WebView : NBPluginBase

@end

@implementation MPPlugin4WebView

- (void)pluginDidLoad
{
    self.scope = kPSDScope_Scene;

    // -- 拦截调用的jsapi信息
    [self.target addEventListener:kEvent_Invocation_Event_Start withListener:self useCapture:NO];
    [self.target addEventListener:kEvent_Invocation_Invoke withListener:self useCapture:NO];

    [super pluginDidLoad];
}

- (void)handleEvent:(PSDEvent *)event
{
    [super handleEvent:event];

    if([kEvent_Invocation_Event_Start isEqualToString:event.eventType] ||
             [kEvent_Invocation_Invoke isEqualToString:event.eventType]){
        PSDInvocationEvent *invocationEvent = (PSDInvocationEvent *)event;
        NSString *apiName = invocationEvent.invocationName;
        NSDictionary *data = invocationEvent.invocationData;

        // 获取到当前页面的url，根据scheme和host进行字符串匹配校验
        NSURL *url = event.context.currentViewController.url;
        if (![url.host isEqualToString:@"xxx"] || ![url.scheme isEqualToString:@"xxx"]) {
            [event preventDefault];
            [event stopPropagation];
            return;
        }

    }
}

- (int)priority
{
    return PSDPluginPriority_High+1;
}
```

> **重要**
>
> URL 要进行精准匹配，至少要匹配到 URI 类的 scheme 和 host 信息，慎用或不用正则匹配，严格避免使用 contains、startsWith、endsWith、indexOf 等不精准函数。

2. 注册 Plugin。

   i. 在 mPaaS 容器初始化时，指定自定义 Plugin 的路径。

ii. 在自定义 Plugin 的 bundle 的 plist 文件中，注册上一步的 Plugin。详情请参考 注册 Plugin。



1.4.5.2. 打开 URL 判断逻辑

为了更好的保障打开 URL 时的 App 的安全性，可在容器调用相关 URL 之前对 URL 进行判断，如果打开的不是白名单内的 URL 则禁止调用。

建议在调用如下接口前进行 URL 判断：

```
// 判断打开的 URL 是否在白名单内
NSString *urlWhiteList = @"xxxx";
NSURL *url = [NSURL URLWithString:@"https://example.com/products/xxx"];
if (![url.host isEqualToString:urlWhiteList]) {
    return;
}

// 打开在线 URL
[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url":
@"https://example.com/products/xxx"}];

// 基于 URL 创建 vc
MPH5WebViewController *vc = (MPH5WebViewController *)[[MPNebulaAdapterInterface shareInstance] createH5
ViewController:@{@"url":@"https://example.com/products/xxx"}];
```

> **重要**
>
> URL 要进行精准匹配，至少要匹配到 URI 类的 scheme 和 host 信息，慎用或不用正则匹配，严格避免使用 contains、startsWith、endsWith、indexOf 等不精准函数。

1.4.5.3. 定制 H5 页面导航栏

在定制 H5 页面的导航栏样式前，您需要确保已经了解以下框架导航栏和 H5 容器相关知识。

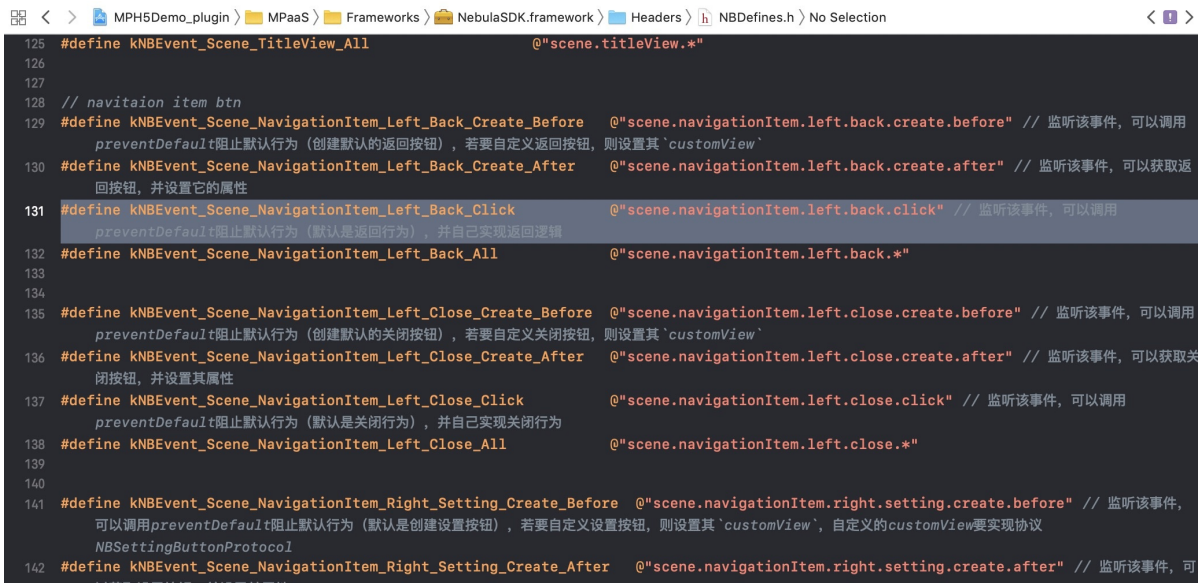
前置条件

- iOS 定制导航栏
- 自定义 JSAPI
- 自定义插件

设置所有 H5 页面导航栏默认样式

在应用主题的基础上，若您需要指定所有 H5 页面统一的样式，可以通过监听 Nebula 容器提供的事件机制进行定制。

- 容器支持的事件，参见头文件 `NebulaSDK/NBDefine.h` ：



- 自定义插件 监听事件需要进行定制：

```
@implementation MPPlugin4TitleView

- (void)pluginDidLoad
{
     self.scope = kPSDScope_Scene;
     // -- 返回区域
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Left_Back_Create_Before
withListener:self useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Left_Back_Create_After
withListener:self useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Left_Close_Create_Before
withListener:self useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Left_Close_Create_After
withListener:self useCapture:NO];

     // -- 标题区域
     [self.target addEventListener:kNBEvent_Scene_TitleView_Create_Before withListener:self
useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_TitleView_Create_After withListener:self
useCapture:NO];

     // -- 控制按钮区域
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Right_Setting_Create_Before
withListener:self useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Right_Setting_Create_After
withListener:self useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Right_SubSetting_Create_After withLi
stener:self useCapture:NO];
     [self.target addEventListener:kNBEvent_Scene_NavigationItem_Right_Setting_Change
withListener:self useCapture:NO];

     // -- 进度条
     [self.target addEventListener:kNBEvent_Scene_ProgressView_Create_Before withListener:self useCap
ture:NO];
     [self.target addEventListener:kNBEvent_Scene_ProgressView_Create_After withListener:self useCapt
ure:NO];

     [super pluginDidLoad];
}
```

- 设置返回区域中返回按钮和关闭按钮的样式：

```
- (void)handleEvent:(PSDEvent *)event
{
    [super handleEvent:event];

    if ([kNBEvent_Scene_NavigationItem_Left_Back_Create_Before isEqualToString:event.eventType]) {
        [event preventDefault];

        event.context.currentViewController.navigationItem.leftBarButtonItem = [[UIBarButtonItem al
loc] initWithTitle:@"取消" style:UIBarButtonItemStylePlain target:self
action:@selector(onClickBack)];
    }else if ([kNBEvent_Scene_NavigationItem_Left_Back_Create_After
isEqualToString:event.eventType]){
        // 修改返回按钮样式
        NSArray *leftBarButtonItems =
event.context.currentViewController.navigationItem.leftBarButtonItems;
        if ([leftBarButtonItems count] == 1) {
            if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButtonItem class
]]) {
                // 在默认返回按钮基础上，修改返回箭头和文案颜色
                AUBarButtonItem *backItem = leftBarButtonItems[0];
                backItem.backButtonColor = [UIColor greenColor];
                backItem.titleColor = [UIColor colorFromHexString:@"#00ff00"];

                // 隐藏返回箭头
//                backItem.hideBackButtonImage = YES;

                // 隐藏返回文案：文案设置为透明，保留返回按钮 s 点击区域
//                backItem.titleColor = [UIColor clearColor];
            }
        }
    }else if ([kNBEvent_Scene_NavigationItem_Left_Close_Create_Before
isEqualToString:event.eventType]){
//        // 隐藏关闭按钮
//        [event preventDefault];
//        NBNavigationItemLeftCloseEvent *itemEvent = (NBNavigationItemLeftCloseEvent *)event;
//        UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
//        button.frame = CGRectMake(0, 0, 44, 44);
//        button.backgroundColor = [UIColor greenColor];
//        [button setTitle:@"Close" forState:UIControlStateNormal];
//        itemEvent.customView = button;
    }else if ([kNBEvent_Scene_NavigationItem_Left_Close_Create_After
isEqualToString:event.eventType]){
//        // 修改关闭按钮样式
//        [event preventDefault];
//        NBNavigationItemLeftCloseEvent *itemEvent = (NBNavigationItemLeftCloseEvent *)event;
//        UIButton *closeButton = (UIButton *)itemEvent.customView;
//        [closeButton setTitle:@"Close" forState:UIControlStateNormal];
//        [closeButton setTitleColor:[UIColor greenColor] forState:UIControlStateNormal];
    }
}
```

- 设置 H5 页面标题样式：

```
  if ([kNBEvent_Scene_TitleView_Create_Before isEqualToString:event.eventType]) {
          // 重写 TitleView 的样式
          NBNavigationTitleViewEvent *e = (id)event;
          [e preventDefault];

      }else if ([kNBEvent_Scene_TitleView_Create_After isEqualToString:event.eventType]) {
          // 更改已创建 TitleView 的样式
          NBNavigationTitleViewEvent *e = (id)event;
          [[e.titleView mainTitleLabel] setFont:[UIFont systemFontOfSize:16]];
          [[e.titleView mainTitleLabel] setTextColor:[UIColor greenColor]];
          [e.titleView mainTitleLabel].lineBreakMode = NSLineBreakByTruncatingMiddle;
      }
```

- 设置 OptionMebu 控制按钮的样式：

```
  if ([kNBEvent_Scene_NavigationItem_Right_Setting_Create_After isEqualToString:event.eventType] || [
kNBEvent_Scene_NavigationItem_Right_SubSetting_Create_After isEqualToString:event.eventType]) {
          // 更改已创建 RightBarItem 的样式
          NBNavigationItemRightSettingEvent *settingEvent = (id)event;
          settingEvent.adjustsWidthToFitText = YES;
          settingEvent.maxWidth = [UIScreen mainScreen].bounds.size.width / 3.0f;
          UIButton *button = settingEvent.customView;
          button.titleLabel.font = [UIFont systemFontOfSize:14.0f];
          CGRect frame = CGRectMake(0, 0, 22, 22);
          button.frame = frame;
          [button setTitleColor:[UIColor blackColor] forState:UIControlStateNormal];
          if (!CGSizeEqualToSize(button.bounds.size, frame.size)) {
              button.frame = frame;
          }
      }
```

  - 设置 H5 页面加载时进度条的样式：

```
if([kNBEvent_Scene_ProgressView_Create_After isEqualToString:event.eventType]){
      NBProgressViewEvent *progressEvent = (NBProgressViewEvent *)event;
      id<NBProgressViewProtocol> progressView = progressEvent.progressView;
      [progressView setProgressTintColor:[UIColor greenColor]];
  }
```

定制某个 H5 页面导航栏样式

若您需要定制某个 H5 页面导航栏的样式，根据修改时机不同，提供的方法也分为两类：**页面加载前** 与 **页面打开后**。

- 页面加载前，在默认导航栏样式基础上定制导航栏，主要分为以下两步：

  i. 自定义启动参数，当前 H5 页面加载时自定义启动参数，指定定制方式：

  - 从 H5 页面进入另一个 H5 页面，传递自定义启动参数的方法参见 pushWindow 打开新页面 和 startApp 启动其他应用。

- 从 Native 页面进入一个 H5 页面，传递自定义启动参数的方法参考如下代码：

```
#pragma mark 进入页面时修改，H5 需通过启动参数设置
- (void)gotoHideNavigator
{
    // 打开 H5 页面，隐藏导航栏
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"http
s://tech.antfin.com", @"showTitleBar":@NO,@"transparentTitle":@"auto"}];
}

- (void)gotoShowNavigator
{
    // 打开 H5 页面，显示导航栏
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"http
s://tech.antfin.com", @"showTitleBar":@YES}];
}

- (void)gotoTransparency
{
    // 打开 H5 页面，设置透明导航栏
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"http
s://tech.antfin.com", @"transparentTitle":@"auto"}];
}

- (void)gotoUpdateBackgroundColor
{
    // 修改导航栏背景颜色
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"http
s://tech.antfin.com", @"titleBarColor":@"16775138"}];
}

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

- (void)gotoUpdateBackTitleColor
{
    // 修改默认返回按钮文案颜色
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"http
s://tech.antfin.com", @"backButtonColor":@"ff0000"}];
}

- (void)gotoUpdateTitleColor
{
    // 修改标题颜色
    [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"http
s://tech.antfin.com", @"titleColor":@"ff0000"}];
}
```

ii. 处理 H5 基类，在基类的 `viewWillAppear` 方法中，根据传入的启动参数调用 native 接口方法对导航栏样式进行修改如下：

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    // 当前页面的 WebView
    UIWebView *currentWebView = (UIWebView *)self.psdContentView;
    NSLog(@"[mpaas] webView: %@", currentWebView);

    // 当前页面的启动参数
    NSDictionary *expandParams = self.psdScene.createParam.expandParams;
```

```
    NSLog(@"[mpaas] expandParams: %@", expandParams);

    if ([expandParams count] > 0) {
        [self customNavigationBarWithParams:expandParams];
    }
    WKWebView *webView = (WKWebView *)[self psdContentView];
    if (@available(iOS 11.0, *)) {
        webView.scrollView.contentInsetAdjustmentBehavior =
UIScrollViewContentInsetAdjustmentNever;
    } else {
        self.automaticallyAdjustsScrollViewInsets = NO;
    }
    webView.scrollView.contentInset = UIEdgeInsetsMake(200, 0, 0, 0);
    webView.scrollView.scrollIndicatorInsets = webView.scrollView.contentInset;
}

- (void)customNavigationBarWithParams:(NSDictionary *)expandParams
{
    // 定制导航栏背景
    NSString *titleBarColorString = expandParams[@"titleBarColor"];
    if ([titleBarColorString isKindOfClass:[NSString class]] && [titleBarColorString length] > 0)
{
        UIColor *titleBarColor = [UIColor colorFromHexString:titleBarColorString];
        [self.navigationController.navigationBar setNavigationBarStyleWithColor:titleBarColor tran
slucent:NO];
        [self.navigationController.navigationBar setNavigationBarBottomLineColor:titleBarColor];
    }

    //导航栏是否隐藏，默认不隐藏。设置隐藏后，webview 需全屏
    NSString *showTitleBar = expandParams[@"showTitleBar"];
    if (showTitleBar && ![showTitleBar boolValue]) {
        self.options.showTitleBar = NO;
        [self.navigationController setNavigationBarHidden:YES];

        // 调整 webview 的位置
        UIWebView *webView = (UIWebView *)[self psdContentView];
        CGRect frame = webView.frame;
        frame.origin.y = [[UIApplication sharedApplication] statusBarFrame].size.height;
        frame.size.height -= [[UIApplication sharedApplication] statusBarFrame].size.height;
        webView.frame = frame;
        self.automaticallyAdjustsScrollViewInsets = NO;

    }

    //导航栏是否透明，默认不透明。设置透明后，webview 需全屏
    NSString *transparentTitle = expandParams[@"transparentTitle"];
    if ([transparentTitle isEqualToString:@"always"] || [transparentTitle
isEqualToString:@"auto"]) {

        // 导航栏和底部横线变为透明
        UIColor *clearColor = [UIColor clearColor] ;
        [self.navigationController.navigationBar setNavigationBarTranslucentStyle];
        [self.navigationController.navigationBar setNavigationBarStyleWithColor:clearColor
translucent:YES];

        // 调整 webview 的位置
        self.edgesForExtendedLayout = UIRectEdgeAll;
        if (@available(iOS 11.0, *)) {
            UIWebView *wb = (UIWebView *)[self psdContentView];
            wb.scrollView.contentInsetAdjustmentBehavior =
UIScrollViewContentInsetAdjustmentNever;
        }else{
            self.automaticallyAdjustsScrollViewInsets = NO;
        }
```

```
    }

    // 修改默认返回按钮文案颜色
    NSString *backButtonColorString = expandParams[@"backButtonColor"];
    if ([backButtonColorString isKindOfClass:[NSString class]] && [backButtonColorString length] >
0) {
        UIColor *backButtonColor = [UIColor colorFromHexString:backButtonColorString];

        NSArray *leftBarButtonItems = self.navigationItem.leftBarButtonItems;
        if ([leftBarButtonItems count] == 1) {
            if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButtonItem
class]]) {
                AUBarButtonItem *backItem = leftBarButtonItems[0];
                backItem.titleColor = backButtonColor;
                backItem.backButtonColor = backButtonColor;
            }
        }
    }

    // 设置标题颜色
    NSString *titleColorString = expandParams[@"titleColor"];
    if ([titleColorString isKindOfClass:[NSString class]] && [titleColorString length] > 0) {
        UIColor *titleColor = [UIColor colorFromHexString:titleColorString];
        id<NBNavigationTitleViewProtocol> titleView = self.navigationItem.titleView;
        [[titleView mainTitleLabel] setFont:[UIFont systemFontOfSize:16]];
        [[titleView mainTitleLabel] setTextColor:titleColor];
    }
}
```

- 页面打开后，在用户操作的过程中动态修改导航栏样式。主要通过自定义 JSAPI 的方式，调用 Native 接口方法进行修改。
  - 自定义 JSAPI 对当前页面导航栏样式进行处理。
  - 参考 定制某一个页面导航栏样式 提供的接口，在 JSAPI 中对原生导航栏进行处理：

```
- (void)handler:(NSDictionary *)data context:(PSDContext *)context callback:
(PSDJsApiResponseCallbackBlock)callback
{
    [super handler:data context:context callback:callback];

    UIViewController *currentVC = context.currentViewController;
    currentVC.navigationItem.titleView = [[AUDoubleTitleView alloc] initWithTitle:@"主标题" detailTi
tle:@"副标题"];
    callback(@{@"success":@YES});
}
```

1.4.5.4. H5 容器自动化埋点

使用 mPaaS 提供的 Nebula 容器加载 H5 页面时，Nebula 容器会自动统计页面的行为数据、加载性能，捕获异常报错等数据，方便您更好地跟踪 H5 页面加载时的各项数据。本文将引导您集成 Nebula 容器自动化埋点能力及查看埋点数据。

前置条件

集成 mPaaS 提供的 H5 容器自动化埋点功能，需满足以下两个条件：

- 您已在 mPaaS 控制台 中创建了一个 App。
- 您已完成 添加 SDK 步骤，客户端工程集成了 `NebulaLogging.framework` 。

操作步骤

初始化配置

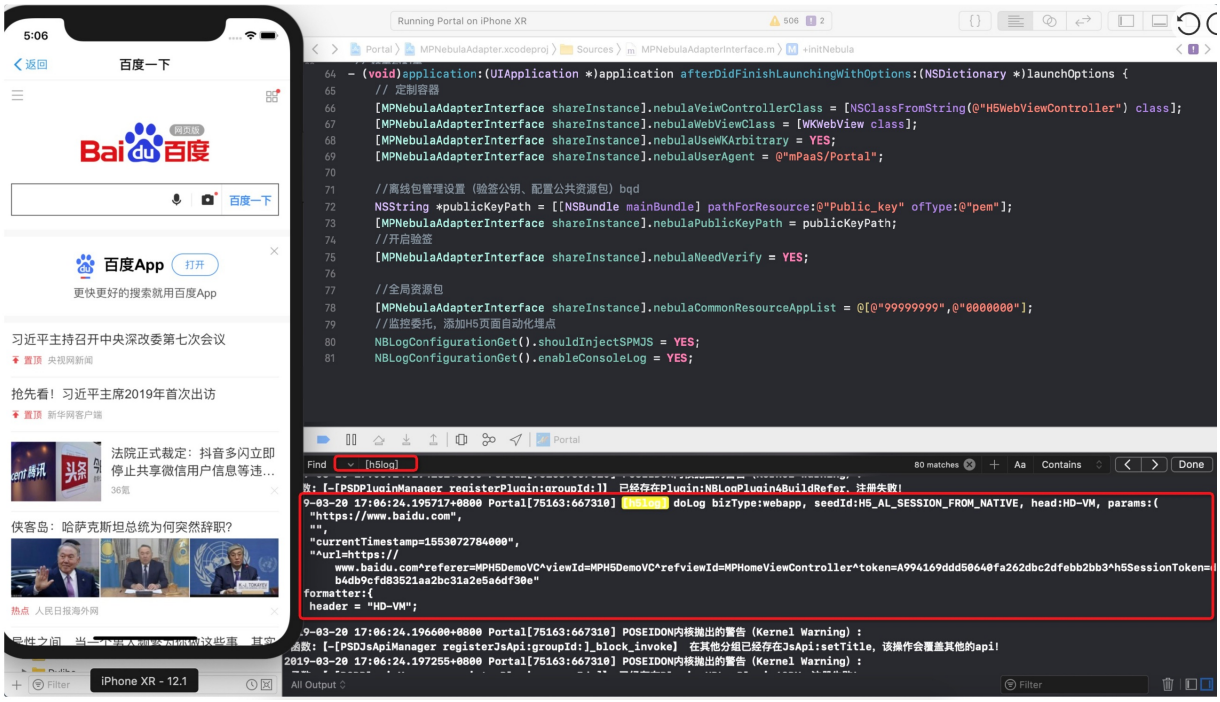
1. 要集成 Nebula 容器的自动埋点功能，需在容器初始化时启动 H5 埋点监控：

```
// 监控委托，添加 H5 页面的自动化埋点
 NBLogConfigurationGet().shouldInjectSPMJS=YES;
 #ifdef DEBUG
 NBLogConfigurationGet().enableConsoleLog = YES;
 #endif
 [NBLogServiceGet() start];
 [[NBMonitor defaultMonitor] setDelegate:NBLogServiceGet()];
```

2. 启动 H5 埋点监控后，使用 Nebula 容器加载 H5 页面，容器就会自动统计页面加载行为、性能及异常数据等。根据查看埋点数据的场景不同，分为以下两种方式：
   - 查看客户端日志：查看客户端本地的埋点数据，适用于应用开发阶段的问题排查。参见下文 查看客户端日志。
   - 查看服务端日志：查看线上用户产生的真实埋点数据，适用于应用发布上线后，排查线上问题。参见下文 查看服务端日志。

查看客户端日志

加载 H5 页面后，在 XCode 控制台搜索关键字 [h5log]，即可查看页面加载相关的埋点数据的关键信息，如下图。

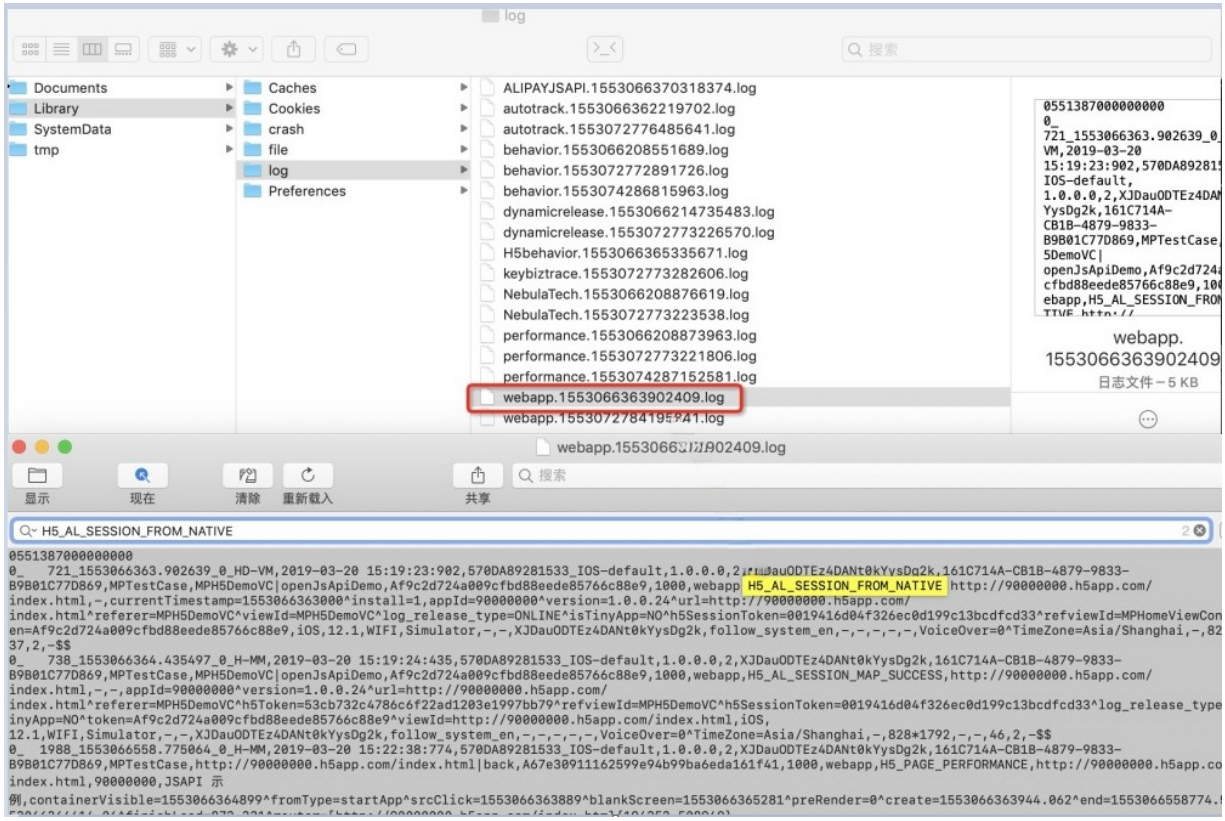


Xcode 控制台打出的 H5 容器埋点数据，只包含了以下关键信息：

SeedId	示例	含义
bizType	webapp	日志类型，写入本地日志文件的文件名。
seedId	H5_AL_SESSION_FROM_NATIVE	埋点的唯一标识，具体含义参见下文的 H5 容器埋点集。
head	HD-VM	日志模型。可参考 性能埋点。
	https://www.baidu.com	当前页面的 URL。
	currentTimestamp=1553072784000	自定义参数，如时间戳，页面加载状态。
	-	自定义参数，如页面元素位置。

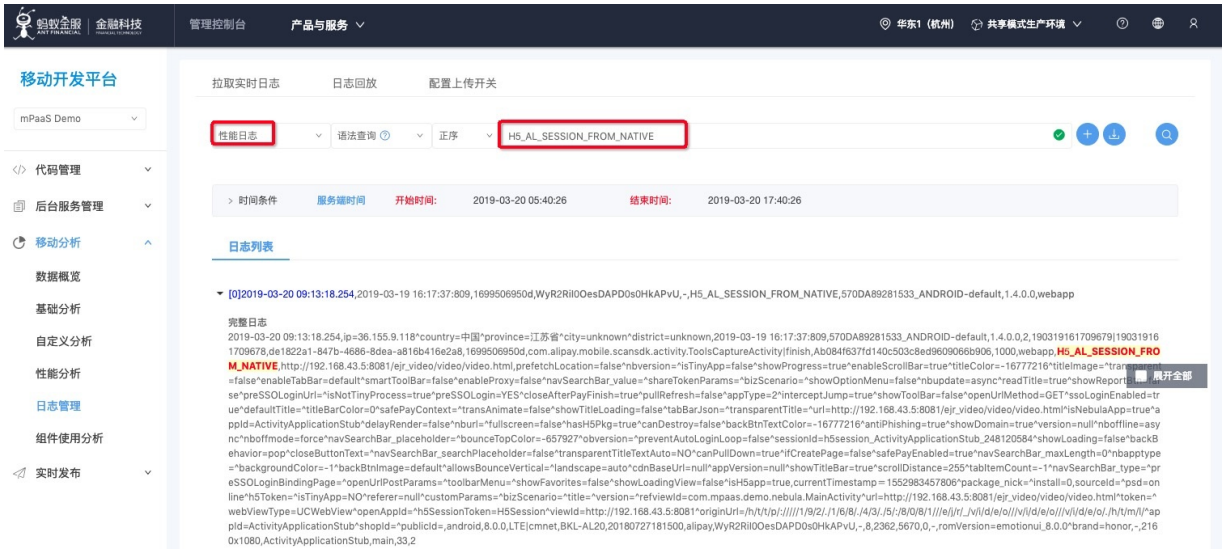
params	^url=https://www.baidu.com^referer=MPH5DemoVC^viewId=MPH5DemoVC^refviewId=MPHomeViewController^token=A994169ddd50640fa262dbc2dfebb2bb3^h5SessionToken=db4db9cfd83521aa2bc31a2e5a6df30e/td>	埋点信息：包括离线包 APPID、离线包版本号、当前 H5 页面加载的 URL、所在的 VC 类名、当前页面的 token 等。
formatter	header = "HD-VM"	同 head 字段说明。

若您需要查看 Xcode 控制台上某条日志的完整内容，可在沙盒中找到 `Library/log` 下以 `bizType` 开头的文件名，然后根据 seedId 关键字搜索查询。



查看服务端日志

若需要查看线上用户 H5 容器的埋点数据，可在 mPaaS 控制台通过 查询历史日志 进行查询。



H5 容器埋点集

H5 容器统计的自动化埋点信息，根据埋点数据的 seedID 作为唯一标识，按加载的 H5 页面方式不同，主要有以下三类：

- **打开在线 URL 的相关埋点集 ：**

SeedId	含义
H5_AL_SESSION_FROM_NATIVE	容器已经启动。
H5_AL_PAGE_START	页面开始加载。
H5_AL_NETWORK_START	页面开始发起网络请求。
H5_OPEN_PAGE_FINISH	页面加载完成。
H5_AL_PAGE_APPEAR	页面首次出现。
H5_AL_JSAPI_SENDEVENT	页面调用 JSAPI。
H5_AL_JSAPI_NOTFOUND	页面调用 JSAPI 失败。
H5_TITLEBAR_BACK_BT	单击导航栏返回按钮。
H5_PAGE_PERFORMANCE	统计页面加载的性能。

- **打开离线包页面的相关埋点集 ：**

SeedId	含义
H5_APP_REQUEST	请求离线包信息。
H5_APP_LOAD_DATASOURCE	加载离线包信息。

H5_AL_SESSION_FROM_NATIVE	容器已经启动。
H5_APP_DOWNLOAD	离线包下载。
H5_APP_UNZIP	离线包解压。
H5_APP_POOL	包管理信息池操作 （包含池内信息的增、删、改）。
H5_APP_VERIFY	离线包验签。
H5_AL_SESSION_VERIFYTAR_FAIL	离线包验签失败。
H5_AL_PAGE_START	页面开始加载。
H5_AL_SESSION_MAP_SUCCESS	加载本地离线包成功。
H5_AL_SESSION_FALLBACK	加载本地离线包失败，走 fallback 请求在线页面。
H5_OPEN_PAGE_FINISH	页面加载完成。
H5_AL_PAGE_APPEAR	页面首次出现。

- **异常埋点：**

seedId	含义
H5_AL_NETWORK_PERFORMANCE_ERROR	资源请求异常。
H5_PAGE_ABNORMAL	页面异常。
H5_AL_PAGE_JSERROR	JS 异常。
H5_AL_JSAPI_RESULT_ERROR	JSAPI 异常。

1.4.5.5. 自定义 JSAPI

要从页面发起 Native 功能调用，例如显示一个 ActionSheet，或显示联系人对话框，您需要扩展一个 JavaScript API（JSAPI）。使用 JSAPI，可以让您在 H5 页面增加 Native 功能调用入口。通过实现自定义 JSAPI 类中的 handler 方法，以 Native 的形式实现特定功能。

H5 容器组件提供以下能力：

- 丰富的内置 JSAPI，实现例如页面 push、pop、标题设置等功能。更多信息，请参见 内置 JSAPI。
- 支持用户自定义 JSAPI 和插件功能，扩展业务需求。

本文将结合 H5容器和离线包 Demo，自定义一个在 H5 页面加载时，修改页面导航栏的插件。

关于此任务

自定义一个 JSAPI 可以有以下两种方式：

- **Plist 注册**

- **代码注册**

操作步骤

Plist 注册

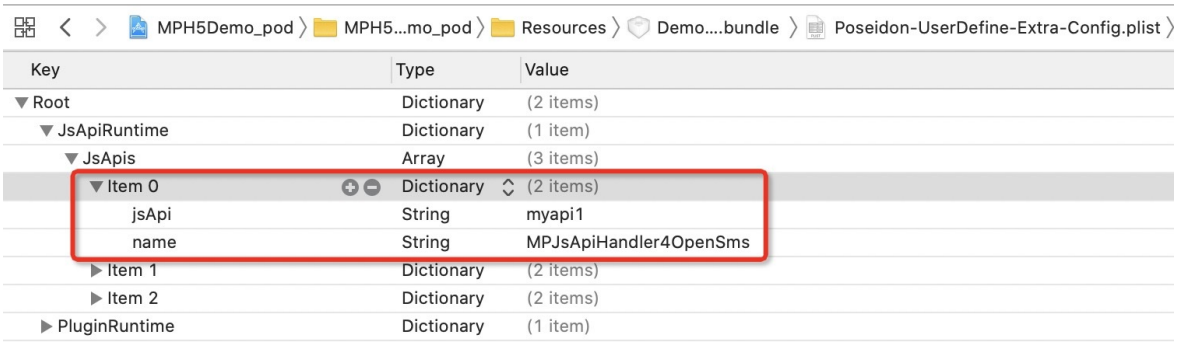
1. 创建 JSAPI 类：
   - 命名规范：为与容器默认提供的插件命名保持一致，创建的 JSAPI 类命名以 `XXJsApiHandler4` 开头，其中 `XX` 为自定义的前缀。
   - 基类：所有 JSAPI 均继承自 `PSDJsApiHandler`。
   - 实现基础方法：在 `.m` 文件中，需重写方法 `-(void)handler:context:callback:`。当在前端调用此 JSAPI 时，会转发到此方法。
   - 该方法的参数含义如下：

参数	描述
data	H5 页面调用此 JSAPI 时传入的参数。
context	当前 H5 页面的上下文，具体可参考 `PSDContext.h`。
callback	调用此 JSAPI 完成后的回调方法，以字典方式传递调用结果到 H5 页面。

   示例代码如下：

```
  #import <NebulaPoseidon/NebulaPoseidon.h>
  @interface MPJsApiHandler4OpenSms : PSDJsApiHandler
  @end
  @implementation MPJsApiHandler4OpenSms
  - (void)handler:(NSDictionary *)data context:(PSDContext *)context callback:
(PSDJsApiResponseCallbackBlock)callback
  {
  [super handler:data context:context callback:callback];
  // 打开系统短信
  NSURL *url = [NSURL URLWithString:@"sms://xxx"];
  BOOL reasult = [[UIApplication sharedApplication] openURL:url];
  callback(@{@"success":@(reasult)});
  }
  @end
```

2. 注册 JSAPI。在自定义的 Plist 文件中注册此 JSAPI。
   - 为统一管理自定义的 JSAPI 和 Plugin，新建一个 Plsit 文件，您可以直接下载此模板文件 DemoCustomPlugins.bundle.zip 并添加到工程中。
   - 在 `JsApis` 数组下注册上一步创建的 JSAPI 类：



   - 注册的 JSAPI 是一个字典类型，包含以下两项内容：

名称	描述
jsApi	在 H5 页面中调用的 JSAPI 接口名。 **重要** 为防止自定义的 JSAPI 与容器内置 JSAPI 相互影响导致不可用，请给自定义 JSAPI 名加上前缀予以区分。
name	创建的 JSAPI 的类名。

- 同时需在初始化容器配置时，指定自定义的 Plist 文件的路径。
  - 初始化 H5 容器，请参见 H5 容器快速开始。

  示例代码如下：

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
  {
      // 初始化容器
      // [MPNebulaAdapterInterface initNebula];

      // 自定义 JSAPI 路径和预置离线包信息
      NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"DemoCustomPresetApps.bundle/h5_json.json"] ofType:nil];
      NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"DemoCustomPresetApps.bundle"] ofType:nil];
      NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"DemoCustomPlugins.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
      [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath
customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:pluginsJsapisPath]
  }
```

代码注册

除使用 Plist 方式自定义一个 JSAPI 外，容器也支持直接调用 Nebula 容器提供的接口方法注册一个自定义的 JSAPI。

1. 参考 自定义插件 文档，新建一个 Plugin。
2. 在 Plugin 中实现 `addJSApis` 方法。

```
- (void)addJSApis
{
    [super addJSApis];

    // 代码注册 jsapi
    PSDJsApi *jsApi4DemoTest2 = [PSDJsApi jsApi:@"demoTest2"
                                        handler:^(NSDictionary *data, PSDContext *context, PSDJsApiR
esponseCallbackBlock responseCallbackBlock) {
                                            responseCallbackBlock(@{@"result":@"jsapi-demoTest2调用 N
ative 的处理结果"});
                                        }
                                    checkParams:NO
                                      isPrivate:NO
                                          scope:self.scope];
    [self registerJsApi2Target:jsApi4DemoTest2];
}
```

- 下表列出注册时各参数及其含义：

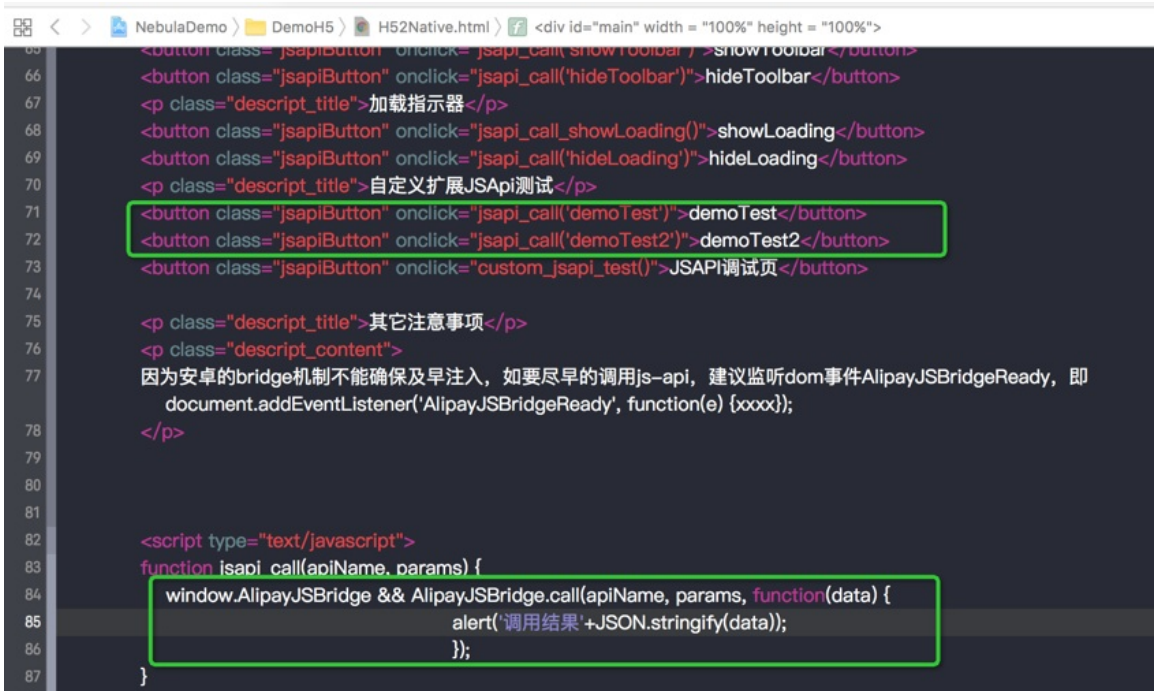
参数	描述

jsApi	在 H5 页面中调用的 JSAPI 接口名。
handler	JSAPI 处理函数，功能同 Plist 注册方式中的 handler 方法。
checkParams	是否检查参数，请设置为 NO。
isPrivate	是否私有 JSAPI，请设置为 NO。
scope	作用域，请设置为 self.scope。

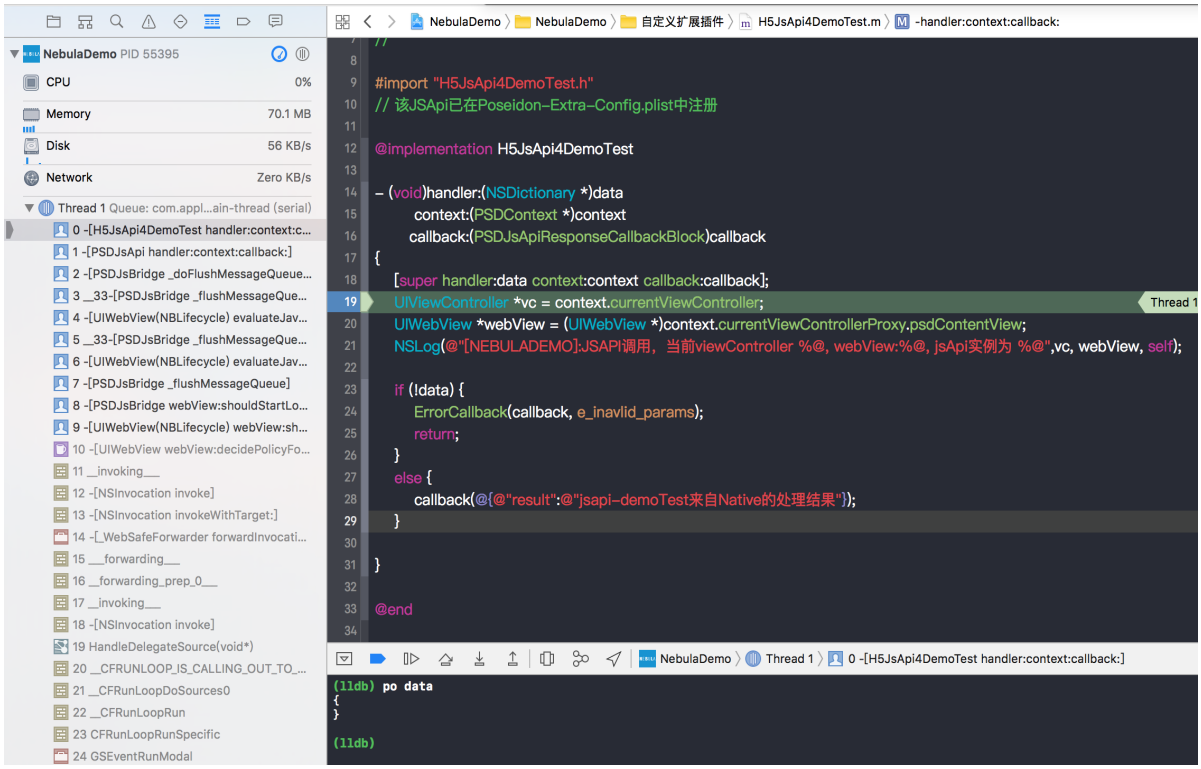
具体示例，请参考代码示例中 `MPPlugin4TitleView` 类相关实现。

后续操作

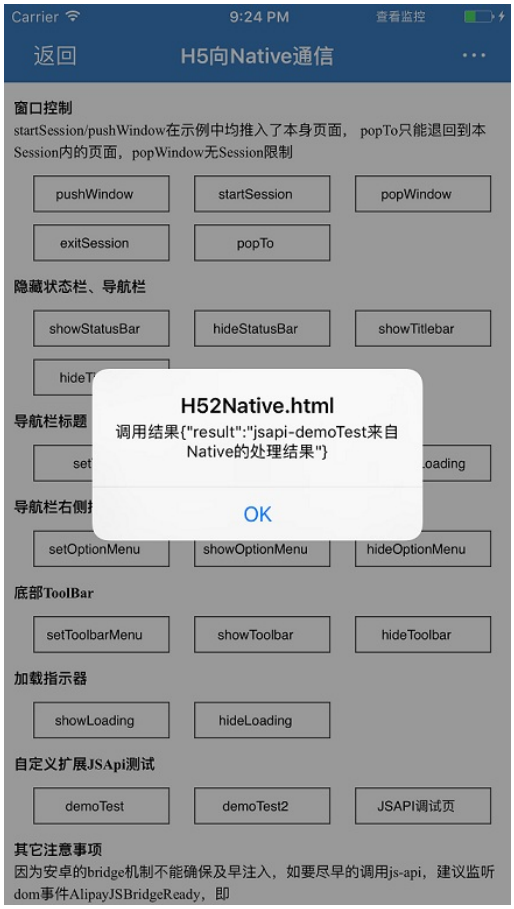
1. 在 H5 页面中调用自定义的 JSAPI。



2. 在 handler 方法中添加断点，观察 H5 页面提供的参数是否符合预期。



3. 查看 H5 页面返回的结果是否符合预期。



1.4.5.6. 自定义插件

如果您需要在某个时机完成某些事件，例如进入页面时记录埋点，那么您需要开发一个插件 (Plugin)。插件订阅相应的事件后，就可以在 handler 中实现对事件所携带的数据进行加工处理。

关于此任务

自定义插件的过程分为以下步骤：

1. 新建 Plugin
2. 注册 Plugin
3. 使用 Plugin

本文将结合 H5 容器和离线包 Demo，自定义一个在 H5 页面加载时，修改页面导航栏的插件。

操作步骤

新建 Plugin

新创建的 Plugin 类一般格式的代码示例如下：

```
#import <NebulaSDK/NBPluginBase.h>

@interface MPPlugin4TitleView : NBPluginBase

@end

@implementation MPPlugin4TitleView

- (void)pluginDidLoad
{
}

- (void)handleEvent:(PSDEvent *)event
{
   [super handleEvent:event];
}

- (int)priority
{
    return PSDPluginPriority_High +1;
}
```

需注意以下几点：

- 命名：为与容器默认提供的 Plugin 命名保持一致，统一以 `XXPlugin4` 开头，其中 `XX` 为自定义的前缀。
- 基类：所有插件均继承自 `NBPluginBase` 。
- 实现基础方法：在 `.m` 文件中，需重写以下方法：
  - `- (void)pluginDidLoad` ：必选。监听的 H5 事件，事件列表请查看头文件 `NBDefines.h` 。
  - `- (void)handleEvent` ：必选。处理监听的事件触发后的逻辑。
  - `- (``**int**``)``priority` ：必选。事件的优先级，设置为 `PSDPluginPriority_High`` +``1` 。
  - `- (void)addJSApis` ：可选。因为要与 H5 通信，可能需要注册 JSAPI。

监听事件

在 `- (void)pluginDidLoad` 方法中注册需要监听的事件。

```
- (void)pluginDidLoad {
    self.scope = kPSDScope_Scene; // 1

    [self.target addEventListener:kNBEvent_Scene_NavigationItem_Left_Back_Create_After
withListener:self useCapture:NO];
    // -- 修改导航栏风格
    [self.target addEventListener:kH5Event_Scene_NavigationBar_ChangeColor withListener:self
useCapture:NO];

    [super pluginDidLoad];
}
```

`addEventListener` 方法用于监听某个事件，各参数说明如下：

名称	含义
scope	设置事件影响范围。目前支持的范围从小到大依次是 Scene、Session 和 Service。
event	设置事件名称，事件常量定义在 `NBDefines.h` 中。
listener	设置事件的处理者，即提供 `- handleEvent:` 的对象。
capture	设置是否使用捕捉的方式传播事件，一般使用 `NO` 。

设置插件优先级

避免自定义插件被覆盖，需要设置插件的高优先级。

```
- (int)priority
{
    return PSDPluginPriority_High +1;
}
```

处理监听

最后，在 `- handleEvent:` 中处理监听的事件触发后的逻辑。

```
- (void)handleEvent:(NBNavigationTitleViewEvent *)event
{
    [super handleEvent:event];

    if ([kNBEvent_Scene_NavigationItem_Left_Back_Create_After isEqualToString:event.eventType]){
        // 在默认返回按钮基础上，修改样式
        NSArray *leftBarButtonItems =
event.context.currentViewController.navigationItem.leftBarButtonItems;
        if ([leftBarButtonItems count] == 1) {
            if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButtonItem class]])
{
                // 在默认返回按钮基础上，修改返回箭头和文案颜色
                AUBarButtonItem *backItem = leftBarButtonItems[0];
                backItem.backButtonColor = [UIColor greenColor];
                backItem.titleColor = [UIColor colorFromHexString:@"#00ff00"];

                // 隐藏返回箭头
                //                backItem.hideBackButtonImage = YES;

                // 隐藏返回文案：文案设置为透明，保留返回按钮点击区域
                //                backItem.titleColor = [UIColor clearColor];
            }
        }
        [event preventDefault];
        [event stopPropagation];
    }else if([kH5Event_Scene_NavigationBar_ChangeColor isEqualToString:event.eventType]) {

        // 禁止容器默认导航栏样式
        [event preventDefault];
        [event stopPropagation];
    }
}
```

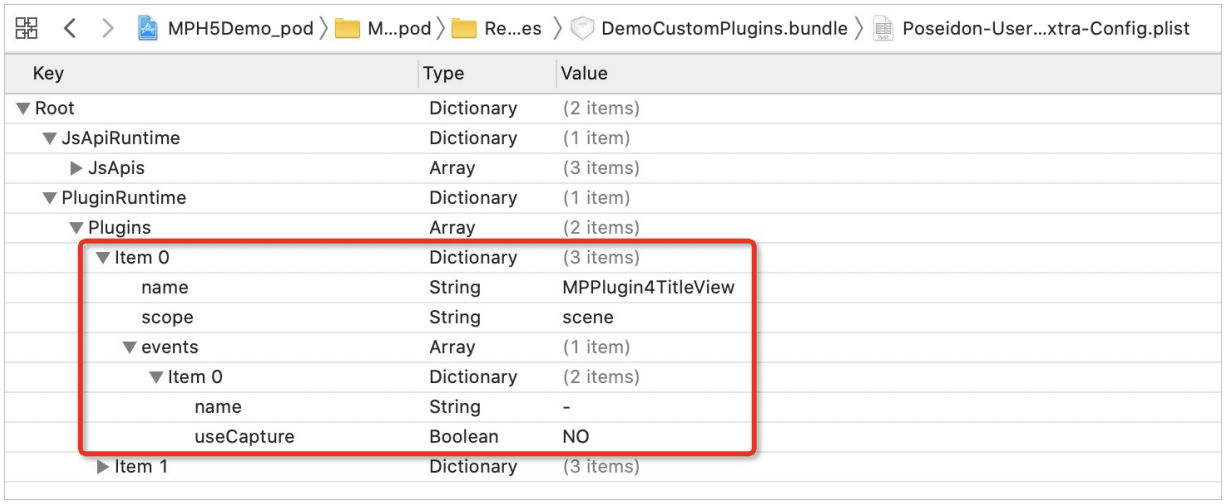
添加 JSAPI

若在注册 Plugin 的过程中，需要自定义 JSAPI 与 H5 页面进行交互，可在 `- (void)addJSApis` 方法中，使用代码注册（参考 自定义 JSAPI > 代码注册）的方式进行处理。此方法为可选项，若无需要可不实现。

```
- (void)addJSApis
{
    [super addJSApis];
    // 可以在这里添加 TitleView 相关的自定义 JSAPI
}
```

注册 Plugin

创建了 Plugin 类后，需要在自定义的 Plist 文件（参见 自定义 JSAPI > 注册 JSAPI 说明）中注册此 Plugin。

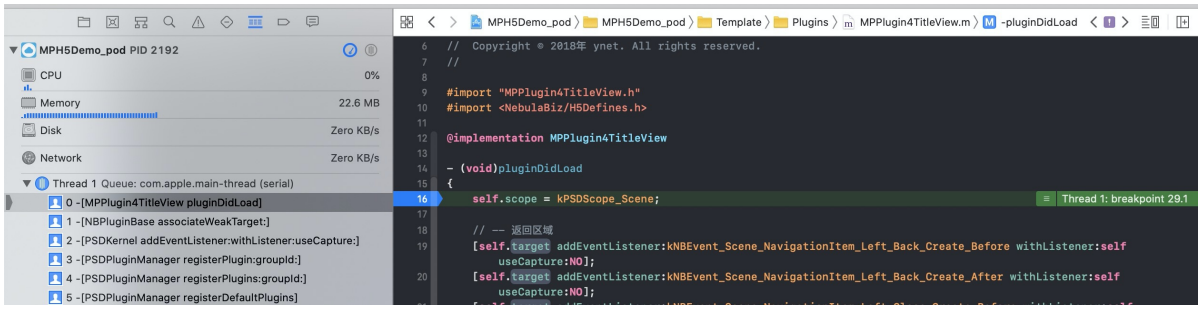


注册的 Plugin 是一个字典类型，包含以下三项内容：

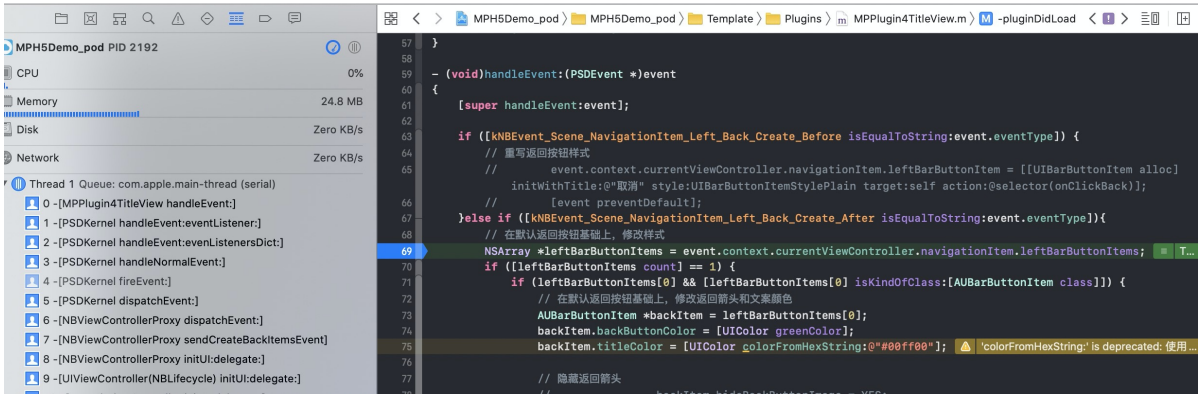
名称	含义
name	创建的 Plugin 类名
scope	Plugin 生效的范围
events	监听的 event 名称

使用 Plugin

1. 在 `pluginDidLoad` 方法中添加断点，观察触发时机的堆栈调用顺序是否正确。



2. 在 `handleEvent` 方法中添加断点，观察监听的事件能否正确触发。



3. 观察 H5 页面自定义的导航栏样式是否生效。

1.4.5.7. H5 自定义报错页面

在加载 H5 页面时，如果网络加载失败或无法打开网站，会出现类似如下的报错：

“网络无法连接（-1009）”。

处理在 H5 页面中如上报错的方法与小程序相同，参见 iOS 小程序自定义报错页面。

1.4.5.8. iOS 小程序添加扩展信息

在控制台使用小程序发布功能时，需要新增小程序包。配置小程序包中的扩展信息必须使用 launchParams 包裹，否则 iOS 端将无法获取到改扩展信息。

获取扩展信息

```
    NAMApp *app = [NAMServiceGet() findApp:appId version:@"1.0.0.2"];
    NSString *extend_info = app.extend_info;
```

1.4.5.9. H5 容器定位偏移解决方案

在使用 mPaaS 容器的过程中可能会遇到 H5 容器定位偏移的问题，请参考以下方法进行设置更新：

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)la
unchOptions
{

    //跳过 LBS 定位
    [LBSmPaaSAdaptor sharedInstance].shouldSkipLBSLocation = YES;

    .......

}
```

1.5. 接入 HarmonyOS NEXT

1.5.1. 快速开始

本文介绍如何将 H5 容器组件接入到 HarmonyOS NEXT 客户端。您可以基于已有工程使用 ohpmrc 方式接入 H5 容器 SDK 到客户端。

前置条件

- 添加 H5 容器 SDK 前，请您确保已经将工程接入到 mPaaS。更多信息请参见开发指南 接入 mPaaS 能力。
- 完成 mppm 插件安装。详情请参考 安装 mppm 工具。

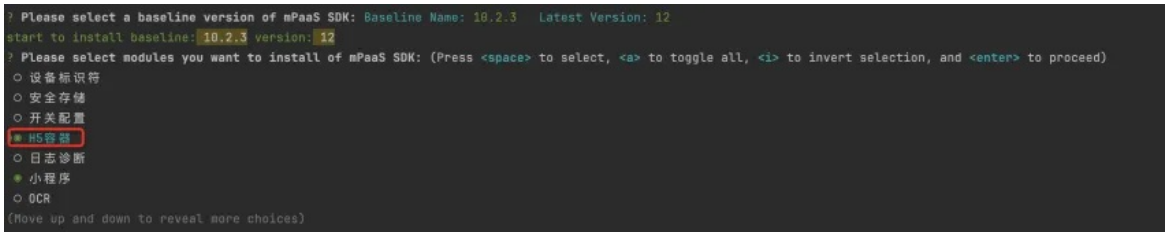
引入依赖

在项目的 `.ohpmrc` 文件中添加如下仓库：

```
@mpaas:registry=https://mpaas-ohpm.oss-cn-hangzhou.aliyuncs.com/meta
```

添加 SDK

通过 使用 mppm 工具 安装 **H5 容器** 组件。



配置权限

在 `module.json5` 中配置所需权限。

```
"requestPermissions":[
  {
  "name" : "ohos.permission.GET_NETWORK_INFO",
  },
  {
  "name" : "ohos.permission.INTERNET",
  }
]
```

使用 SDK

初始化

在 mPaaS 框架初始化完成之后，初始化 `HRiver` 。代码如下：

```
HRiver.init();
```

使用 SDK 之前必须初始化 mPaaS 框架，设置 userId 和 appsecret，其中 appsecret 从 AppCenter 后台获取，具体查看 获取 HarmonyOS NEXT config 配置文件 。

代码示例如下：

```
import AbilityStage from '@ohos.app.ability.AbilityStage';
import { MPFramework } from '@mpaas/framework';

export default class ModuleEntry extends AbilityStage {
  async onCreate() {
    MPFramework.create(this.context);

    MPFramework.instance.userId = 'MPTestCase'
    MPFramework.instance.appSecret = "12a711d78980f661aca5401788fdf09a";
  }
}
```

打开离线包

初始化 `HRiver` 后调用 `startApp` 打开离线包。

```
import { HRiver } from '@mpaas/hriver'

/**
* 打开离线包
* @param appId: 离线包id
* @param startParams: 启动参数，可以不传。控制TitleBar、指定页面等
*/
HRiver.startApp(appId: string, startParams?: Map<string, Object>)
```

代码示例如下：

```
import { HRiver } from '@mpaas/hriver'

// 示例1:
HRiver.startApp('20190517') // 直接启动离线包

// 示例2:
let startParams: Map<string, Object> = new Map();
startParams.set('defaultTitle', '默认标题') //加载阶段显示默认标题
HRiver.startApp('20190517', startParams)
```

打开在线页面

初始化 `HRiver` 之后调用 `startUrl` 打开在线页面。

```
import { HRiver } from '@mpaas/hriver'

/**
* 打开在线页面
* @param url: 在线地址
* @param startParams: 启动参数。可以不传，控制TitleBar、指定页面等
*/
HRiver.startUrl(url: string, startParams: Map<string, Object>)
```

注册自定义 JSAPI

初始化 `HRiver` 之后调用 `registerPlugin` 注册自定义 JSAPI。

```
import { HRiver } from '@mpaas/hriver'

/**
* 注册自定义 JSApi 实现
* @param pluginClass: pluginClass列表，如 registerPlugin({CustomPlugin1, CustomPlugin2},
HRiver.SCOPE_PAGE)
* @param scope: 可以不传。默认HRiver.SCOPE_PAGE。HRiver.SCOPE_APP表示离线包App级别生命周期；HRiver.SCOPE_PAGE
表示页面级别生命周期
*/
HRiver.registerPlugin(pluginClass: ESObject, scope: string)
```

代码示例：

```
HRiver.registerPlugin({H5CustomPlugin, H5Custom1Plugin})
```

plugin 实现代码示例如下：

```
import { HRiver, H5SimplePlugin, H5EventFilter, H5Event, H5BridgeContext } from '@mpaas/hriver'

class H5CustomPlugin extends H5SimplePlugin {
  onPrepare(filter: H5EventFilter): void {
    filter.addAction('myapi1')
  }

  handleEvent(event: H5Event, context: H5BridgeContext): Boolean {
    if ('myapi1' == event.action) {
      context.sendBridgeResult({
        success: true,
        data: 'myapi1调用成功'
      })
      return true
    }
    return super.handleEvent(event, context);
  }
}

class H5Custom1Plugin extends H5SimplePlugin {
  onPrepare(filter: H5EventFilter): void {
    filter.addAction('myapi2')
  }

  handleEvent(event: H5Event, context: H5BridgeContext): Boolean {
    if ('myapi2' == event.action) {
      context.sendBridgeResult({
        success: true,
        data: 'myapi2调用成功'
      })
      return true
    }
    return super.handleEvent(event, context);
  }
}
```

Native 调用 H5

Native 调用 H5 有以下两种方法。

- 在自定义 JSAPI 中通过 `H5BridgeContext.sendToWeb` 方法调用 H5。

```
import { H5BridgeContext, H5Event, H5EventFilter, H5SimplePlugin, HRiver } from '@mpaas/hriver';
class H5CustomPlugin extends H5SimplePlugin {
  handleEvent(event: H5Event, context: H5BridgeContext): Boolean {
    // native 调用 h5
    context.sendToWeb('customCallWeb', {
        data: 'abc'
    })

    ... // 其他代码
  }
}
```

- 在 Native 代码中获取 `TopApp` 的 `activityPage`，获得最新的页面，通过页面调用 `sendToWeb` 方法。

```
import { HRiver,
  XRiverProxy,
  getProxy,
  AppManager,
  AppNode,
  Page
} from '@mpaas/hriver';
{
  let appManager = getProxy(XRiverProxy.AppManager) as AppManager
  let appNode: AppNode | null = appManager.findTopApp()
  if (appNode != null) {
    let page: Page | null = appNode.getActivePage()
    if (page != null) {
      page.sendToWeb('testAction', {data: ''})
    }
  }
}
```

加载内置离线包

加载所有内置离线包，包括内置的公共离线包。

初始化 `HRiver` **之后调用** `loadOfflineResource` **加载内置离线包。**

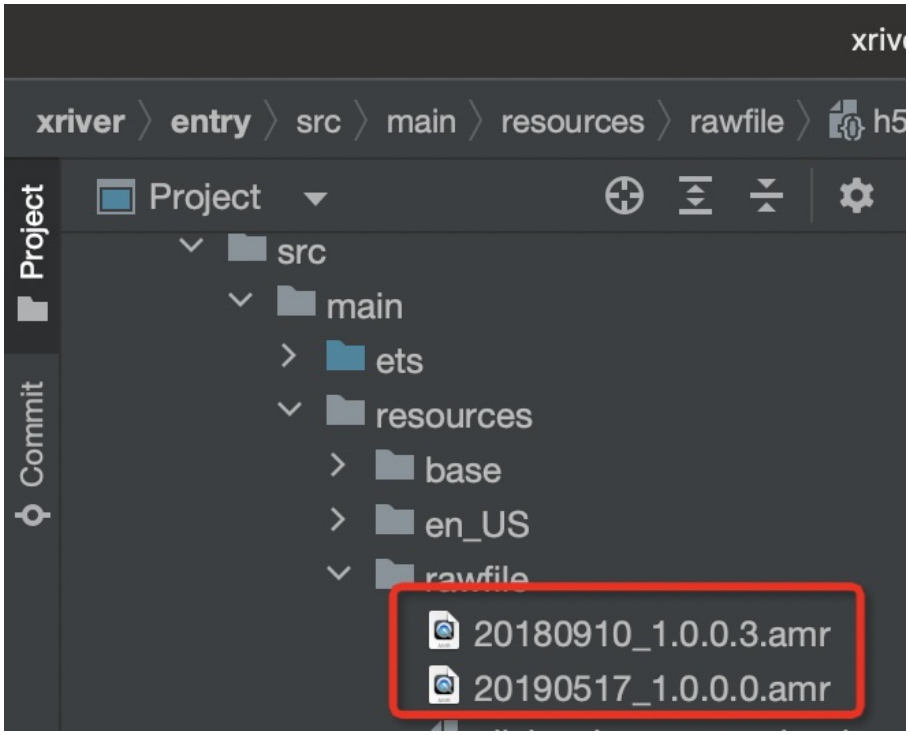
```
import { HRiver } from '@mpaas/hriver'

/**
* 加载内置离线包
* @param jsonFileName: 内置离线包的 h5_json.json 的文件名，放到rawfile目录中。如：h5_json.json。
* @param callback: 格式 (result: string) => {}。内置离线包加载完成回调的 Function
*/
HRiver.loadOfflineResource(jsonFileName: string, callback: Function)
```

代码示例如下：

1. **在** `entry/src/main/resources/rawfile` **下添加** `h5_json.json` **（文件从 Appcenter 后台下载即可）。**

```
{
   "config":{
      "updateReqRate":16400,
      "limitReqRate":13600,
      "appPoolLimit":3,
      "versionRefreshRate":86400
   },
   "data":[
      {
         "app_desc":"离线包1",
         "app_id":"20180910",
         "auto_install":1,
         "fallback_base_url":"https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/20180910/1.0.0.3_all/nebula/fallback/",
         "global_pack_url":"",
         "icon_url":"",
         "installType":1,
         "main_url":"/www/index.html",
         "name":"离线包1",
         "online":1,
         "package_url":"https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/20180910/1.0.0.3_all/nebula/20180910_1.0.0.3.amr",
         "patch":"",
         "sub_url":"",
         "version":"1.0.0.3",
         "vhost":"https://20180910.h5app.com"
      },
        {
            "app_desc":"离线包2",
            "app_id":"20190517",
            "auto_install":1,
            "fallback_base_url":"https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/20190517/1.0.0.0_all/nebula/fallback/",
            "global_pack_url":"",
            "icon_url":"",
            "installType":1,
            "main_url":"/www/index.html",
            "name":"离线包2",
            "online":1,
            "package_url":"https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/20190517/1.0.0.0_all/nebula/20190517_1.0.0.0.amr",
            "patch":"",
            "sub_url":"",
            "version":"1.0.0.0",
            "vhost":"https://20190517.h5app.com"
        }
   ],
   "resultCode":100,
   "resultMsg":"操作成功",
   "state":"success"
}
```


2. 下载 amr 并命名为 `${appid}_${version}.amr` 。



3. 调用 API。

```
import { HRiver } from '@mpaas/hriver'

HRiver.loadOfflineResource('h5_json.json', (result: string) => {
  this.resultText = this.resultText + `\n${result} 预加载成功`
})
```

更新离线包

```
import { HRiver } from '@mpaas/hriver'

/**
 * 批量更新离线包
 * @param appIds: 需要更新离线包的appId列表
 * @param updateCallback: 格式必须 (result: boolean, code: number) => {}。更新接口返回后回调的Function
 */
HRiver.updateApp(appIds?: Array<string>, updateCallback?: Function)

/**
 * 更新所有离线包。
 * @param updateCallback: 格式必须 (result: boolean, code: number) => {}。更新接口返回后回调的Function
 */
HRiver.updateAll(updateCallback: Function)

/**
 * 批量更新离线包
 * @param appIds: 离线包的appId和version的map
 * @param updateCallback: 格式必须 (result: boolean, code: number) => {}。更新接口返回后回调的Function
 */
HRiver.updateAppWithVersion(appIds?: Map<string, string>, updateCallback?: Function)
```

代码示例如下：

```
import { HRiver } from '@mpaas/hriver'

HRiver.updateApp(['90000002'], (result: boolean, code: number) => {
  this.resultText = `90000002更新结果： ${result}`
})
```

配置公共离线包

1. 设置 `H5CommonAppProvider` 。

```
import { HRiver, H5CommonAppProvider } from '@mpaas/hriver'

/**
* 配置公共离线包 Provider，必须在 HRiver.init() 之前调用
*/
HRiver.setProvider(H5CommonAppProvider.name, new H5AppCommonProviderImpl())
```

> **说明**
> `setProvider` 必须在 `HRiver.init()` 之前调用。

2. 实现 `H5AppCommonProviderImpl.ets` 的 `getCommonResourceAppList` 方法。

```
import { HRiver, H5CommonAppProvider } from '@mpaas/hriver'

export class H5AppCommonProviderImpl extends H5CommonAppProvider {
  getCommonResourceAppList(): Array<string> {
    return ['20220719'] // 返回公共离线包id列表
  }
}
```

支持 fallback 逻辑

保持和 iOS/Android 一致，离线包没下载成功情况下优先打开 fallback 在线地址。

该功能默认关闭，可以通过以下开关打开：

实现 H5CommonAppProvider 的 configJSON 方法，增加 enableFallback 参数。

```
export class H5AppCommonProviderImpl extends H5CommonAppProvider {
  ... // 其他配置

  // configJson，配置更新频率等
  configJSON(): string {
    return JSON.stringify({
      enableFallback: 'YES',
      xxx // 其他配置
    })
  }
}
```

设置 UserAgent

```
import { HRiver } from '@mpaas/hriver'

HRiver.setUserAgent(userAgent)
```

> **说明**
> `setUserAgent` 在 `HRiver.init()` 之后调用，会在默认 `UserAgent` 之后拼接设置的 `useragent` 。

设置离线包默认更新频率

1. 设置 `H5CommonAppProvider` 。

```
import { HRiver, H5CommonAppProvider } from '@mpaas/hriver'

/**
* 配置公共离线包Provider、离线包更新频率等功能，必须在HRiver.init()之前调用
*/
HRiver.setProvider(H5CommonAppProvider.name, new H5AppCommonProviderImpl())
```

> **说明**
> `setProvider` 必须在 `HRiver.init()` 之前调用。

2. 实现 `H5CommonAppProvider` 的 `configJSON` 方法，代码示例如下。

```
import { HRiver, H5CommonAppProvider } from '@mpaas/hriver'

export class H5AppCommonProviderImpl extends H5CommonAppProvider {
  ... // 其他配置

  // configJson，配置更新频率等
  configJSON(): string {
    return JSON.stringify({
      h5_nbmngconfig: "{\"config\":{\"al\":\"3\",\"pr\":
{\"4\":\"86400\",\"common\":\"864000\"},\"ur\":\"1\",\"fpr\":
{\"common\":\"3888000\"}},\"switch\":\"yes\"}"
    })
  }
}
```

具体参数如下：

```
h5_nbmngconfig: "{\"config\":{\"al\":\"3\",\"pr\":
{\"4\":\"86400\",\"common\":\"864000\"},\"ur\":\"1800\",\"fpr\":
{\"common\":\"3888000\"}},\"switch\":\"yes\"}"
```

其中的 `ur: 1800` 表示更新频率为 1800 秒，使用时修改 `ur` 的值即可。

配置离线包签名校验

1. 设置 `H5CommonAppProvider` 。

```
import { HRiver, H5CommonAppProvider } from '@mpaas/hriver'

/**
* 配置公共离线包Provider、离线包更新频率、签名校验等功能，必须在HRiver.init()之前调用
*/
HRiver.setProvider(H5CommonAppProvider.name, new H5AppCommonProviderImpl())
```

> **说明**
> `setProvider` 必须在 `HRiver.init()` 之前调用。

2. 实现 `H5CommonAppProvider` 的 `shouldVerify` 和 `pubKey` 方法，代码示例如下：

```
import { HRiver, H5CommonAppProvider } from '@mpaas/hriver'

export class H5AppCommonProviderImpl extends H5CommonAppProvider {
 ... // 其他配置

 /**
 * 是否开启离线包校验，默认关闭
 * return: true表示开启，false表示关闭
 */
 shouldVerify(): boolean {
 return false
 }

 // 离线包校验的公钥，如果shouldVerify为false 则无需设置，否则必须设置公钥
 pubKey(): string {
 return ''
 }
}
```

开启调试模式

```
HRiver.enableDebug(true)
```

监听页面生命周期

初始化完成后设置 provider。

```
import {H5PageLifeCycleProvider} from '@mpaas/hriver';

HRiver.setProvider(H5PageLifeCycleProvider.name, new H5PageLifeCycle())
```

H5PageLifeCycle

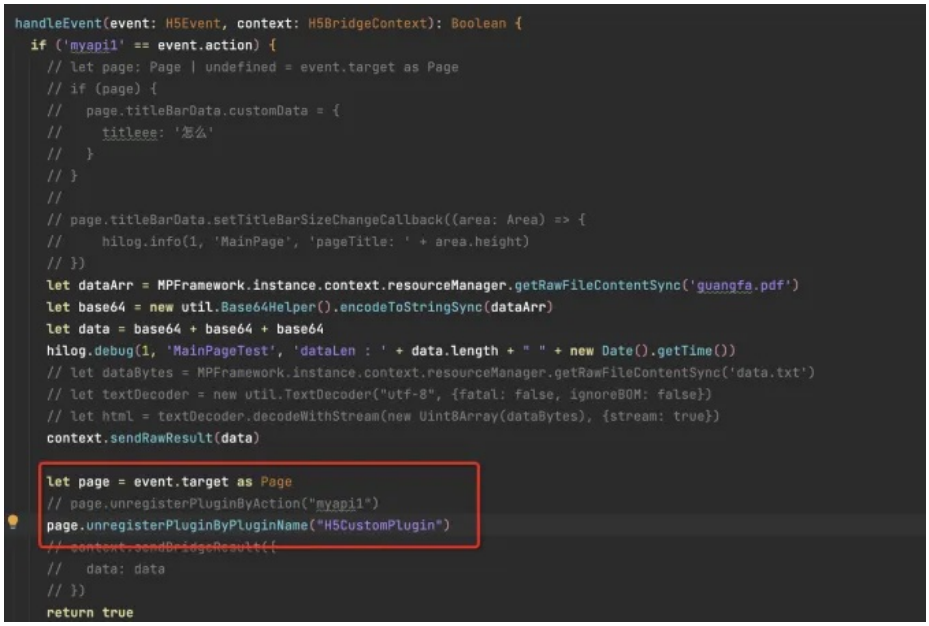
```
import { H5PageLifeCycleProvider, Page } from '@mpaas/hriver';
import { hilog } from '@kit.PerformanceAnalysisKit';

export class H5PageLifeCycle extends H5PageLifeCycleProvider {
  onPageShow(routerName: string, page?: Page | undefined): void {
    super.onPageShow(routerName, page);
    hilog.debug(1, 'H5PageLifeCycle', "pageshow: " + page?.pageUrl)
  }

  onPageHide(routerName: string, page?: Page | undefined): void {
    super.onPageHide(routerName, page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageHide: " + page?.pageUrl)
  }

  onPageCreate(page?: Page | undefined): void {
    super.onPageCreate(page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageCreate: " + page?.pageUrl)
  }

  onPageExit(page?: Page | undefined): void {
    super.onPageExit(page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageExit: " + page?.pageUrl)
  }

  onBackPress(page?: Page | undefined): boolean {
    hilog.debug(1, 'H5PageLifeCycle', "onBackPress: " + page?.pageUrl)
    return super.onBackPress(page);

  }
}
```

支持注销 Plugin

通过 Page 的 Page.ets 实现注销。

```
// 根据action注销
unregisterPluginByAction(action: string);

// 根据pluginName注销
unregisterPluginByPluginName(name: string);
```

示例如下：



```
handleEvent(event: H5Event, context: H5BridgeContext): Boolean {
  if ('myapi1' == event.action) {
    // let page: Page | undefined = event.target as Page
    // if (page) {
    //   page.titleBarData.customData = {
    //     titleee: '怎么'
    //   }
    // }
    //
    // page.titleBarData.setTitleBarSizeChangeCallback((area: Area) => {
    //     hilog.info(1, 'MainPage', 'pageTitle: ' + area.height)
    // })
    let dataArr = MPFramework.instance.context.resourceManager.getRawFileContentSync('guangfa.pdf')
    let base64 = new util.Base64Helper().encodeToStringSync(dataArr)
    let data = base64 + base64 + base64
    hilog.debug(1, 'MainPageTest', 'dataLen : ' + data.length + " " + new Date().getTime())
    // let dataBytes = MPFramework.instance.context.resourceManager.getRawFileContentSync('data.txt')
    // let textDecoder = new util.TextDecoder("utf-8", {fatal: false, ignoreBOM: false})
    // let html = textDecoder.decodeWithStream(new Uint8Array(dataBytes), {stream: true})
    context.sendRawResult(data)

    let page = event.target as Page
    // page.unregisterPluginByAction("myapi1")
    page.unregisterPluginByPluginName("H5CustomPlugin")
    // context.sendBridgeResult({
    //   data: data
    // })
    return true
```

支持页面嵌入模式

页面嵌入模式基于 Navigation 实现。

1. 创建离线包需要的 NavPathStack。

```
pageInfos: NavPathStack = new NavPathStack()
```

2. 在页面需要嵌入的位置添加空白组件，以下为示例：

> **重要**
>
> mode 必须使用 `NavigationMode.Stack`，否则横竖屏/折叠屏切换有问题。

```
Navigation(this.pageInfos) {
          Column() {
            // 空白页面用于嵌入离线包页面，不用填任何内容
          }.width('100%') // 宽度根据需要
          .backgroundColor(Color.Red) // 只是示例
          .height(500)  // 高度根据实际
        }.navDestination(this.PagesMap)
        .mode(NavigationMode.Stack)


// pagesMap实现：
import {HRBuilder, RouterUtils, HRiver, H5RouterNavStackProvider} from '@mpaas/hriver'

let mPaaSHRiverBuilder: WrappedBuilder<[string, ESObject]> = wrapBuilder(HRBuilder);

@Builder
PagesMap(name: string, params: ESObject) {
  if (RouterUtils.isMPHRiverPage(name)) {
    mPaaSHRiverBuilder.builder(name, params)
  } else if (name == 'xxx') {
    // 其他业务的页面
  }
}
```

3. 启动离线包和在线页面时，需要添加第三个参数并传入步骤 1 中创建的 NavPathStack，增加 **embedPage** 参数用来表示内嵌页面。

```
let param: Map<string, Object> = new Map<string, Object>()
param.set('embedPage', 'YES')
HRiver.startUrl('https://www.baidu.com', param, this.pageInfos)
// HRiver.startApp('90000000', param, this.pageInfos)
```

4. 添加返回事件拦截。

   HRiver.setProvider(H5PageLifeCycleProvider.name, new H5PageLifeCycle())

```
import { H5PageLifeCycleProvider, Page } from '@mpaas/hriver';
import { hilog } from '@kit.PerformanceAnalysisKit';
import { router } from '@kit.ArkUI';

export class H5PageLifeCycle extends H5PageLifeCycleProvider {

  onPageShow(routerName: string, page?: Page | undefined): void {
    super.onPageShow(routerName, page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageShow: " + page?.pageUrl)

  }
  onPageHide(routerName: string, page?: Page | undefined): void {

    super.onPageHide(routerName, page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageHide: " + page?.pageUrl)
  }
  onPageCreate(page?: Page | undefined): void {

    super.onPageCreate(page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageCreate: " + page?.pageUrl)
  }
  onPageExit(page?: Page | undefined): void {

    super.onPageExit(page);
    hilog.debug(1, 'H5PageLifeCycle', "onPageExit: " + page?.pageUrl)
  }
  onBackPress(page?: Page | undefined): boolean {
    hilog.debug(1, 'H5PageLifeCycle', "onBackPress: " + page?.pageUrl)
    let navPathStack: NavPathStack|undefined = page?.getSession()?.getRouter()?.getNavPathStack()
    if (navPathStack && page && page.embedPage && navPathStack.size() == 1) {
      router.back()
      return true;
    }
    return super.onBackPress(page);
  }
}
```

UI 定制

自定义导航栏

1. 初始化完成后通过 `provider` 设置自定义导航栏。

```
import {HRBuilder, RouterUtils, HRiver, H5CommonAppProvider, H5RouterNavStackProvider,
  CustomUIBuilderProvider,
  H5CacheProvider
} from '@mpaas/hriver'

HRiver.setProvider(CustomUIBuilderProvider.name, new CustomUIBuilderProviderImpl())
```

2. 实现 `CustomUIBuilderProviderImpl` 。

```
import { CustomUIBuilderProvider, Page } from '@mpaas/hriver';
import { CustomUIBuilder } from '../pages/CustomTitleBarComponent';

export class CustomUIBuilderProviderImpl extends CustomUIBuilderProvider {
  getCustomUIBuilder(): WrappedBuilder<[string, Page]> {
    return wrapBuilder(CustomUIBuilder);
  }
}
```

其中 `CustomUIBuilder` 为业务自定义 `titlebar` 组件的全局 Builder。实现参考如下：

```
/**
*  name: 自定义组件的名称
*  page: 自定义titlebar对应的页面Page
*/
@Builder
export function CustomUIBuilder(name: string, p: Page) {
  if (name === 'titleBar') {
    CustomTitleBarComponent({page: p, titleBarData: p.titleBarData})
  }
}
```

`CustomTitleBarComponent` 为具体的标题组件，`titleBarData` 为标题栏所需要的数据，数据发生变化会实时刷新。示例参考如下：

```
import { H5NavMenuItemData, HRiverUtil, Page, TitleBarData } from '@mpaas/hriver'

const TAG: string = "CustomTitleBarComponent"


const MENU_MARGIN: number = 5
const DEFAULT_MARGIN: number = 12

@Builder
export function CustomUIBuilder(name: string, p: Page) {
  if (name === 'titleBar') {
    CustomTitleBarComponent({page: p, titleBarData: p.titleBarData})
  }
}

@Component
export struct CustomTitleBarComponent {
  page: Page | null = null
  @Prop titleBarData: TitleBarData

  aboutToAppear(): void {
  }

  build() {
    RelativeContainer() {
      Button() {
        Image(this.getBackIconImage())
          .id('titlebar_back_img')
          .width(12)
          .height(20)
      }
      .width(48)
      .height('100%')
      .borderRadius(0)
      .backgroundColor(Color.Transparent)
      .align(Alignment.Center)
      .alignRules({
        top: { anchor: '__container__', align: VerticalAlign.Top }
      })
```

```
      .id("h5_nav_close")
      .onClick((event) => {
        if (this.page != null) {
          this.page.backClickEvent()
        }
      })

      Flex({ justifyContent: FlexAlign.Center, direction: FlexDirection.Column }) {
        //如果设置了图片标题就只显示图片
        if (this.titleBarData.titleImage) {
          Image(this.titleBarData.titleImage)
            .id('titlebar_title_image')
            .height(36).onClick(() => {
            this.onTitleClick()
          })
        } else {
          Flex({ justifyContent: FlexAlign.Start,direction: FlexDirection.Row }){
            Text(this.titleBarData.title)
              .fontSize(18)
              .textAlign(TextAlign.Start)
              .textOverflow({ overflow: TextOverflow.Ellipsis })
              .maxLines(1)
              .fontColor(this.titleBarData.titleColor)
              .onClick(() => {
                this.onTitleClick()
              })
            if(this.titleBarData.showTitleLoading ){

              Image(this.getTitleBarLoadingIcon()).width(18).height(18)
                .id('titlebar_progress_img')
                .margin({left:5,top:1})
                .rotate({ angle: this.titleBarData.loadingRotateAngle })
                .animation({
                  duration:3000,
                  curve: Curve.Linear,
                  delay: 0,
                  iterations: -1,
                  playMode: PlayMode.Normal,
                }).onAppear(()=>{
                this.titleBarData.loadingRotateAngle = 360
              })

            }

          }

          if (this.titleBarData.subtitle) {
            Text(this.titleBarData.subtitle)
              .textAlign(TextAlign.Start)
              .fontColor(this.titleBarData.titleColor)
              .textOverflow({ overflow: TextOverflow.Ellipsis })
              .maxLines(1)
              .fontSize(18)
              .onClick(() => {
                this.onTitleSubtitleClick()
              })
          }
        }

      }.id("h5_tv_title")
      .height("100%")
      .alignRules({
        top: { anchor: '__container__', align: VerticalAlign.Top },
```

```
        left: { anchor: 'h5_nav_close', align: HorizontalAlign.End },
        right: { anchor: "h5_nav_options", align: HorizontalAlign.Start }
      })

      if (this.titleBarData.optionMenuState) {
        if (this.titleBarData.menuType == TitleBarData.MENU_TYPE_TITLE) {
          Text(this.titleBarData.menuTitle)
            .fontSize(16)
            .align(Alignment.Center)
            .textAlign(TextAlign.Center)
            .fontColor(this.titleBarData.menuColor)
            .height('100%')
            .id("h5_nav_options")
            .alignRules({
              top: { anchor: '__container__', align: VerticalAlign.Top },
              right: { anchor: '__container__', align: HorizontalAlign.End }
            })
            .onClick(() => {
              this.onMoreClick(false, 0)
            })
            .margin({
              right: DEFAULT_MARGIN
            })
        } else if (this.titleBarData.menuType == TitleBarData.MENU_TYPE_ICON) {
          Button() {
            Image(HRiverUtil.getIconImage(this.titleBarData.menuIcon))
              .id('titlebar_right_icon')
              .width(22)
              .height(22)
              .objectFit(ImageFit.Contain)
          }
          .width(30)
          .borderRadius(0)
          .backgroundColor(Color.Transparent)
          .align(Alignment.Center)
          .id("h5_nav_options")
          .height("100%")
          .onClick(() => {
            this.onMoreClick(false, 0)
          })
          .alignRules({
            top: { anchor: '__container__', align: VerticalAlign.Top },
            right: { anchor: '__container__', align: HorizontalAlign.End }
          })
          .margin({
            right: DEFAULT_MARGIN
          })
        } else if (this.titleBarData.menuType == TitleBarData.MENU_TYPE_MORE) {
          Button() {
            Image(HRiverUtil.getIconImage('more'))
              .id('titlebar_right_more')
              .width(22)
              .height(22)
              .objectFit(ImageFit.Contain)
          }
          .width(48)
          .borderRadius(0)
          .backgroundColor(Color.Transparent)
          .align(Alignment.Center)
          .id("h5_nav_options")
          .height("100%")
          .margin({
            right: DEFAULT_MARGIN
          })
```

```
      .alignRules({
        top: { anchor: '__container__', align: VerticalAlign.Top },
        right: { anchor: '__container__', align: HorizontalAlign.End }
      }).onClick(() => {
        if (!this.titleBarData.preventDefault) {
          this.titleBarData.customPopup = !this.titleBarData.customPopup
        }

        this.onMoreClick(true, 0)
      })
      .bindPopup(this.titleBarData.customPopup , {
        builder: this.MenuBuilder,
        placement:Placement.BottomLeft,
        popupColor:"#fff",
        onStateChange: (e) => {
          console.info(JSON.stringify(e.isVisible))
          if (!e.isVisible) {
            this.titleBarData.customPopup = false
          }
        }
      })
    }

    if (this.titleBarData.menuType1 == TitleBarData.MENU_TYPE_TITLE) {
      Text(this.titleBarData.menuTitle1)
        .fontSize(16)
        .align(Alignment.Center)
        .textAlign(TextAlign.Center)
        .fontColor(this.titleBarData.menuColor1)
        .height('100%')
        .id("h5_nav_options1")
        .alignRules({
          top: { anchor: '__container__', align: VerticalAlign.Top },
          right: { anchor: 'h5_nav_options', align: HorizontalAlign.Start }
        })
        .margin({
          right: MENU_MARGIN
        })
        .onClick(()=> {
          this.onMoreClick(false, 1)
        })
    } else if (this.titleBarData.menuType1 == TitleBarData.MENU_TYPE_ICON) {
      Button() {
        Image(HRiverUtil.getIconImage(this.titleBarData.menuIcon1))
          .id('titlebar_right_icon1')
          .width(22)
          .height(22)
          .objectFit(ImageFit.Contain)
      }
      .width(30)
      .borderRadius(0)
      .backgroundColor(Color.Transparent)
      .align(Alignment.Center)
      .id("h5_nav_options1")
      .height("100%")
      .onClick(()=> {
        this.onMoreClick(false, 1)
      })
      .alignRules({
        top: { anchor: '__container__', align: VerticalAlign.Top },
        right: { anchor: 'h5_nav_options', align: HorizontalAlign.Start }
      })
      .margin({
```

```
            right: MENU_MARGIN
          })
        } else if (this.titleBarData.menuType1 == TitleBarData.MENU_TYPE_MORE) {
          Button() {
            Image(HRiverUtil.getIconImage('more'))
              .id('titlebar_right_more1')
              .width(22)
              .height(22)
              .objectFit(ImageFit.Contain)
          }
          .width(48)
          .borderRadius(0)
          .backgroundColor(Color.Transparent)
          .align(Alignment.Center)
          .id("h5_nav_options1")
          .height("100%")
          .margin({
            right: MENU_MARGIN
          })
          .alignRules({
            top: { anchor: '__container__', align: VerticalAlign.Top },
            right: { anchor: 'h5_nav_options', align: HorizontalAlign.Start }
          }).onClick(() => {
            if (!this.titleBarData.preventDefault) {
              this.titleBarData.customPopup1 = !this.titleBarData.customPopup1
            }

            this.onMoreClick(true, 1)
          })
          .bindPopup(this.titleBarData.customPopup1 , {
            builder: this.MenuBuilder,
            placement:Placement.BottomLeft,
            popupColor:"#fff",
            onStateChange: (e) => {
              console.info(JSON.stringify(e.isVisible))
              if (!e.isVisible) {
                this.titleBarData.customPopup1 = false
              }
            }
          })
        }

      }



    }.height(this.titleBarData.showTitleBar ? 48 : 0)
  }

  getTitleBarLoadingIcon(): Resource | string {
    return $rawfile(`icon/h5_title_bar_progress_bg.webp`)
  }

  getIconImage(icon: string): Resource | string {
    return HRiverUtil.getIconImage(icon)
  }

  getBackIconImage(): Resource | string {
    return $rawfile(`icon/hriverback.webp`)
  }

  onTitleClick() {
    if (this.page != null) {
      this.page.titleBarClickEvent()
```

```
    }

  }

  onTitleSubtitleClick() {
    if (this.page != null) {
      this.page.subTitleBarClickEvent()
    }

  }

  onMoreClick(fromMenu:boolean, index: number) {

    if (this.page != null) {
      this.page.onMoreClick(fromMenu, index)
    }

  }
  onMoreItemClick(tag: string, name: string,isShowPopMenu:boolean) {
    if (this.page != null) {
      this.page.onMoreItemClick(tag, name, isShowPopMenu)
    }

  }

  @Builder
  MenuBuilder() {
    Flex({ direction: FlexDirection.Column, justifyContent: FlexAlign.Center, alignItems:
ItemAlign.Center }) {
      ForEach(this.titleBarData.h5NavMenuItemList, (item: H5NavMenuItemData, index) => {
        Column() {
          Row() {
            Image(item.icon).width(20).height(20).margin({ right: 5 })
            Text(item.name).fontSize(16)
          }
          .width('100%')
          .height(30)
          .padding({ left: 10 })
          .justifyContent(FlexAlign.Start)
          .align(Alignment.Center)
          .onClick(() => {
            if (item) {
              this.onMoreItemClick(item.tag || '', item.name || '',false)
              this.titleBarData.customPopup = false
              this.titleBarData.customPopup1 = false
            }

          })

          if (index != this.titleBarData.h5NavMenuItemList.length - 1) {
            Divider().height(10).width('90%').color('#ccc')
          }
        }.padding(5).height(40)
      })
    }.width(150).height(165).backgroundColor("#fff")

  }

}
```

自定义离线包加载页/错误页

1. 初始化完成之后设置 Provider。

```
import {
  CustomLoadingBuilderProvider} from '@mpaas/hriver';

HRiver.setProvider(CustomLoadingBuilderProvider.name,  new CustomLoadingBuilderProviderImpl())
```

2. 实现 `CustomLoadingBuilderProviderImpl` 类。

```
import { CustomLoadingBuilderProvider, H5Router, HRLoadingData } from '@mpaas/hriver';
import { CustomUIBuilder } from './CustomLoadingComponent';

export class CustomLoadingBuilderProviderImpl extends CustomLoadingBuilderProvider {
  getCustomUIBuilder(): WrappedBuilder<[string, HRLoadingData, H5Router]> {
    return wrapBuilder(CustomUIBuilder);
  }
}
```

3. 实现 `CustomLoadingComponent.ets` 类，其中通过 `loadingStatus` 控制加载状态。

```
import { H5Router, HRLoadingData, LoadingStatus } from '@mpaas/hriver'

const TAG: string = "CustomLoadingComponent"


export const Loading_STATE_Init = 0
export const Loading_STATE_Start = 1
export const Loading_STATE_End = 2
export const Loading_STATE_Err = 3
@Builder
export function CustomUIBuilder(name: string, loadingData: HRLoadingData, h5Router: H5Router) {
  if (name === 'loading') {
    CustomLoadingComponent({loadingStatus: loadingData.loadingStatus, h5Router: h5Router})
  }
}

@Component
export struct CustomLoadingComponent {
  @ObjectLink loadingStatus: LoadingStatus
  h5Router?: H5Router

  aboutToAppear(): void {
  }

  build() {
    Row() {
      Column() {
        Flex({ direction: FlexDirection.Row }) {
          //返回按钮
          Button() {
            Image($rawfile("icon/hriverback.webp"))
              .width(12)
              .height(20)
          }
          .width(48)
          .height('100%')
          .borderRadius(0)
          .backgroundColor(Color.Transparent)
          .align(Alignment.Center)
          .onClick((event) => {
            this.h5Router?.routerBack()
          })

        }.width('100%')
        .height(48)
        Image(this.loadingStatus.icon ? this.loadingStatus.icon :
```

```
$rawfile("icon/hriverloading.webp"))
          .width(40)
          .height(40)
          .margin({top: 38})
        if (this.loadingStatus.state == Loading_STATE_Err ||
          this.loadingStatus.state == Loading_STATE_Start){
          Text(this.loadingStatus.state == Loading_STATE_Err ? `网络不给力，请稍后再试
\n(${this.loadingStatus.code} ${this.loadingStatus.msg})` : this.loadingStatus.title)
            .fontSize(18)
            .margin({top: 15})
            .textAlign(TextAlign.Center)
        }

      }
      .width('100%')
      .margin({
        top: 48
      })
    }
  }


}
```

自定义在线 URL 加载失败的错误页

1. 设置 Provider 拦截网页错误回调。

```
HRiver.setProvider(H5WebClientProvider.name, new H5WebClientProviderImpl());
```

2. 在 `H5WebClientProviderImpl` 中实现 `onErrorReceive` 方法。

```
import { MPFramework } from '@mpaas/framework';
import {  H5WebClientProvider, Page } from '@mpaas/hriver';
import { util } from '@kit.ArkTS';

export class H5WebClientProviderImpl  extends H5WebClientProvider{

  onErrorReceive(page: Page | undefined, request: WebResourceRequest | undefined, err:
WebResourceError | undefined): boolean {
    let errorUrl = request?.getRequestUrl()
    let errorCode = err?.getErrorCode()

    if (errorCode == 403 || errorCode == 404) {
      // keep same with ios,not show errorPage for 404 and 403
      // log(TAG, "ignoreErrorPage 404 or 403, return ");
      return true;
    }
    let lastUrl = page?.webcontroller?.getUrl()

    if (errorUrl == page?.pageUrl || errorUrl == `${page?.pageUrl}/`) {
      // 从rawfile中读取自定义错误页面demo_custom_err.html

      let dataBytes =
MPFramework.instance.context.resourceManager.getRawFileContentSync('demo_custom_err.html')
      let textDecoder = new util.TextDecoder("utf-8", { fatal: false, ignoreBOM: false })
      let html = textDecoder.decodeWithStream(new Uint8Array(dataBytes), { stream: true })

      page?.webcontroller?.loadData(html, "text/html", "utf-8", lastUrl)
      return true;
    }
    return false
  }
}
```

3. 在 `demo_custom_err.html` 文件中编写错误页代码。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="apple-mobile-web-app-capable" content="yes"/>
    <meta name="apple-mobile-web-app-status-bar-style" content="black"/>
    <meta name="format-detection" content="telephone=no"/>
    <meta name="format-detection" content="email=no"/>
    <meta name="viewport"
          content="width=device-width,initial-scale=1,maximum-scale=1,minimum-scale=1,user-
scalable=0"/>
    <title>!!!!</title>
    <style type="text/css">
        body {
        background-color: #FFF;
        }

        .am-page-result {
        text-align: center;
        }

        .am-page-result .am-page-result-pic {
        width: 135px;
        height: 135px;
        margin: 40px auto;
        }

        .am-page-result .am-page-result-pic img {
```

```
width: 100%;
height: 100%;
}

.am-page-result p {
margin: 0;
font-size: 16px;
color: #999;
}

.am-page-result-button {
margin-top: 25px;
display: -webkit-box;
display: -webkit-flex;
display: flex;
}

.am-button {
display: block;
margin: 0 10px;
padding: 0 10px;
height: 42px;
text-align: center;
font-size: 18px;
line-height: 42px;
border-radius: 4px;
outline: 0;
-webkit-appearance: none;
-webkit-box-flex: 1;
-webkit-flex: 1;
flex: 1;
width: 50%;
}

.am-button[am-mode~=white] {
border: 1px solid #DDD;
color: #666;
background-color: #FFF;
}

.am-button[am-mode~=white]:active {
border-color: #D8D8D8;
background-color: #F8F8F8;
}

.am-button[am-mode~=blue] {
border: 1px solid #28F;
color: #FFF;
background-color: #39F;
}

.am-button[am-mode~=blue]:active {
border-color: #17F;
background-color: #28F;
}

.am-button[am-mode~=light], .am-button[am-mode~=light]:active {
border: none;
color: #39F;
background-color: #FFF;
}

@media screen and (min-device-width: 375px) {
.am-page-result .am-page-result-pic {
```

```
        width: 160px;
        height: 160px;
        margin: 45px auto;
        }

        .am-page-result-button {
        margin-top: 30px;
        }

        .am-button {
        margin: 0 20px;
        height: 44px;
        line-height: 44px;
        }
        }

        @media screen and (min-device-width: 414px) {
        .am-page-result .am-page-result-pic {
        width: 180px;
        height: 180px;
        margin: 50px auto;
        }

        .am-button {
        margin: 0 24px;
        height: 50px;
        line-height: 50px;
        }
        }
    </style>
<body>
<div class="am-page-result">
    <div class="am-page-result-pic">
        <img
src="
///90wNwQvOzn39X3x5D///8UvO321KL///8Guu4SvO73x5H///6hyun3x5Clyumlyummyun+1aD////////wu4SjxuKlyuk+wOpZYo
+7/1qFYYYnwuoOlyukAuu3/1p8AsewAufr/25ZMV4U+Snt62PVKxuZ5wcT/56b6ypP/u3TZz6qCobRscZCAgpmll5TIt5qg5vn////t
55P/C5//Z8f/J0OOqv8Kmuo/+AAAAIHRSTlMAPkjv/X99EPYjepuk92HIXqPcQE3Ye7PQsMfRX5aRu7xgWQUAAA+TSURBVHhe7Jrdii
EZXCpRG8MI7fZP6Mfv+b7VLIDAzziRD1NZKPDeB9FUfvvrU7v6zCpvNZrNxhwnW+pRzjHAlxpyTtzaYw72ViGBTBrwDZG/DG0g5jE2A
SNcfrpsLYKhMPgWyNe8FYBA/4JODDS4XEhQTYRgqvkhHjATsA3rxCMDJ2IyuPyOEBuwL+UDwlOABvtoxPKEzI8UHGFmIBhwJW05xEHE
uYk4SkkFRMTAE8Cwo7Gl4Ds1tDTIBZPZ90lxiWcQHKL1kbEKcQlC8QATgIWLJCAEwkr29g+LE7Grmpj+wi4AGFFG9uHwUVYYr09cBmC
cxFaICLsQ3XQdqUlEKXKjlHYpSfESS0WEmeWK8H+kkO7l1rTIYJYvMEvRXKeAT1JuLmohhA3EmTY8PgcJy08wNwXE6xsVYpY7sOgcl9
5iPqGxVIZZHsOhbXVy3bNQ09YfT1KPC8guYlLXp0W9Uvv1L2dkl9QkH8+2n37h4teG4bcypMKuLR3tzFP5py0XCetvDYY9ZYaqudZkW
eCNLdO+43LUHBhrSaFbl7VdHW1GAHHfU4yCdIzcnFd9FBlY7qso4yjU836Ufoka3niRpmBal8BMfpQKP+JWStQ8m5NuMA6u5Q8lDdwQ
MEWYEJ1DA2HaHmmbs8IBxctb7A9joCqI1wTXnmTCkvHWcE882VTO1KHQ0mXnrCqcNHzecMJNkTR1y/Da1SY3lkHccOoTF9ph9eooCYd
DVH1pOtaGMKEqHYNrtKAuHeNrVL+O+TWqv0rrGmVSrGP+/osul79XLhdSraP9qPKPezNYbRgGgmirQkUp+JBbcjHMcfZo6VC5lv7/r0
w2ByNkIeeP3CQ/PstZIS5kCSPmFRJzK8txbqf5Wx1IiRUjiD5IiiCU/7y9cmpU28gQR4gqKMJa+tfWhYVZUgyPHCxdXRkLpWTwdG+rQ
i4CYiIX8aap30FXadHISoITK6XjqGLWu3IoI1EKOJe2H62bFGxyTEWkZvYpbqG/0aKQruwRkoWtSNflVHCoL7cHt+KhoEMOjDb2MjCm
3ELG0Wgwe0zLYmEUR/eUFTqfOwwLVkILTt8eD8J1IxGNzE9CYQe9xzDNF1yMys4Y3Z8vDUvFeallhXDcRkaNU5nIXNdRyBs+zg00LG6
gYDouQ08dmYTxz+PUU4fgUZy53UOvowgexps89tDrsB6XNwM6TMalvw79MuZcl7jY1wF/tgHXZ/mwr2N03mG00zHo9w60YjS7jKX5gu
kA0wtttKJe1xUJqpsPt4mAsEFeQNFg0nDY6DCMAUu5xwn82fpg7v95WcSCKb/40cZNC7lZ9uK32we5qxmBCqLhAsrvpzff/VmsTsy51
qCXsqRYpO+5CfzswwBcXL4er5LYyj3FUQVR4E+31VvvZFIpKGxvRuM/8gzkkvbR7upqPw8I8WITBjmRQL9lXx6iciktc/nlar1dPybt
42A+gR08+1/8Y3xn3tpUjIpVC8SSV6VkkgnDVF9DzKGDIOTBcMpMKAMkOnYPOzbLEnBMmYkU5JXhXATSYp9pv8AY9z5kgHm1bbBG5Dl
bRlIEmc6GIcL2LiJClEwlQwcqjk8WDeB+QQPlbsx5a9+TFK+KRiP8SKT83EbUu+KHtFD9NAHZNTRAZQA4p1u/CHANxN9A1repFpuGgY
FDPmkSclYii+EGiU4ztX45TiYNqcUj7PYl13IIOlH/ChLeqFiHyDFEj0K+tNiJnTVWWRVGUZfXniURRqmkYxfl8Dg2NtP8Drcf0HBCy
QDlMtQuwzkwhLGknDZhdFNMpjZoQ6HpyCpjEopdtzPh7GPtmpTAyNtmw0pkdEFCKCNrf4FFFDY4jSM4/lyKct/EwsGh7pz6xqBS0rP0
gbnlHA9XfyaXG/iztQaJ5I2DewKShzQCCSOXYzaMDYGQKWAvw/fw+v2QcY+jqNKJI2qTzZMqQCtcTAjg8OUyuBy0cNl1IAEA2lgQKnB
KAn78BRxHXS09FV5lh8myStHAYZ3DgYMhIwoH+aWcEoVy0//ZsfXXw8C3LK/Y5XDgpbcYpxEFcOFA6bpr5Xj453A4HJ3VAhQ2Ix5Im7
GQdGCwLGfkvHXipsSgcW+fVhZLbtXqpT5Pnr6aBjhbhBEJorShFNJSaFx8OAPDgGO385pmi4ZcBZmaHEgSxo7N0fmAvjqG6Xzx+UThY
Dz+mg0Sm2ucU59eCgXhzDFX5BzcwQHdWBXWBQ9w6ISIC2nUNHK015t2tNlhvWzGOrVtCPwkwW6hq1WHdagMuL/d8UKO1ywbHEXfPCzC
DAOXtrSaHvDRxMe4Ft3NTjdQ+1+G+Ne3ex5cb96s+qg91UpNLPFceFBXFsa0C5X6l4CuGlAZi+LUO3RK3Su8Ebo2llA4iCIeBEH5Vtn
dzS1u5hoS68Vk2QyoFgr0z9r2Rqt6qVlOqrU4tpYjrzPldk2tXDsgs8fn9SKsSRh9Z5ZwyNZCIcBL2VEWh88r/jtwRYI6XMqVubnmw0
0UYhpPJ5F5ysPXkyISvrdYfuc5HbuNT3pnHod0oQZdKpzu95knooQfk7/54oOtfHlGNI2bowEE5T00E3lP+gQa43IerPk22+Jd68+tx
```

```
8w/89Uf09Uf75f2dtUijz7HIOk4REGbIO6NO49TPwEZz/AQVk/Lnd/2Tnb5ZWEOXMU/Ya3wPIOk4xB5oavQTSNZkAZx47WicscvC2t7
Ix9Xl2gE0nmKQZByouFBKu641sAEHtrQ2RAHra8RQKZ4r+ZKKAynEJTQeUK1XEN2ypwUQMPCRQDjg8XSDpOMo4IUUKVGB1MOadlb3U4
ubA96wEjKcbJB1HMySMcOjLpfAnHueu336qn7WP67hD0ni+QdJx1MY4igU5/4OJEjWF69WT+EDJl2QcyhhDHmSIqpo4AgVkPEnxEYJB
IM2mg7BFFkgtRgF4Mh/hYyZckHLW5ydG7laBdirki+Z7yaDQC+iiDJOBoDHgE5KdMFRiJT8NnwgGDbMaRL55ed58ulEoWGjnhps+GAw
KMw4LG1I1XLAwIG7sChoZeEWEmJlyIyDxtPSnwOffmyLZFCw5bu1NViVuVFSgAHIymz4nHPEy488et8AUmyMp4KwUivdz0Aic9vFW5C
sMilLHHJ3/Nr3365vBkgAhJLKEWVggE8KRy1Y6Fhm1s4jZ3bp3wyIBDb7usgFpOOxAuWCgXaHFuomZRMRABncqtnQqhaPKsgdTWV8SQ
t6TmJkILGJUoXU1SWul4AuEijfT0KmjYUCjbH3Zhcjj5BV4Vy2vyDGvaljW2ZJ+I8vePdKf3RC/DZHrEyltOBaWvYcBX+dDPW6IFYS6
AHOwKy87aAIwWJ+9RR9aSKvRoRDcMM6nMWksbYznSs3QSWtXSX8xUwSZNGYaZtKRILFw6Wz0O+s5pVdTLxjuvC30KSyuGqWP3SMKCiY
q30TROoqJRh2IRl8AawYBWCtWtMyBho0XXDMFb2VNpAIr1Q5ojzQVnhFyvStjHh/Tai5YxLG4axUlLF1auyyXKYVjjFyq4NC+eq4zs5
B+PRgB9RVzxs4CrjhzZboPgFiYA67woC/0YJF755+6sMwkR7IcoKvI2KFokDLgQYzc3itF36hyqbLGabm2FBlU4g4RArjsmL9LIoXEB
EQKz7IyizzWGGrB4orQUHzKCpjgo+3Hc3ryMEFJds9HkUzv9mgvV/crERfrPmANhqEiwSBIpeuyu0VAITzBAAa/qgJzxGW7m0XGM4GE
xcWCURStqgyWm6+ylHx2JBIwGw5mG2IdDBLJ2IdJm/XU8nSc5AsowGN40RBaAYQOM5tXtxKQCDqDgTuMIM1Cir84nqLPLOCwyEMMsCh
SBnmi4zryZBNYN9vY0xiKvrU2g0OcsVdQKeN+BgHyiosv/PHzYlVpQUHHQkaUTSB2aZhFhRueChw2o+WVXZqHe6vwq1JWcUUjDRzjOB
xFJ1sXlSrJsqiFnxU0kGxKYLwyp6sCwJKM6HI/yPnClIchoGYMRg7i8E+5AG56P9f3B42sFjMTN1bFUEPaXPJwEgajZv0TagU9+D+BJ
r0koB7nef784aRKfAGmsVZ5y7WjJjEM533LYRONJIZM/hsbnrQkMRRqgaBH8P/mnMTQtt0o1HsF1nMDD+uVTl1AZozqpdsrtUVDCpya
GecE4GhcDV1c0HTGwZMc3plqvLC6UjBfcsVbHkrJBj9V1FZoh3SswisBTv6jMVuOBDZlZL7OermwMKfIyO6gj2G7YuUfX05X9dYKuzE
CEOXV1BFA7qJz8Vm54cUJ5mJ/xakCHVpo7O2xWQ9TQ9GapyDoattT30qCNf21CmjhIf+IETgmRVS8oyC9q70FUT7BVYBt3WHFWHXqgD
b1kmvyMeCsL6AfoZe25tZMdaqQJE6J1XBYVH9DL1YxMDAAxz6IN4ke/ok6vDnNu9XTer4LAmDpuvojrLGgBx1BCYjmPWzokNfgHUFC7
Tp8mAdvValjNxGUjkmHTZ5RNaBHHcfNkhyBsm/nHL1VaerwN9X8/RCnDg7F3HrpZx1b/5Scj12/4YL8sfyJSFYIEF7OFpdH4adiQD9e
oUgcfcAG8uvy2cz6rDcMwHN6l2IZCBsLSxQ47SLfkFfz+j7V4hqVbmqitwU0g3zUnf4Sf/hw0Ji74CSKKERFDAHCHMmQAAmIkskp+Kp
DrZT8xu3F7NGAdZgvdc4EHJCS1NWGE2E3FPXhxkDfyPNMpmRmh9uuUn8YQB3ivFBJxFLHRsaNAZE7+CjwjmPSqi5w3srEAVIktLQ+KX
GMiCygpoZsjQkv15k4Co8uoa5Saxy14A8fLOCq8FmOjxrpFGJCkWI5XpiKx2skkbl0XqSVkPNaqtnjfFOQZWNFaH8zZKhXgc1rDBBE2
qswVMtoXFwAiTdY67REXQhJfZISul11YWl46JCLrIz5YjlT34e1x9zr0dMo82uX7179Q6QiST0pJppKCWWAm3WI6MGxXKQ/r8MTYRlX
xZby/dtZaKUu/RfOlmpFSWNQRv4yyh8AYB/D5een7/tp1nc3cDinr1yhvk3S5AgoBFAcH0VME/XAp9JnrLf3E9AERwwRk3HHWXycnJy
A9otAf5V/kAAAAASUVORK5CYII="/>
    </div>
    <p>这是自定义错误页面</p>

</div>
</body>
<script>




</script>
</html>
```

设置和原生 View 一起滚动

```
//在启动参数里添加
// scrollForward： 0: SELF_ONLY, 1: SELF_FIRST, 2: PARENT_FIRST, 3 : PARALLEL
// scrollBackward: 0: SELF_ONLY, 1: SELF_FIRST, 2: PARENT_FIRST, 3 : PARALLEL
param.set("scrollForward", 2)
param.set("scrollBackward", 1)
```

支持限制 Web 组件宽度

- 通过启动参数控制。

```
params.set('webWidth', xxx)。 // 参数支持string ('100%'百分比方式) 或者number (例如800表示800px)
```

- 动态控制。

```
page.setWebWidth(webWidth: string | number)
```

鸿蒙 Web 行为定制

H5WebClientProvider

可以通过实现 `H5WebClientProvider` 修改鸿蒙 H5 容器 Web 的一些默认行为 API 。

```
HRiver.setProvider(H5WebClientProvider.name, new H5WebClientProviderImpl())
```

可定制的 API 如下：

```
export class H5WebClientProvider {

  // 页面http错误回调
  onHttpErrorReceive(page: Page | undefined, request: WebResourceRequest | undefined, response: WebReso
urceResponse | undefined) {
  }

  // 页面下载回调
  onDownloadStart(page: Page | undefined, url: string | undefined, userAgent: string | undefined, conte
ntDisposition: string | undefined,
    mimetype: string | undefined, contentLength: number | undefined) {
  }

  // 页面ssl错误回调
  onSslErrorEventReceive(page: Page | undefined, handler: SslErrorHandler, error: SslError) {

  }

  // 页面错误回调
  onErrorReceive(page: Page | undefined, request: WebResourceRequest | undefined, error:
WebResourceError | undefined): boolean {
    return false
  }

  // 全屏回调
  onFullScreenEnter(page: Page | undefined, handler: FullScreenExitHandler) {

  }

  // 退出全屏回调
  onFullScreenExit(page: Page | undefined) {

  }

  // web权限申请回调
  onPermissionRequest(page: Page | undefined, request: PermissionRequest | undefined) {

  }

  // web screencapturerequest回调
  onScreenCaptureRequest(page: Page | undefined, handler: ScreenCaptureHandler | undefined) {

  }

  onPageBegin(page: Page | undefined, url: string | undefined) {

  }

  onAppear(page: Page | undefined) {

  }

  // web scroll回调
  onScroll(x: number, y: number, page: Page | undefined) {

  }

  getWindow(context: Context, page: Page | undefined): Promise<window.Window> | undefined {
    return undefined
  }

  // web 长按菜单
```

```
  onContextMenuShow(param: WebContextMenuParam | undefined, result: WebContextMenuResult | undefined) {
    return false
  }

  // web 文件选择回调
  onShowFileSelector(fileSelector: FileSelectorParam, result: FileSelectorResult, page: Page | undefine
d) {
    return false
  }

  // web 定位相关回调
  onGeolocationShow(origin: string | undefined, geoLocation: JsGeolocation | undefined, page: Page | un
defined) {

  }

  // web 定位相关回调
  onGeolocationHide(page: Page | undefined) {

  }
}
```

H5MixModeSettingProvider

可以通过 `H5MixModeSettingProvider` 设置 Web 支持 HTTP/HTTPS 的 `MixedMode` ，具体使用如下：

```
HRiver.setProvider(H5MixModeSettingProvider.name, new H5MixModeSettingProviderImpl()) // 业务实现
H5MixModeSettingProviderImpl
```

```
export class H5MixModeSettingProviderImpl extends H5MixModeSettingProvider {
  mixMode(page: Page | undefined): MixedMode {
      // 根据业务实际情况返回对应的MixedMode
    return MixedMode.Compatible
  }
}
```

> **重要**
>
> 鸿蒙系统不支持在 HTTPS 页面加载地址为 HTTP 的 IP 链接，要么加载 HTTPS 的 IP 的链接，要么加载 HTTP 的非 IP 链接。

页面路由支持 Navigation

默认页面路由使用 router 方式，鸿蒙 router 方式不支持关闭栈中某个页面，只能一级级回退。离线包支持 Navigation 模式：

1. 支持关闭栈中页面
2. 支持分栏模式

全局 Navigation 模式

1. HRiver 初始化完成后，在启动离线包之前，需设置 `H5RouterNavStackProvider` 。

```
HRiver.setProvider(H5RouterNavStackProvider.name, new NavStackProvider(this.pageInfos))
```

```
import { H5RouterNavStackProvider } from '@mpaas/hriver';

export class NavStackProvider extends H5RouterNavStackProvider {
  navStack: NavPathStack // 全局的navPathStack栈

  constructor(navStack: NavPathStack) {
    super();
    this.navStack = navStack;
  }

  getNavPathStack(): NavPathStack {
    return this.navStack
  }
}
```

2. 在 `navDestination` 中配置 `builder`，`builder` 中加载 mPaaS 全局 Builder。

```
import {HRBuilder, RouterUtils, HRiver, H5RouterNavStackProvider} from '@mpaas/hriver'

let mPaaSHRiverBuilder: WrappedBuilder<[string, ESObject]> = wrapBuilder(HRBuilder);

@Builder
PagesMap(name: string, params: ESObject) {
  if (RouterUtils.isMPHRiverPage(name)) {
    mPaaSHRiverBuilder.builder(name, params)
  } else if (name == 'xxx') {
    // 其他业务的页面
  }
}

build() {
    Navigation(this.pageInfos) {
      Row() {
        Column() {
          // 业务页面
          MainPage()
        }
        .width('100%')
      }
    }.navDestination(this.PagesMap)
  }
```

离线包独立使用 Navigation 模式

1. `HRiver.startApp` / `HRiver.startUrl` 的第三个参数传入 NavPathStack 即可。
2. 参考 全局 Navigation 模式 中的步骤 2。

1.5.2. HarmonyOS NEXT 支持的功能说明

HarmonyOS NEXT 版本支持的 JSAPI

启动参数

HarmonyOS NEXT 启动参数仅支持以下 key。

名称	缩写	类型	说明	默认值	pushWindow 是否可用
url	-	string	起始 URL	`""`	是

defaultTitle	dt	string	默认标题，在页面第一次加载之前显示在标题栏上。	`""`	是
showTitleBar	sl	string	true/false，是否显示标题栏。	`true`	是
readTitle	rt	string	YES/NO，是否读取网页标题显示在 titleBar 上。	`"YES"`	是
backgroundColor	bc	int	设置背景颜色（十进制，例如：bc=16775138）。	`#FFFFFF`	-
showOptionMenu	so	bool	YES/NO，是否显示右上角的"..."按钮。	对于 H5App 为 `NO`，对于非 H5App 为 `YES`。	-
showBackButton	--	string	YES/NO，是否显示左上角返回按钮。	`YES`	是
centerTitle	--	string	YES/NO，标题是否居中。	`NO`	是
showBackButton	--	String	YES/NO，是否显示返回键。	`YES`	是
showBottomLine	--	String	YES/NO，是否显示 Titlebar 底部分割线。	`YES`	是
embedPage	--	String	YES/NO，是否是内嵌页面。	`NO`	是
buttonColor	--	String	Titlebar 按钮颜色值，默认是 0xFF333333。	`0xFF333333`	是
scrollForward	--	Number	WebView 滚动冲突时的模式设置。 0：NestedScrollMode.SELF_ONLY 1：NestedScrollMode.SELF_FIRST 2：NestedScrollMode.PARENT_FIRST 3：NestedScrollMode.PARALLEL	`0`	是

scrollBackward	--	Number	WebView 滚动冲突时的模式设置。 0：NestedScrollMode.SELF_ONLY 1：NestedScrollMode.SELF_FIRST 2：NestedScrollMode.PARENT_FIRST 3：NestedScrollMode.PARALLEL	`0`	是
webBackground	--	String	Web 背景色。 `transparent` 表示透明； 颜色值表示具体的颜色，比如：`0xFFFFFF` 表示白色。	`0xFFFFFF`	是
bottomSafe	--	String	YES/NO，是否保留底部安全区域，避免 Web 内容和底部系统导航冲突。	`NO`	是
forceStatusBar	--	String	YES/NO，是否强制显示顶部状态栏。	`NO`	是
showOptionMenu	--	String	YES/NO，是否显示Titlebar 的菜单。	`YES`	是

事件扩展

初始化操作

```
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
```

点击标题栏

```
document.addEventListener('titleClick', function(e) {
  alert('title clicked')
}, false);
```

点击副标题

```
document.addEventListener('subtitleClick', function (e) {
  alert('subtitle clicked')
}, false);
```

页面压后台

```
document.addEventListener('pause', function(e) {
  alert("pause");
}, false);
```

页面恢复运行

```
document.addEventListener('resume', function(e) {
  console.log("resumed");
}, false);
```

点击右上角按钮

```
document.addEventListener('optionMenu', function (e) {
  alert("option menu clicked");
}, false);
```

回退

```
// 首先屏蔽默认行为，然后调用页面跳转 API 进行手动控制
document.addEventListener('back', function(e) {
  e.preventDefault();
  console.log('do something...')
  AlipayJSBridge.call('popWindow');
}, false);
```

添加通知

```
AlipayJSBridge.call('addNotifyListener', {
  name:'fortest'
}, function (result) {
  console.log(result);
});
```

移除通知

```
AlipayJSBridge.call('removeNotifyListener', {
  name: 'fortest'
}, function (result) {
  console.log(result);
});
```

分发消息

```
AlipayJSBridge.call('postNotification', {
  name:'fortest',
  data:{}
}, function (result) {
  console.log(result);
});
```

页面上下文

打开新页面

```
// 打开淘宝首页，自动读取 title，并且去除右边菜单按钮
AlipayJSBridge.call('pushWindow', {
  url: 'https://m.taobao.com/',    // 要打开页面的 URL
  // 打开页面的配置参数
  param: {
    readTitle: true,               //自动读取 title
    showOptionMenu: false          // 隐藏右边菜单
  },
  // 用于给新开的页面传递参数，可选
  // 在新开的页面使用 AlipayJSBridge.startupParams 可以获取到 passData
  passData: {
    key1: "key1Value",
    key2: "key2Value"
  }
});
```

关闭当前页面

```
AlipayJSBridge.call('popWindow');
```

启动其他应用

```
AlipayJSBridge.call('startApp', {
  appId: '90000000',
  param: {
    url: '/index.html'
  }
}, function(result) {
  // noop
});

// 注意，如果要打开多个 App 实例：
// 请将 appClearTop 和 startMultApp 都放在 param 里
AlipayJSBridge.call('startApp', {
  appId: '90000000',
  param: {
    url: location.href,
    appClearTop: false,
    startMultApp: 'YES' // 注意这个值是 YES，而不是 bool 类型
  }
}, function(result) {
  // noop
});
```

退出当前应用

```
AlipayJSBridge.call('exitApp');
```

界面

警告框

```
AlipayJSBridge.call('alert', {
  title: '亲',
  message: '你好',
  button: '确定'
}, function(e) {
  alert(JSON.stringify(e));
});
```

确认框

```
AlipayJSBridge.call('confirm', {
  title: '亲',
  message: '确定要退出吗？',
  okButton: '是',
  cancelButton: '否'
}, function(e) {
  alert(JSON.stringify(e));
});
```

弱提示

```
AlipayJSBridge.call('toast', {
  content: '操作成功',
  type: 'success',
  duration: 2000
}, function() {
  alert("toast消失后执行");
});

// 可以通过 hideToast 接口隐藏已经弹出的 toast

AlipayJSBridge.call('hideToast', {}, function() {
});
```

选择列表

```
AlipayJSBridge.call('actionSheet', {
  'title': '标题',
  'btns': ['第一个按钮', '第二个按钮', '第三个按钮'],
  'cancelBtn': '取消',
  'destructiveBtnIndex': 2
}, function(data) {
  switch (data.index) { // index 标示用户点击的按钮在 actionSheet 中的位置，从 0 开始
    case 0:
      alert('第一个按钮');
      break;
    case 1:
      alert('第二个按钮');
      break;
    case 2:
      alert('第三个按钮');
      break;
    case 3:
      alert('取消按钮');
      break;
  }
});
```

设置标题

```
AlipayJSBridge.call('setTitle', {
  title: '标题',
});
```

设置导航栏背景色

```
AlipayJSBridge.call("setTitleColor", {
  color: 16775138
});
```

显示右上角按钮

```
AlipayJSBridge.call('showOptionMenu');
```

设置右上角按钮

```
AlipayJSBridge.call('setOptionMenu', {
  xx // 参考入参
});
```

入参：

属性	类型	描述	必填	默认值
title	string	右按钮文字。	Y	“”
icon	string	右按钮图标 URL，base64（since 9.0）。 8.3 及以前版本：iOS 40x40（周边不留白），Android 50x50（四边各透明留白 5px）。 8.4 及以后版本：两个平台统一使用 40x40（周边不留白）。	Y	“”
reset	bool	重置为系统默认，当 reset=true 时，忽略其他参数。	Y	false
color	string	文字颜色值。	N	“#FFFFFFFF”
override	bool	在需要设置多个 option 的情况下，是否保留默认的 optionMenu。	N	false
menus	array	设置多个按钮。	N	[ ]
preventDefault	bool	是否阻止默认的分享功能（默认是弹分享框）preventDefault=true 时，会阻止默认的分享。	N	[ ]

icontype	string	根据图片类型加载容器预置图片，与 title、icon 三选一。 重要 具体类型包含 user（账户）、filter（筛选器）、search（查找）、add（添加）、settings（设置）、scan（扫一扫）、info（信息）、help（帮助）、locate（定位）、more（更多）、mail（邮箱）。	N	“”

隐藏右上角按钮

```
AlipayJSBridge.call('hideOptionMenu');
```

显示加载中

```
AlipayJSBridge.call('showLoading', {
  text: '加载中',
});
```

隐藏加载中

```
AlipayJSBridge.call('hideLoading');
```

显示标题栏加载中

```
AlipayJSBridge.call('showTitleLoading');
```

隐藏标题栏加载中

```
AlipayJSBridge.call('hideTitleLoading');
```

设置导航栏底部细线颜色

```
AlipayJSBridge.call("setBarBottomLineColor", {
  color: 16711688
});
```

设置 pop 菜单

```
AlipayJSBridge.call('showPopMenu');
AlipayJSBridge.call('setToolbarMenu');
```

隐藏/显示返回按钮

```
AlipayJSBridge.call('showBackButton');
AlipayJSBridge.call('hideBackButton');
```

工具类

获取容器的启动参数

```
AlipayJSBridge.call('getStartupParams', {
  key: ['url', 'xxx'] // 可选，根据 key 值过滤返回结果，不填返回全部
}, function(result) {
  console.log(result);
});
```

RPC 调用

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

设置 AP 数据

```
AlipayJSBridge.call('setAPDataStorage', {
  type: "common",
  business: "customBusinessKey",
  key: "customKey",
  value: "customValue"
}, function(result) {
  alert(JSON.stringify(result));
});
```

获取 AP 数据

```
AlipayJSBridge.call('getAPDataStorage', {
  type, business, key
});
```

移除 AP 数据

```
AlipayJSBridge.call('removeAPDataStorage', {
  type: "common",
  business: "customBusinessKey",
  key: "customKey",
}, function(result) {
  alert(JSON.stringify(result));
});
```

不支持的 JSAPI 列表

Native 功能

目前不支持扫码解析。

```
AlipayJSBridge.call('scan', {
  type: 'bar',
  actionType: 'scan'
}, function(result) {
  alert(JSON.stringify(result));
});
```

替代方案

由业务实现，注册对应的 JSAPI，并在 API 里调用 mPaaS 扫码组件或者系统接口。

工具类

- 目前不支持截屏。

```
AlipayJSBridge.call('snapshot', function(result) {
 console.log(result.success);
});
```

替代方案

由业务实现，注册对应 JSAPI，并在 API 里调用系统接口。

```
export class H5CustomPlugin extends H5SimplePlugin {
  static pageInfos?: NavPathStack

  onPrepare(filter: H5EventFilter): void {
    filter.addAction('snapshot')
  }

  handleEvent(event: H5Event, context: H5BridgeContext): Boolean {
    if ('snapshot' == event.action) {
      let page = event.target as Page
      componentSnapshot.get(page.webComponentId, (error: Error, pixmap: image.PixelMap) => {
        if (error) {
          console.log("error: " + JSON.stringify(error))
          return;
        }

        // pixmap 截图结果
      })
      return true
    }
    return super.handleEvent(event, context);
  }
}
```

- 目前不支持上报埋点。

```
AlipayJSBridge.call('remoteLog', {
 bizType: "Nebula", // 业务类型
 logLevel: 1, // 1 - high, 2 - medium, 3 - low
 actionId: "event", // 埋点类型，固定为 "event"
 seedId: "Login", // 埋点唯一标识
 param1: "",
 param2: "",
 param3: "",
 param4: {key1:"value1",key2:"value2"}, // 自定义参数
});
```

1.6. 内置 JSAPI

1.6.1. 启动参数

H5 容器运行时的外观和行为受一组参数控制，可在启动一个新实例或者 pushWindow 时指定，例如：

```
mpaas://platformapi/startapp?
appId=20000067&url=http%3A%2F%2Fm.taobao.com&showOptionMenu=NO&startMultApp=YES
```

从前端打开一个新的 H5 实例

第一个 URL 也可以带一个魔法参数 `__webview_options__`，其内容将被容器取出并传给容器本身。

```
?__webview_options__=showOptionMenu%3DNO&startMultApp%3DYES
urlencode('showOptionMenu=NO&startMultApp=YES') => showOptionMenu%3DNO&startMultApp%3DYES
```

从客户端打开一个新的 H5 实例

启动参数的设置方法如下。

```
Bundle bundle = new Bundle();
bundle.putString("showOptionMenu", "NO");
MPNebula.startUrl(url,bundle);
```

客户端透传给前端的启动参数，前端可以直接通过 `AlipayJSBridge.startupParams` **或** `jsapi:getStartupParams` **获取。**

名称	缩写	类型	说明	默认值	pushWindow 可用
url		String	起始 URL	“”	Y
defaultTitle	dt	String	默认标题，在页面第一次加载之前显示在标题栏上。	“”	Y
showLoading	sl	String	YES/NO，是否在页面加载前显示全局菊花。	“NO”	Y
readTitle	rt	String	YES/NO，是否读取网页标题显示在 titleBar 上。	“YES”	Y
bizScenario	bz	String	业务场景来源，这个值会记录到每一个埋点中，可以用来区分不同来源。	“”	-
backBehavior	bb	String	back，pop，auto 指定后退按钮行为。 **back**：如存在浏览器历史则后退到上一页，否则关闭当前 WebView。 **pop**：直接关闭当前窗口。 **auto**：在 iOS 上相当于 pop；在 Android 上，toolbar 可见时相当于 back，toolbar 不可见时相当于 pop。	非 H5App 的通用浏览器模式（appId 为 20000067）为 back，H5App（用 startApp 来启动）为 pop	-
pullRefresh	pr	String	YES/NO，是否支持下拉刷新。只有本地文件允许设置为 YES。	“NO”	Y
showProgress	sp	bool	YES/NO，是否显示加载的进度条。	“NO”	-
canPullDown	pd	String	YES/NO，页面是否支持下拉（显示出黑色背景或者域名）。只有本地文件允许设置为 NO。	“YES”	YES
showDomain	sd	bool	YES/NO，页面下拉时是否显示域名。只有本地文件允许设置为 NO，离线包强制设置为 NO，不允许显示。	“YES”	-

backgroundColor	bc	int	设置背景颜色（十进制，例如：bc=16775138）。	“”	-
showOptionMenu	so	bool	YES/NO，是否显示右上角的“...”按钮。	对于 H5App 为 NO 对于非 H5App 为 YES	
showTitleLoading	tl	bool	YES/NO，是否在 TitleBar 的标题左边显示小菊花。）	NO	Y
enableScrollBar	es	bool	YES/NO，是否使用 WebView 的滚动条，包括垂直和水平。只对 Android 有效 。	默认为 ”YES”	-

1.6.2. 事件扩展

1.6.2.1. 初始化操作

当执行 `window.onload` 后，容器会执行初始化操作，产生全局变量 `AlipayJSBridge` ， 然后触发 JS Bridge 初始化完毕（ `AlipayJSBridgeReady` ）事件。

> **重要**
> - `AlipayJSBridge` 注入是一个异步过程，一定要在监听该事件后调用 `AlipayJSBridgeReady` 。
> - 执行初始化操作请务必使用 `ready` 方法，否则可能会导致 H5 获取 AlipayJSBridge 失败。

AlipayJSBridgeReady 使用方法

```
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
```

代码示例

以下代码示例为 `bridge` 入口的标准写法：

```
<h1>bridge 使用方法</h1>

<script>
function ready(callback) {
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}

ready(function() {
  alert('bridge ready');
});
</script>
```

1.6.2.2. 单击标题栏

单击导航栏标题（ `titleClick` ）触发回调。

titleClick 接口使用方法

```
document.addEventListener('titleClick', function(e) {
  alert('title clicked')
}, false);
```

代码示例

以下代码示例为基本功能演示：

```
<h1>请单击标题查看效果</h1>

<script>
document.addEventListener('titleClick', function(e) {
  alert('title clicked')
}, false);
</script>
```

1.6.2.3. 单击副标题栏

单击导航栏副标题（ `subtitleClick` ）触发回调。

subtitleClick 接口使用方法

```
document.addEventListener('subtitleClick', function (e) {
  alert('subtitle clicked')
}, false);
```

代码示例

以下代码示例为基本功能演示：

```
<h1>请单击副标题查看效果</h1>
<script>
document.addEventListener('subtitleClick', function(e) {
  alert('subtitle clicked')
}, false);

function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  AlipayJSBridge.call('setTitle', {
    title: '标题',
    subtitle: '副标题'
  });
});
</script>
```

1.6.2.4. 页面压后台

当一个 WebView 界面不可见时，例如被压入后台、锁屏、或 `pushwindow` 到下一页面，会触发页面压后台（ `pause` ）事件 。

> **说明**
>
> - 对于 10.0.15 以上版本的客户端，容器新增了 `pagePause` 和 `appP` ause 事件，用于区分业务中具体哪种情况触发 `pause` 。页面压后台（ `pause` ）= 客户端压后台不可见（ `appPause` ）+ 窗口压栈底不可见（ `pagePause` ）。
> - 由于 Android 系统无法区分原生的 `resume` 和 `pause` 事件是由于压后台导致的还是由于页面切换导致的，所以 `pageResume` 和 `pagePause` 事件是通过 JSAPI 调用记录去回调的，仅适用于同一个 `session` 内 Window 之间的互相切换，对于 `startApp` 和其他客户端直接切换页面方式不生效，例如 `chooseImage` 。

pause 接口使用方法

```
document.addEventListener('pause', function(e) {
  alert("pause");
}, false);
```

代码示例

以下示例为离开当前页面后弹出警告：

```
<h1>请单击按钮打开一个新窗口</h1>
<a href="javascript:void(0)" class="btn J_new_window">新窗口打开当前页面</a>
<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.J_new_window').addEventListener('click', function() {
    AlipayJSBridge.call('pushWindow', {
      url: location.pathname,
    });
  });

  document.addEventListener('pause', function(e) {
    alert('paused');
  }, false);
});
</script>
```

1.6.2.5. 页面恢复运行

当一个 WebView 界面重新回到栈顶时，例如从后台被唤起、锁屏界面恢复、从下一页面回退，会触发页面恢复运行（ `resume` ）事件。

如果这个界面是通过 `popWindow` 或 `popTo` 到达，且传递了 `data` 参数，则此页可以获取到这些参数。

如果在界面的 `resume` 之前先发生了 App 的 `resume` ，则 `event` 还会有一个 `resumeParams` ，包含 `app resume` 时接收到的参数。

> **说明**
>
> - 对于 10.0.15 以上版本的客户端，容器新增了 `pageResume` 和 `appResume` 事件，用于区分业务中具体哪种情况触发 `resume` 。页面恢复运行（ `resume` ）= 客户端从后台被唤起、锁屏界面恢复（ `appResume` ）+ 窗口从下个页面回退后恢复显示（ `pageResume` ）。
> - 由于 Android 原生的 `resume` 和 `pause` 事件不能区分是压后台导致还是页面切换导致，所以 `pageRe` sume 和 `pagePause` 事件是通过 JSAPI 调用记录回调的，仅适用于同一个 `session` 内 Window 之间的互相切换。对于 `startApp` 和其他客户端直接切换页面方式不生效，例如 `chooseImage` 。

resume 接口使用方法

```
document.addEventListener('resume', function(e) {
  console.log("resumed");
}, false);
```

代码示例

以下示例为带有数据的返回：

```
<h1>点击"打开新页面"，然后返回时，会带数据回这个页面</h1>
<a href="#" class="btn J_demo">打开新页面</a>
<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('a').addEventListener('click', function() {
    AlipayJSBridge.call('pushWindow', {
      url: location.pathname
    });
  });

  document.addEventListener('back', function(e) {
    e.preventDefault();

    AlipayJSBridge.call('popWindow', {
      data: {
        from: location.href,
        info: Date.now()
      }
    });
  });

  document.addEventListener('resume', function(event) {
    alert('页面回退时带过来的内容： ' + JSON.stringify(event.data));
  });
});
</script>
```

API

出参

事件处理函数的 event 参数有以下属性：

名称	类型	描述
data	Object	当前 App 实例内 `popTo` 或者 `popWindow` 传递过来的内容，无法跨 `appId` 传递数据。
resumeParams	Object	包含 `app resume` 时接收到的参数。

1.6.2.6. 单击右上角按钮

当您调用了 `setOptionMenu` 接口自定义了导航栏右上角按钮以后，单击按钮时触发该事件（ `optionMenu` ）。

optionMenu 接口使用方法

```
document.addEventListener('optionMenu', function (e) {
  alert("option menu clicked");
}, false);
```

代码示例

- 以下示例为基本功能演示：

```
<h1>请单击右上角菜单查看效果</h1>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
AlipayJSBridge.call('setOptionMenu', {
  title : '按钮',
  redDot : '5', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
  color : '#ffff6600', // 必须以 # 开始 ARGB 颜色值
});
document.addEventListener('optionMenu', function(e) {
  alert("optionMenu clicked!");
}, false);
});
</script>
```

- 多个 optionMenu 情况的事件：

```
<h1>请单击右上角每个菜单查看效果</h1>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
AlipayJSBridge.call('setOptionMenu', {
  // 显示的时候是从后往前显示的
  menus: [{
    icontype: 'scan',
    redDot: '-1', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
  }, {
    icontype: 'user',
    redDot: '10', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
  }],
  override: true //在需要设置多个 option 的情况下，是否保留默认的 optionMenu
});
// 必须强制调用一把刷新界面
AlipayJSBridge.call('showOptionMenu');

document.addEventListener('optionMenu', function(e) {
  alert(JSON.stringify(e.data));
}, false);
});
</script>
```

> **说明**
>
> 如果没有设置过 `optionMenu`，那么这个事件是无法被触发的，也就是说单击是默认的 `optionMenu`，是无法触发 `optionMenu` 事件的。

1.6.2.7. 回退

用户点击导航栏左上角返回按钮或者 Android 设备的物理返回键时，页面将会收到此事件（`back`）。

如果在事件的处理函数中调用了 `event.preventDefault()`，容器将忽略 `backBehaviour`，JS 需要负责回退或做其他操作。

> **重要**
>
> 对于 iOS 端侧滑返回，由于手势操作具有可中途撤销的特性，故无法触发 back 事件。 容器内部有保护机制，如果用户多次重复点击返回按钮依然未能退出当前页，将会触发强制退出，忽略 preventDefault。

back 接口的使用方法

```
// 首先屏蔽默认行为，然后调用页面跳转 API 进行手动控制
document.addEventListener('back', function(e) {
  e.preventDefault();
  console.log('do something...')
  AlipayJSBridge.call('popWindow');
}, false);
```

代码示例

点击回退跳转到指定页面

```
<h1>请点击左上角返回按钮</h1>
<p>点击后会跳回到淘宝页面</p>
<script>
// 注意：如果自定义了 back，并使用了 location.href 去跳到指定的地址，需要包装一个 setTimeout 以保证不会阻塞客户端线程。
document.addEventListener('back', function(e) {
e.preventDefault();
setTimeout(function() {
  location.href = "https://m.taobao.com"
}, 10);
}, false);
</script>
```

点击回退弹出确认框

```
<h1>请点击左上角返回按钮</h1>
<script>
document.addEventListener('back', function(e) {
e.preventDefault();

AlipayJSBridge.call('confirm', {
  title: '亲',
  message: '确定要退出吗',
  okButton: '确定',
  cancelButton: '算了'
}, function(e) {
  if (e.ok) {
    AlipayJSBridge.call('popWindow');
  }
});
}, false);
</script>
```

1.6.2.8. 添加通知

此接口用于添加 native 通知的监听。

addNotifyListener 接口的使用方法

```
AlipayJSBridge.call('addNotifyListener', {
  name:'fortest'
}, function (result) {
  console.log(result);
});
```

代码示例

以下示例为基本功能演示：

```
<h1>请点击下面的按钮来进行测试</h1>
<p>这里只测试在同一个页面内的情况，这个 API 可以在不同应用间通信</p>

<a href="#" class="btn start">开始监听</a>
<a href="#" class="btn stop">停止监听</a>
<a href="#" class="btn send">发通知</a>
<script>
function callback(e) {
  alert(JSON.stringify(e));
};

function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('.start').addEventListener('click', function() {
    AlipayJSBridge.call('addNotifyListener', {
      name:'NEBULANOTIFY_TEST_EVENT' // H5 发出的事件必须以 NEBULANOTIFY_ 开通进行监听
    }, callback);
  });

  document.querySelector('.stop').addEventListener('click', function() {
    AlipayJSBridge.call('removeNotifyListener', {
      name:'NEBULANOTIFY_TEST_EVENT' // H5 发出的事件必须以 NEBULANOTIFY_ 开通进行监听
    }, function(e) {
      alert(JSON.stringify(e));
    });
  });

  document.querySelector('.send').addEventListener('click', function() {
    AlipayJSBridge.call('postNotification', {
      name:'TEST_EVENT', // H5 发出的事件必须以 NEBULANOTIFY_ 开通进行监听
      data: {
        hello: 'world'
      }
    });
  });
});
</script>
```

API

> **重要**
>
> - 在监听事件时，如果这个事件是 H5 发出来的，那么需要加上 `NEBULANOTIFY_` 前缀。
> - 在调用 `addNotifyListener` 时，为了避免多次 add 报错，请先做一次 remove 操作。
> - 如果 keep 设置为 false，那么事件触发一次后自动反注册监听。
> - Android 在回调中返回的 data 只有一层 key/value 键值对，其中 value 如果在 postNotification 时是 Object，则会被 Android 进行 `JSON.stringify` 操作，使用时建议使用 `try{ JSON.parse }`。

```
AlipayJSBridge.call('addNotifyListener', {
  name, keep
}, fn)
```

入参

名称	类型	描述	必选	默认值
name	String	通知名称。	Y	“”
keep	String	是否在 ViewController 的生命周期内一直监听该通知。	N	“true”
fn	Function	回调函数。	N	-

出参

`result: {}` ：回调函数带入的参数，对应为事件的消息。

错误码

错误码	描述
4	无权限调用
12	在一个页面中多次监听同名通知

1.6.2.9. 移除通知

该接口用于移除 native 通知的监听。

removeNotifyListener 接口的使用方法

```
AlipayJSBridge.call('removeNotifyListener', {
  name: 'fortest'
}, function (result) {
  console.log(result);
});
```

代码示例

以下示例为基本功能演示：

```
<h1>请点击下面的按钮来进行测试</h1>
<p>这里只测试在同一个页面内的情况，这个 API 可以在不同应用间通信</p>

<a href="#" class="btn start">开始监听</a>
<a href="#" class="btn stop">停止监听</a>
<a href="#" class="btn send">发通知</a>
<script>
function callback(e){
  alert(JSON.stringify(e));
};

function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('.start').addEventListener('click', function() {
    AlipayJSBridge.call('addNotifyListener', {
      name:'NEBULANOTIFY_TEST_EVENT' // H5 发出的事件必须以 NEBULANOTIFY_ 开通进行监听
    }, callback);
  });

  document.querySelector('.stop').addEventListener('click', function() {
    AlipayJSBridge.call('removeNotifyListener', {
      name:'NEBULANOTIFY_TEST_EVENT' // H5 发出的事件必须以 NEBULANOTIFY_ 开通进行监听
    }, function(e) {
      alert(JSON.stringify(e));
    });
  });

  document.querySelector('.send').addEventListener('click', function() {
    AlipayJSBridge.call('postNotification', {
      name:'TEST_EVENT', // H5 发出的事件必须以 NEBULANOTIFY_ 开通进行监听
      data: {
        hello: 'world'
      }
    });
  });
});
</script>
```

API

> **重要**
> 移除操作无论是否注册，都会返回 `success：true` 。

```
AlipayJSBridge.call('removeNotifyListener', {
  name
}, fn)
```

入参

名称	类型	描述	必选	默认值

name	String	通知名称	Y	“”
fn	Function	回调函数	N	-

出参

`result: {success}`：回调函数传入的参数。

名称	类型	描述
success	bool	是否成功移除

错误码

错误码	描述
4	无权限调用

1.6.2.10. 分发消息

前端可以通过此接口给客户端发送通知，H5 会统一给传入的 `name` 标记加一个前缀 `NEBULANOTIFY_`，然后作为通知名称进行发送。

在 Android 10.1.0 之前的版本中，`postNotification` 的 `data` 字段不得为空，`addNotifyListener` 才可以接受。而在 10.1.60 版本中，此限制已被取消，`addNotifyListener` 可以接受一个空数据。

Android 是通过 `LocalBroadcastManager` 发送的广播，可以通过监听 `NEBULANOTIFY_` + `name` 来进行监听。

postNotification 接口的使用方法

```
AlipayJSBridge.call('postNotification', {
  name:'fortest',
  data:{}
}, function (result) {
  console.log(result);
});
```

代码示例

以下示例为基本功能演示：

```
<h1>请点击下面的按钮来进行测试</h1>
<p>这里只测试在同一个页面内的情况，这个 API 可以在不同应用间通信</p>

<a href="#" class="btn start">开始监听</a>
<a href="#" class="btn stop">停止监听</a>
<a href="#" class="btn send">发通知</a>
<script>
function callback(e){
  alert(JSON.stringify(e));
};

function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('.start').addEventListener('click', function(){
    AlipayJSBridge.call('addNotifyListener', {
      name:'NEBULANOTIFY_TEST_EVENT' // H5 发出的事件必须以 `NEBULANOTIFY_` 开通进行监听
    }, callback);
  });

  document.querySelector('.stop').addEventListener('click', function(){
    AlipayJSBridge.call('removeNotifyListener', {
      name:'NEBULANOTIFY_TEST_EVENT' // H5 发出的事件必须以 `NEBULANOTIFY_` 开通进行监听
    }, function(e){
      alert(JSON.stringify(e));
    });
  });

  document.querySelector('.send').addEventListener('click', function(){
    AlipayJSBridge.call('postNotification', {
      name:'TEST_EVENT', // H5 发出的事件必须以 `NEBULANOTIFY_` 开通进行监听
      data: {
        hello: 'world'
      }
    }, function(e){
      alert(JSON.stringify(e));
    });
  });
});
</script>
```

API

```
AlipayJSBridge.call('postNotification', {
  name, data
}, fn)
```

入参

名称	类型	描述	必选	默认值
name	string	通知名称。	Y	“”

data	object	通知给客户端带的信息，Android 下会把 JSON 数据遍历，然后把里面每一项的 Value 都转成 String 类型发送，请注意兼容数据格式。	N	-
fn	function	回调函数。	N	-

出参

`result: {success}`：回调函数带入的参数。

名称	类型	描述
success	bool	是否消息发送成功。

错误码

错误码	描述
4	无权限调用。

1.6.3. 页面上下文

1.6.3.1. 打开新页面

`pushWindow` 用于在同一个离线包内打开一个新的页面，打开时自带系统转场动画。若您需要跨 appId 打开其他离线应用页面，请使用 startApp 接口。

`pushWindow` 与前端 `location.href` 的区别，类似于 PC 浏览器的新开标签页，每个 window 都是一个新的标签页，因此原页面仅仅是被压到后台，状态始终保持，JS 也会继续运行。

pushWindow 接口的使用方法

```
// 打开淘宝首页，自动读取 title，并且去除右边菜单按钮
AlipayJSBridge.call('pushWindow', {
  url: 'https://m.taobao.com/',    // 要打开页面的 URL
  // 打开页面的配置参数
  param: {
    readTitle: true,               //自动读取 title
    showOptionMenu: false          // 隐藏右边菜单
  },
  // 用于给新开的页面传递参数，可选
  // 在新开的页面使用 AlipayJSBridge.startupParams 可以获取到 passData
  passData: {
    key1: "key1Value",
    key2: "key2Value"
  }
});
```

> **重要**
>
> - 对于 Android 应用，需要将参数放在 `param: { }` 中。
> - 对于 iOS 应用，则需要将参数放在 `passData: { }` 中。

代码示例

- 打开淘宝首页，去除右边菜单。

```
<h1>打开淘宝首页</h1>
<a class="btn J_demo">执行</a>
<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
      callback && callback();
  } else {
      // 如果没有注入则监听注入的事件
      document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('a').addEventListener('click', function() {
      // 打开淘宝首页，自动读取 title，并且去除右边菜单
      AlipayJSBridge.call('pushWindow', {
          url: 'https://m.taobao.com/',
          param: {
              readTitle: true,
              showOptionMenu: false
          }
      });
  });
});
</script>
```

- 打开时设置透明标题栏。

```
<h1>打开淘宝首页</h1>
<a class="btn J_demo">执行</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('a').addEventListener('click', function() {
  AlipayJSBridge.call('pushWindow', {
    url: 'https://m.taobao.com',
    param: {
      transparentTitle: 'auto'
    }
  });
});
});
</script>
```

API 说明

> **重要**
>
> - pushWindow 打开 URL 页面的时候不会关闭已经存在的页面，注意打开的数量，不要同时开太多而影响性能。
> - 建议同一个应用 pushWindow 的层级不要超过 5 层，否则会影响用户体验而且有可能导致应用崩溃。

```
AlipayJSBridge.call('pushWindow', {
    url, param, passData
})
```

入参

名称	类型	描述	必选	默认值
url	string	要打开的 URL。	Y	“”
param	dictionary	支持的 key/value 对下面的表格。	N	{}
passData（仅适用于 iOS)	dictionary	传递的参数，打开页面使用 `AlipayJSBridge.startupParams` 获取。	N	{}

param 参数详解

名称	类型	描述	默认值
defaultTitle	string	默认标题，在页面第一次加载之前显示在标题栏上。	“”
showLoading	bool	是否在页面加载前显示全局菊花。	false
readTitle	bool	是否读取网页标题显示在 titleBar 上。	true
pullRefresh	bool	是否支持下拉刷新只有本地文件允许设置为 true。	false
allowsBounceVertical	bool	页面是否支持纵向拽拉超出实际内容，Android 只支持下拉（显示出背景或者域名）。只有 `.alipay.com/.alipay.net/` 本地文件允许设置为 false。	true
bounceTopColor	int	下拉超出时，顶部间隙颜色（十进制，例如：bc=16775138）。	-
showTitleLoading	bool	是否在 TitleBar 的标题左边显示小菊花。	false
transparentTitle	string	不能与 `titleBarColor` 同时使用。always/auto： 1、`always`：当前页面上下滚动时，titlebar 一直全透明。 2、`auto`：当页面往下滚动时，透明度不断增加，直到 80 pt 时变成完全透明，此时页面再往上滚动则反之，透明度不断减小直到回到顶部时变成完全不透明。 3、`none`：如果这个页面不需要透明效果，则需要用 `pushWindow` 的 `param` 参数重新指定 `transparentTitle` 为“none”。	-
titleBarColor	int	自定义 titlebar 的背景色（十进制，例如：bc=16775138）。 **重要** 不能与 `transparentTitle` 同时使用。	-
scrollDistance	int	在 `transparentTitle` 为 auto 的情况下，滑动到透明度为 0.96 的距离。	80（iOS)

titleImage	string	所要使用的 title 图片地址，请上传一张 3X PNG 图，只影响当前的 VC，`pushWindow` 不会自动传递此参数，为了更好的体验可以把图放在全局运营资源包中。	“”
closeCurrentWindow	bool	打开窗口的同时，关闭当前窗口。	false
closeAllWindow	bool	打开窗口的同时，关闭当前 App 的所有窗口。	false
animationType	string	动画类型，默认为 “push”，可用枚举 “none”/“push”。 **重要** Android 未实现，均无动画。	“”

pushWindow 参数默认继承

名称	继承	备注
url	Y	-
defaultTitle	Y	-
backBehavior	Y	优先用户指定，否则 pop。
showLoading	Y	-
readTitle	Y	-
pullRefresh	Y	-
toolbarMenu	Y	-
showProgress	Y	-
defaultSubtitle	Y	-
backgroundColor	Y	-
canPullDown	Y	-
showOptionMenu	Y	-
showTitleLoading	Y	-
showDomain	Y	-

pushWindow 和 location.href 的区别

您可以根据以下内容理解 `pushWindow` 和 `location.href` 的区别：

- 首先把 Nebula 容器理解为一个 PC 浏览器。
- window 可以理解为标签页，location.href 就是正常的标签跳转。
- pushWindow 实际上就是新开了一个标签页，并且把之前那一页压到下面，push 出来的这页放在顶层展现。此时被压到下面的上一页所有状态均保留。
- location.href 就是依然停留在这个标签页，直接做页面跳转。
- 当 pushWindow 出来的标签页被关闭（pop）时，如果之前还有 window 存在，那么之前那个 window 就会恢复显示，并触发 `resume` 。

1.6.3.2. 关闭当前页面

此接口用来关闭当前页面。

popWindow 接口的使用方法

```
// 关闭当前打开的页面
AlipayJSBridge.call('popWindow');
```

代码示例

- 关闭当前页面：

```
<h1>关闭当前页面</h1>
<a href="#" class="btn J_demo">执行</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function(){
document.querySelector('a').addEventListener('click', function() {
  AlipayJSBridge.call('popWindow');
});
});
</script>
```

- 关闭当前页面并且传递数据：

```
<h1>点击"新开窗口"，然后点击"回退窗口"查看效果</h1>
<a href="#" class="btn pop">回退窗口</a>
<a href="#" class="btn new">新开窗口</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('.new').addEventListener('click', function() {
  AlipayJSBridge.call('pushWindow', {
    url: location.pathname
  });
});

document.querySelector('.pop').addEventListener('click', function() {
  AlipayJSBridge.call('popWindow', {
    data: {
      from: location.href,
      info: Date.now()
    }
  });
});

document.addEventListener('resume', function(event) {
  alert('页面回退时带过来的内容： ' + JSON.stringify(event.data));
});
});
</script>
```

API 说明

> **重要**
>
> 关于 `popWindow` 时所带的 data 如何被接收，请查看 页面恢复运行（resume 事件）。

```
AlipayJSBridge.call('popWindow', {
  data
})
```

入参

名称	类型	描述	必选
data	object	传递给当前 App 内即将露出页面的内容，无法跨 appId 传递数据。	N

1.6.3.3. 关闭多个页面

此接口用于一次回退多级页面。

> **说明**
>
> 只允许 `popTo` 到当前 App 实例内的页面，不允许跨 appId 的跳转。

popTo 接口的使用方法

```
// 关闭当前打开的页面
AlipayJSBridge.call('popTo', {
  index: -1
});
```

代码示例

- 关闭当前页面并且传递数据：

```
<h1>点击"执行"关闭当前页面并返回数据</h1>
<a href="#" class="btn J_demo">执行</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function(){
document.querySelector('.J_demo').addEventListener('click', function() {
  // 传递的 data 对象将会被即将露出的页面通过 resume 事件接收
  AlipayJSBridge.call('popTo', {
    index: -1, // 回退到上一个页面，假如这个时候没有上一个页面，就会报错
    data: { // 特别说明：data 是字典，不是数组
      from: location.href,
      info: Date.now()
    }
  }, function(e) { // 添加回调，因为 popTo 不一定会成功（当前页面是唯一打开的页面的时候，会报错）
    alert(JSON.stringify(e));
  });
});
});
</script>
```

- 通过 `urlPattern` 返回到符合正则表达式匹配的页面：

```
<h1>返回到符合某个规则的 URL</h1>
<h3></h3>
<a href="javascript:void(0)" class="btn J_new_window">新窗口打开当前页面</a>
<a href="javascript:void(0)" class="btn J_demo">返回</a>
<script>
var query = getQuery();
var depth = (+query.depth) || 0;
document.querySelector('h3').innerHTML = '当前页面深度: ' + depth;
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function(){
document.querySelector('.J_demo').addEventListener('click', function() {
  AlipayJSBridge.call('popTo', {
    urlPattern: 'pop-to-url-pattern.html',
  }, function(e) {
    alert(JSON.stringify(e));
  });
});

document.querySelector('.J_new_window').addEventListener('click', function() {
  AlipayJSBridge.call('pushWindow', {
    url: location.pathname + '?depth=' + (1+depth),
  });
});
});
</script>
```

API 说明

> **重要**
>
> - 一般情况下，`popTo` 用于分多步才能完成的场景，比如用户信息完善时回退，三级地址选择后返回等。
> - 如果通过 `urlPattern` 来返回，`popTo` 会返回到离当前页面最远的页面，也就是栈底。同时不会去检测当前页面的 URL 是否符合。
> - 关于 `popTo` 时所带的数据如何被接收，请查看 页面恢复运行（resume 事件）。

```
AlipayJSBridge.call('popTo',{
  index, urlPattern
}, fn)
```

`index` 、`urlPattern` 是两种查询模式，不应该同时使用。

入参

名称	类型	描述	必选	默认值
index	int	目标界面在会话界面栈中的索引；如果小于零，则将与当前界面的 index 相加。	Y	-
urlPattern	string	目标界面的 URL 匹配表达式，URL 如果包含 `urlPattern` ，匹配成功。	Y	“”

fn	function	操作成功时，回调可能不被调用；操作失败时，回调函数执行。	N	-

出参

名称	类型	描述
result	undefined	操作成功时，回调可能不被调用；result 不应被使用。

错误码

错误码	描述
10	未配置参数；无效的 index；未匹配 `urlPattern` 。

1.6.3.4. 启动其他应用

此接口用于跨包打开 mPaaS 应用内的其他 H5 应用（离线包）。

startApp 接口的使用方法

```
AlipayJSBridge.call('startApp', {
  appId: '90000000',
  param: {
    url: '/index.html'
  }
}, function(result) {
  // noop
});

// 注意，如果要打开多个 App 实例：
// 请将 appClearTop 和 startMultApp 都放在 param 里
AlipayJSBridge.call('startApp', {
  appId: '90000000',
  param: {
    url: location.href,
    appClearTop: false,
    startMultApp: 'YES' // 注意这个值是 YES，而不是 bool 类型
  }
}, function(result) {
  // noop
});
```

代码示例

- 打开标题栏透明的应用：

```
<h1>点击按钮查看效果</h1>
<a href="javascript:void(0)" class="btn dream">打开心愿储蓄</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function(){
document.querySelector('.dream').addEventListener('click', function() {
  AlipayJSBridge.call('startApp', {
    appId: '20000981',
    param: {
      url: '/www/dream-create.html',
      // 启动参数传入
      canPullDown: true,
      transparentTitle: 'auto'
    }
  }, function(result) {
    // noop
  });
});
});
</script>
```

- 打开新应用并关闭当前应用：

```
<h1>点击按钮打开新应用，当前应用会被关闭</h1>
<a href="javascript:void(0)" class="btn forest">打开蚂蚁森林</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('.forest').addEventListener('click', function() {
  AlipayJSBridge.call('startApp', {
      appId: '60000002',
      // 不传入 URL，就会使用 App 默认配置的 URL
      param: {
        transparentTitle: 'auto'
      },
      closeCurrentApp: true
    }, function(result) {
      // noop
    });
});
});
</script>
```

- 默认只开一个 appId 实例：

```
<h1>尝试再打开当前页面，先退出当前应用，然后再次打开</h1>
<a href="javascript:void(0)" class="btn self">打开当前页面</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function(){
document.querySelector('.self').addEventListener('click', function() {
  AlipayJSBridge.call('startApp', {
    // 当前页面打开的时候，是通过通用应用 20000067 打开，
    // 因此在此 startApp 的时候，就会把其他的 20000067 关闭
    // 所以这个时候其实还是只有一个当前页面打开
    appId: '20000067',
    param: {
      url: location.href,
    }
  }, function(result) {
    // noop
  });
});
});
</script>
```

- 打开多个相同 appId 的应用：

```
<h1>打开多个相同 appId 的应用</h1>
<a href="javascript:void(0)" class="btn multi">再开启一个应用</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('.multi').addEventListener('click', function() {
  AlipayJSBridge.call('startApp', {
    appId: '90000000',
    param: {
      url: '/index.html',
      appClearTop: false,
      startMultApp: 'YES' // 注意这个值是 YES，而不是 bool 类型
    }
  }, function(result) {
    // noop
  });
});
});
</script>
```

API 说明

```
AlipayJSBridge.call('startApp', {
  appId, param: {}, closeCurrentApp
}, fn)
```

入参

名称	类型	描述	必选	默认值	基线
appId	string	离线包 ID。	Y	“”	-
param	dictionary	启动应用的参数，由具体业务应用定义。如打开离线包时需指定 URL，请在 param 中配置。若要运行多个实例，参见下方 注意事项。	N	value 值支持字符、 bool、int、double	-
closeCurrentApp	bool	是否先退出当前 App 再启动新的 App。适用于页面用作中转页的情况。	N	-	>10.1.60
fn	function	调用失败后的回调函数。	N	-	-

错误码描述

错误码	描述
10	指定 appId 无效。
11	启动 App 失败。

注意事项

- `startApp` 是用来打开 App 的，因此其定位是 App 级别，在这一点上与 `pushWindow` 不同。
- 默认情况下，如果当前 App 已经打开了，再次使用 `startApp` 打开 App 时，会进行一个类似重启的操作。也就是说不会出现同时运行两个具有相同 appId 的实例的情况。
- 假如一个 appId 要运行多个实例，那么请在 param 参数中添加 `appClearTop=false&startMultApp=YES` 选项。

1.6.3.5. 退出当前应用

此接口用于退出当前栈顶 App。

exitApp 接口的使用方法

```
AlipayJSBridge.call('exitApp');
```

代码示例

- 退出当前的页面：

```
<h1>点击退出当前页面</h1>
<a href="#" class="btn J_demo">执行</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('.J_demo').addEventListener('click', function() {
  AlipayJSBridge.call('exitApp');
});
});
</script>
```

- 页面跳转完整示例：

```
<h1>请点击下面按钮来进行页面间跳转</h1>
<h3></h3>
<a href="javascript:void(0)" class="btn new">新开当前页面</a>
<a href="javascript:void(0)" class="btn back">返回一级</a>
<a href="javascript:void(0)" class="btn popTo">通过 popTo 退 2 级</a>
<a href="javascript:void(0)" class="btn exit">关闭所有页面</a>
<script>
var query = getQuery();
var depth = (+query.depth) || 0;
document.querySelector('h3').innerHTML = '当前页面深度： ' + depth;
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('.new').addEventListener('click', function() {
  AlipayJSBridge.call('pushWindow', {
    url: location.pathname + '?depth=' + (1+depth),
  });
});
document.querySelector('.back').addEventListener('click', function() {
  AlipayJSBridge.call('popWindow', {
    data: {
      method: 'popWindow',
      depth: depth,
    }
  });
});
document.querySelector('.popTo').addEventListener('click', function() {
  AlipayJSBridge.call('popTo', {
    index: -2,
    data: {
      method: 'popTo',
      depth: depth,
    }
  }, function(e) {
    if (e.error) {
      alert('发生错误: ' + JSON.stringify(e));
    }
  });
});
document.querySelector('.exit').addEventListener('click', function() {
  AlipayJSBridge.call('exitApp');
});
});
document.addEventListener('resume', function(event) {
alert('页面回退时带过来的内容： ' + JSON.stringify(event.data));
});
</script>
```

API 说明

> **重要**
>
> 对于没有申请过 AppId 的页面，都是以 `20000067` 这个 AppId 运行，因此任何页面内调用 `exitApp` ，所有页面都会关闭。

```
AlipayJSBridge.call('exitApp', {
  closeActionType, animated
}, fn);
```

入参

名称	类型	描述	必选	默认值
closeActionType	string	`exitSelf` ：退出自身应用。 `exitTop` ：退出栈顶应用。	N	-
animated	bool	是否开启动画。	N	true

1.6.3.6. 打开 H5 应用

使用 startH5App 方法打开 H5 应用，此方法目前已经不再支持，如有需求请使用 startApp 接口。

startH5App 接口的使用方法

```
AlipayJSBridge.call('startH5App', {
  appId: '20000067',
  url: '/www/index.html',
});
```

代码示例

```
<h1>点击按钮查看效果</h1>
<a href="javascript:void(0)" class="btn dream">打开心愿储蓄</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function(){
document.querySelector('.dream').addEventListener('click', function() {
  AlipayJSBridge.call('startH5App', {
    appId: '20000981',
      url: '/www/dream-create.html',
      // 启动参数传入
  });
});
});
</script>
```

API 说明

```
AlipayJSBridge.call('startH5App', {
  appId, url, ...param
})
```

入参

参数	类型	描述	必填
appId	string	应用 H5 appId。	Y
url	string	启动应用的参数，由具体业务应用定义。	N
param	object	k-v 形式，不支持嵌套。	N

1.6.4. Native 功能

1.6.4.1. 扫码解析

此接口用于调用扫码组件，且仅限于 Android 系统。另外，在使用此接口前，请确认您已经在工程中添加了扫码组件。其中 actionType 表示获取码值。

scan 接口的使用方法

```
AlipayJSBridge.call('scan', {
  type: 'bar',
  actionType: 'scan'
}, function(result) {
  alert(JSON.stringify(result));
});
```

代码示例

获取二维码的信息：

```
<h1>点击扫码后输出码对应的信息</h1>
<a href="#" class="btn read">开始扫码</a>
<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.read').addEventListener('click', function() {
    AlipayJSBridge.call('scan', {
      type: 'qr'
    }, function(result) {
      alert(JSON.stringify(result));
    });
  });
});
</script>
```

API

```
AlipayJSBridge.call('scan', {
  type, actionType, qrcode
}, fn);
```

入参

属性	类型	描述	必填	默认值
type	String	扫描目标类型有二维码和条形码。	Y	“”
actionType	String	操作类型，“scan”指获取码值。	N	“scan”
qrcode	String	指定用于“route”操作类型的码值。	N	“”
fn	function	扫码获取码信息后的回调函数。	N	-

出参

回调函数带入的参数 `result: {error, barCode, qrCode, cardNumber}` 。

属性	类型	描述
barCode	String	扫描所得条码数据。
qrCode	String	扫描所得二维码数据。
error	int	错误码： 10：用户取消。 11：操作失败。

1.6.5. 界面

1.6.5.1. 警告框

该接口用于警告框的 native 实现。

alert 接口的使用方法

```
AlipayJSBridge.call('alert', {
  title: '亲',
  message: '你好',
  button: '确定'
}, function(e) {
  alert(JSON.stringify(e));
});
```

代码示例

alert 和 confirm：

```
<h1>点击以下按钮看不同效果</h1>
<a href="javascript:void(0)" class="btn alert">点击 Alert</a>
<a href="javascript:void(0)" class="btn confirm">点击 Confirm</a>
<script>
function ready(callback) {
// 如果 jsbridge 已经注入则直接调用
if (window.AlipayJSBridge) {
  callback && callback();
} else {
  // 如果没有注入则监听注入的事件
  document.addEventListener('AlipayJSBridgeReady', callback, false);
}
}
ready(function() {
document.querySelector('.alert').addEventListener('click', function() {
  AlipayJSBridge.call('alert', {
    title: '亲',
    message: '你好',
    button: '确定'
  }, function(e) {
    alert(JSON.stringify(e));
  });
});
document.querySelector('.confirm').addEventListener('click', function() {
  AlipayJSBridge.call('confirm', {
    title: '亲',
    message: '确定要退出吗？',
    okButton: '是',
    cancelButton: '否'
  }, function(e) {
    alert(JSON.stringify(e));
  });
});
});
</script>
```

API 说明

> **重要**
>
> 与 `window.alert` 不同的是，`alert` 不是阻塞式的，即如果先后弹了 2 个警告框，最后看到的是后弹的那个。

```
AlipayJSBridge.call('alert',{
  title, message, button
}, fn)
```

入参

属性	类型	描述	必填	默认值
title	string	Alert 框标题。	N	“”
message	string	Alert 框文本。	N	“”
align	string	message 对齐方式，枚举值包括 left、center、right。	N	iOS：“center” Android：“left”

button	string	按钮文字。	N	“确定”
fn	function	回调函数，当点击 button 后被调用。	N	-

1.6.5.2. 确认框

确认框的 native 实现。

confirm 接口的使用方法

```
AlipayJSBridge.call('confirm', {
  title: '亲',
  message: '确定要退出吗？',
  okButton: '是',
  cancelButton: '否'
}, function(e) {
  alert(JSON.stringify(e));
});
```

代码示例

alert 和 confirm：

```
<h1>点击以下按钮看不同效果</h1>
<a href="javascript:void(0)" class="btn alert">点击Alert</a>
<a href="javascript:void(0)" class="btn confirm">点击Confirm</a>
<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.alert').addEventListener('click', function() {
    AlipayJSBridge.call('alert', {
      title: '亲',
      message: '你好',
      button: '确定'
    }, function(e) {
      e && alert(JSON.stringify(e))
    });
  });
  document.querySelector('.confirm').addEventListener('click', function(){
    AlipayJSBridge.call('confirm', {
      title: '亲',
      message: '确定要退出吗？',
      okButton: '是',
      cancelButton: '否'
    }, function(e) {
      alert(JSON.stringify(e))
    });
  });
});
</script>
```

API 说明

> **重要**
> `confirm` 与 `alert` 都不是阻塞式的，即如果先后弹出两个确认框，最后会看到后弹出的那个。

```
AlipayJSBridge.call('confirm',{
  title, message, okButton, cancelButton
}, fn)
```

入参

属性	类型	描述	必填	默认值
title	string	Alert 框标题	N	“”
message	string	Alert 框文本	N	“”
align	string	message 对齐方式，可用枚举值有 left、center、right。	N	iOS：“center” Android：“left”
okButton	string	确定按钮文字。	N	“确定”
cancelButton	string	取消按钮文字。	N	“取消”
fn	function	回调函数，当单击按钮后被调用。	N	-

1.6.5.3. 弱提示

此接口用于显示一个弱提示，可选择多少秒之后消失。

toast 接口的使用方法

```
AlipayJSBridge.call('toast', {
  content: '操作成功',
  type: 'success',
  duration: 2000
}, function() {
  alert("toast消失后执行");
});

// 可以通过 hideToast 接口隐藏已经弹出的 toast

AlipayJSBridge.call('hideToast', {}, function() {
});
```

代码示例

```
<h1>点击以下按钮看不同效果</h1>
<a href="javascript:void(0)" class="btn success">显示成功信息</a>
<a href="javascript:void(0)" class="btn fail">显示失败信息</a>
<a href="javascript:void(0)" class="btn exception">显示异常信息</a>
<a href="javascript:void(0)" class="btn none">只显示信息</a>
<a href="javascript:void(0)" class="btn duration">3.5s信息显示</a>

<script>
function toast(config, callback){
  AlipayJSBridge.call('toast',config, callback);
}

function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.success').addEventListener('click', function() {
    toast({
      content: '操作成功',
      type: 'success'
    });
  });

  document.querySelector('.fail').addEventListener('click', function() {
    toast({
      content: '网络繁忙，请稍后再试',
      type: 'fail'
    });
  });

  document.querySelector('.exception').addEventListener('click', function() {
    toast({
      content: '发生异常，请注意',
      type: 'exception'
    });
  });

  document.querySelector('.none').addEventListener('click', function() {
    toast({
      content: '欢迎光临',
    });
  });

  document.querySelector('.duration').addEventListener('click', function() {
    toast({
      content: '请稍等',
      duration: 3500,
    }, function(e){
      alert('toast消失回调');
    });
  });
});
</script>
```

API 说明

> **重要**
>
> - toast 虽然会自动关闭，但是也可以通过 hideLoading 来关闭，这种使用方式不常见，但是要防止被 hideLoading 关闭。
> - 对于 Android 系统，如果系统通知关闭，则此弹框不会出现。
> - 在 Android 10.1.2 以下版本中，duration 显示时长只支持 2000/3500 两种，小于或等于 2000 的选择 2000，大于 2000 的选择 3500。

```
AlipayJSBridge.call('toast', {
  content, type, duration
}, fn)
```

入参

属性	类型	描述	必填	默认值
content	string	文字内容。	Y	“”
type	string	参数值可以是 none、success、fail、exception。 当为 excption 类型时，必须传递文案。	N	“none”
duration	int	显示时长，单位为毫秒。	N	2000
xOffset	float	左为正方向，单位为 px。	N	0
yOffset	float	上为正方向，单位为 px。	N	0
fn	function	回调函数，当 toast 消失后被调用。	N	-

1.6.5.4. 选择列表

此接口提供一种有选项的列表，它将停留在屏幕的下边沿。

actionSheet 接口的使用方法

```
AlipayJSBridge.call('actionSheet', {
  'title': '标题',
  'btns': ['第一个按钮', '第二个按钮', '第三个按钮'],
  'cancelBtn': '取消',
  'destructiveBtnIndex': 2
}, function(data) {
  switch (data.index) { // index 标示用户点击的按钮在 actionSheet 中的位置，从 0 开始
    case 0:
      alert('第一个按钮');
      break;
    case 1:
      alert('第二个按钮');
      break;
    case 2:
      alert('第三个按钮');
      break;
    case 3:
      alert('取消按钮');
      break;
  }
});
```

代码示例

```
<h1>点击按钮调出actionSheet</h1>
<a href="javascript:void(0)" class="btn actionSheet">打开 actionSheet</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.actionSheet').addEventListener('click', function() {
    AlipayJSBridge.call('actionSheet',{
      'title': '标题',
      'btns': ['第一个按钮', '第二个按钮', '第三个按钮'],
      'cancelBtn': '取消',
      'destructiveBtnIndex': 2
    }, function(data) {
      switch (data.index) { // index 标示用户点击的按钮，在 actionSheet 中的位置，从 0 开始
        case 0:
          alert('第一个按钮');
          break;
        case 1:
          alert('第二个按钮');
          break;
        case 2:
          alert('第三个按钮');
          break;
        case 3:
          alert('取消按钮');
          break;
      }
    });
  });
});
</script>
```

API 说明

```
AlipayJSBridge.call('actionSheet',{
  title, btns, cancelBtn, destructiveBtnIndex
}, fn)
```

入参

属性	类型	描述	必填	默认值
title	string	标题。	N	“”
btns	array	一组按钮，item 类型是 string。	Y	[]
cancelBtn	string	设置取消按钮及文字。	N	“”

destructive BtnIndex	int	经 iOS 特殊处理过的指定按钮的索引号从 0 开始。 用于需要删除或清除数据等场景，默认为红色。	N	-
fn	function	不是 API 调用后被回调，而是用户选择选项之后的回调函数。	N	-

出参

格式为 `{data: {index: 0}}` 。其中 `index` 是用户单击的按钮在 actionSheet 中的位置，从 0 开始。

1.6.5.5. 设置标题

此接口用于设置页面的标题栏，包括主标题、副标题以及标题菜单项。

> **说明**
> 由于苹果的 ATS 限制，Image URL 必须为 HTTPS 链接或 Base64，而 HTTP 链接会被忽略。

setTitle 接口的使用方法

```
AlipayJSBridge.call('setTitle', {
  title: '标题',
});
```

代码示例

设置各种类型的标题栏：

```
<h1>点击以下按钮看不同效果</h1>
<a href="javascript:void(0)" class="btn title">只设置标题</a>
<a href="javascript:void(0)" class="btn subTitle">标题+副标题</a>
<a href="javascript:void(0)" class="btn clear">清空标题</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('.title').addEventListener('click', function() {
    AlipayJSBridge.call('setTitle', {
      title: '标题'
    });
  });

  document.querySelector('.subTitle').addEventListener('click', function() {
    AlipayJSBridge.call('setTitle', {
      title: '标题',
      subtitle: '副标题'
    });
  });

  document.querySelector('.clear').addEventListener('click', function() {
    AlipayJSBridge.call('setTitle', {
      title: ' ',
      subtitle: ' ',
    });
  });
});
</script>
```

API 说明

```
AlipayJSBridge.call('setTitle',{
  title, subtitle, image
}, fn)
```

入参

属性	类型	描述	必填	默认值
title	string	主标题文案。	N	“”
subtitle	string	副标题文案。	N	“”
image	string	支持 URL 或者 Base64，请使用一张 3X 图，如果设置了 image，则前两个参数失效，并且不从 WebView 的回调中读取 title。	N	“”

注意事项

在 Android 10.0.18 版本之前不支持设置空的 title，但是可以通过设置一个不可见的字符串绕过这个限制，不过在 Android 10.0.20 版本中已经去掉了这个限制。

```
AlipayJSBridge.call('setTitle', {
  title: '\u200b',
});
```

1.6.5.6. 设置导航栏底部细线颜色

此接口用于自定义导航栏底部细线的颜色，当细线的颜色与导航栏相同时可达到隐藏的效果。

setBarBottomLineColor 接口的使用方法

```
AlipayJSBridge.call("setBarBottomLineColor", {
  color: 16711688
});
```

代码示例

```
<div style="padding-top:80px;">
  <a href="javascript:void(0)" class="btn title">设置导航栏底部细线颜色</a><br>
</div>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('.title').addEventListener('click', function() {
    AlipayJSBridge.call("setBarBottomLineColor", {
      color: parseInt('ff0000', 16)
    });
  }, false);
},false);
</script>
```

API 说明

```
AlipayJSBridge.call('setBarBottomLineColor',{
  color
}, fn)
```

入参

属性	类型	描述	必填	默认值
color	int	色值，十进制。	Y	-

1.6.5.7. 设置导航栏背景色

此接口用于设置 TitleBar 的颜色。

setTitleColor 接口的使用方法

```
AlipayJSBridge.call("setTitleColor", {
  color: 16775138,
  reset: false //（可选，默认为 false） 是否重置 title 颜色为默认颜色。
});
```

> **说明**
>
> 使用 setTitleColor 接口时，需要在启动页面前配置参数 `H5Param.LONG_TRANSPARENT_TITLE` 为 always 或者 auto。
> 示例如下：`param.putString(H5Param.LONG_TRANSPARENT_TITLE,"auto");`

代码示例

```
<div style="padding-top:80px;">
  <a href="javascript:void(0)" class="btn title">设置导航栏背景色</a>
  <a href="javascript:void(0)" class="btn reset">重置导航栏背景色</a>
  <a href="javascript:void(0)" class="btn pushWindow">新开一个透明导航栏的窗口</a>
  <a href="javascript:void(0)" class="btn resetTransparent">重置为透明导航栏</a>
</div>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(){
  document.querySelector('.title').addEventListener('click', function() {
    AlipayJSBridge.call("setTitleColor", {
      color: parseInt('108ee9', 16),
      reset: false //  （可选，默认为 false） 是否重置 title 颜色为默认颜色。
    });
  });

  document.querySelector('.reset').addEventListener('click', function() {
    AlipayJSBridge.call("setTitleColor",{
      color: 16775138,
      reset: true
    });
  });

  document.querySelector('.pushWindow').addEventListener('click', function() {
    AlipayJSBridge.call("pushWindow", {
      url: location.pathname + '?__webview_options__=transparentTitle%3Dalways'
    });
  });

  document.querySelector('.resetTransparent').addEventListener('click', function() {
    AlipayJSBridge.call("setTitleColor", {
      color: 16775138,
      resetTransparent: true
    });
  });
});
</script>
```

API 说明

```
AlipayJSBridge.call('setTitleColor', {
  color: 16775138,
  reset: false,
  resetTransparent: false
}, fn)
```

入参

属性	类型	描述	必填	默认值
color	int	色值，十进制。	Y	-

1.6.5.8. 设置右上角按钮

此接口用于设置标题栏右边的按钮属性（仅负责设置），该按钮的显示需要额外调用 `showOptionMenu` 。

> **说明**
> 由于苹果的 ATS 限制，icon URL 必须为 HTTPS 链接或 Base64，而 HTTP 链接会被忽略。

setOptionMenu 接口的使用方法

```
AlipayJSBridge.call('setOptionMenu', {
  title : '按钮',  // 与 icon、icontype 三选一
  redDot : '-1', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
  color : '#ff00ff00', // 必须以 # 开始 ARGB 颜色值
});
```

代码示例

设置各种类型的右侧按钮：

```
<h1>点击以下按钮看不同效果</h1>

<a href="javascript:void(0)" class="btn button">单按钮</a>
<a href="javascript:void(0)" class="btn icon">单图标</a>
<a href="javascript:void(0)" class="btn menu">多菜单(9.9.3)</a>
<a href="javascript:void(0)" class="btn reset">重置</a>
<a href="javascript:void(0)" class="btn hide">隐藏</a>
<a href="javascript:void(0)" class="btn show">显示</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function(e) {
  document.querySelector('.button').addEventListener('click', function() {
    AlipayJSBridge.call('setOptionMenu', {
      title : '按钮',
      redDot : '5', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
      color : '#ff00ff00', // 必须以#开始 ARGB 颜色值
    });
    AlipayJSBridge.call('showOptionMenu');
  });
```

```
  });

  document.querySelector('.icon').addEventListener('click', function() {
    AlipayJSBridge.call('setOptionMenu', {
      icon : 'http://pic.alipayobjects.com/e/201212/1ntOVeWwtg.png',
      redDot : '-1', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
    });
    AlipayJSBridge.call('showOptionMenu');
  });

  document.querySelector('.menu').addEventListener('click', function() {
    AlipayJSBridge.call('setOptionMenu', {
      // 显示时的顺序为从右至左
      menus: [{
        icontype: 'scan',
        redDot: '-1', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
      }, {
        icontype: 'user',
        redDot: '-1', // -1 表示不显示，0 表示显示红点，1-99 表示在红点上显示的数字
      }],
      override: true // 在需要设置多个 option 的情况下，是否保留默认的 optionMenu
    });

    // 必须强制调用一次以刷新界面
    AlipayJSBridge.call('showOptionMenu');
  });

  document.querySelector('.reset').addEventListener('click', function() {
    AlipayJSBridge.call('setOptionMenu', {
      reset: true,
    });
    AlipayJSBridge.call('showOptionMenu');
  });

  document.querySelector('.show').addEventListener('click', function() {
    AlipayJSBridge.call('showOptionMenu');
  });

  document.querySelector('.hide').addEventListener('click', function() {
    AlipayJSBridge.call('hideOptionMenu');
  });

  document.addEventListener('optionMenu', function(e) {
   alert(JSON.stringify(e.data));
  }, false);
});
</script>
```

API 说明

> **重要**
> 设置 `setOptionMenu` 后，如果效果不对，请调用一次 `showOptionMenu` 。

reset、title、icontype、icon 这四个属性有一个即可，属性的优先级为：reset > title > icontype > icon。

```
AlipayJSBridge.call('setTitle',{
  title, icon, redDot, reset, color, override, menus, icontype
})
```

入参

属性	类型	描述	必填	默认值
title	string	右按钮文字。	Y	“”
icon	string	右按钮图标 URL，base64（since 9.0）。 8.3 及以前版本：iOS 40x40（周边不留白），Android 50x50（四边各透明留白 5px）。 8.4 及以后版本：两个平台统一使用 40x40（周边不留白）。	Y	“”
redDot	string	红点数值。	N	“”
reset	bool	重置为系统默认，当 `reset=true` 时，忽略其他参数。	Y	false
color	string	文字颜色值。	N	“#FFFFFFFF”
override	bool	在需要设置多个 option 的情况下，是否保留默认的 optionMenu。	N	false
menus	array	设置多个按钮。	N	[ ]
preventDefault	bool	是否阻止默认的分享功能（默认是弹分享框）preventDefault=true 时，会阻止默认的分享。	N	[ ]
icontype	string	根据图片类型加载容器预置图片，可选择其中之一：icontype、title、icon。 **重要** 只支持单个 optionMenu 变色。 具体类型包含 user（账户）、filter（筛选器）、search（查找）、add（添加）、settings（设置）、scan（扫一扫）、info（信息）、help（帮助）、locate（定位）、more（更多）、mail（邮箱 10.0.8 及以上）。	N	“”
contentDesc	string	设置盲人模式来阅读文案。	N	“”

1.6.5.9. 显示右上角按钮

此接口用于显示标题栏右边按钮的属性。

showOptionMenu 接口的使用方法

```
AlipayJSBridge.call('showOptionMenu');
```

代码示例

请参考：设置右上角按钮 setOptionMenu。

1.6.5.10. 隐藏右上角按钮

此接口用于隐藏标题栏右边按钮的属性。

hideOptionMenu 接口的使用方法

```
AlipayJSBridge.call('hideOptionMenu');
```

代码示例

请参考：设置右上角按钮 setOptionMenu。

1.6.5.11. 显示加载中

此接口用于显示全局加载框。

showLoading 接口的使用方法

```
AlipayJSBridge.call('showLoading', {
  text: '加载中',
});
```

代码示例

显示/隐藏全局加载框：

```
<h1>点击以下按钮看不同效果</h1>
<p>注意安卓下显示 loading 后，会覆盖整个界面，所以请使用系统回退键关闭 loading</p>
<a href="javascript:void(0)" class="btn show">显示 loading</a>
<a href="javascript:void(0)" class="btn delay">延迟 2 秒显示 loading</a>
<a href="javascript:void(0)" class="btn notext">无文字菊花</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.show').addEventListener('click', function() {
    AlipayJSBridge.call('showLoading', {
      text: 'Loading',
    });
    setTimeout(function() {
      AlipayJSBridge.call('hideLoading');
    }, 3000);
  });

  document.querySelector('.delay').addEventListener('click', function() {
    AlipayJSBridge.call('showLoading', {
      text: 'Loading',
      delay: 2000,
    });
    setTimeout(function() {
      AlipayJSBridge.call('hideLoading');
    }, 5000);
  });

  document.querySelector('.notext').addEventListener('click', function() {
    AlipayJSBridge.call('showLoading', {
      text: ' ',
    });
    setTimeout(function() {
      AlipayJSBridge.call('hideLoading');
    }, 3000);
  });
});
</script>
```

API 说明

> **重要**
>
> - Android 下显示 loading 后，会覆盖整个界面，所以请使用系统返回键关闭 loading。
> - iOS 下的情况是，当没有设置 `text` 值的时候，只有标题栏和工具栏可以被点击，有文字的时候不能覆盖任何内容。9.9.5 版本以后已修复。
> - `showLoading` 是 WebView 级别的，因此在 `pushwindow` 后的 WebView 上调用 `hideLoading` 无法关掉上个 WebView 的 loading，需要保证 `showLoading` 和 `hideLoading` 在一个 WebView 环境内执行。

```
AlipayJSBridge.call('showLoading',{
  text, delay
})
```

入参

属性	类型	描述	必填	默认值
text	string	文本内容，如需设为无文案，需传入一个空格。	N	“加载中”
delay	int	延迟多少毫秒后显示，如果在此时间之前调用了 `hideLoading` ，则不会再显示。	N	0
autoHide	bool	默认情况下容器会在 `pageFinish` 后主动隐藏加载框，默认为 true，传入 false，关掉自动隐藏（仅限 Android 系统）。	N	true
cancelable	bool	安卓返回键是否消掉加载框，默认物理返回键会消掉加载框（仅限 Android 系统）。	N	true

1.6.5.12. 隐藏加载中

此接口用于隐藏全局加载框。

hideLoading 接口的使用方法

```
AlipayJSBridge.call('hideLoading');
```

代码示例

显示/隐藏全局加载框：

```
<h1>点击以下按钮看不同效果</h1>
<p>注意安卓下显示 loading 后，会覆盖整个界面，所以请使用系统回退键关闭 loading</p>
<button class="btn show">显示 loading</button>
<button class="btn delay">延迟 2 秒显示 loading</button>
<button class="btn notext">无文字菊花</button>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.show').addEventListener('click', function() {
    AlipayJSBridge.call('showLoading', {
      text: 'Loading',
    });
    setTimeout(function() {
      AlipayJSBridge.call('hideLoading');
    }, 3000);
  });

  document.querySelector('.delay').addEventListener('click', function() {
    AlipayJSBridge.call('showLoading', {
      text: 'Loading',
      delay: 2000,
    });
    setTimeout(function() {
      AlipayJSBridge.call('hideLoading');
    }, 5000);
  });

  document.querySelector('.notext').addEventListener('click', function() {
    AlipayJSBridge.call('showLoading', {
      text: ' ',
    });
    setTimeout(function() {
      AlipayJSBridge.call('hideLoading');
    }, 3000);
  });
});
</script>
```

1.6.5.13. 显示标题栏加载中

此接口用于在标题栏显示加载框。

showTitleLoading 接口的使用方法

```
AlipayJSBridge.call('showTitleLoading');
```

代码示例

显示/隐藏全局加载框：

```
<h1>点击下面按钮查看效果</h1>
<a href="javascript:void(0)" class="btn show">显示 loading</a>
<a href="javascript:void(0)" class="btn hide">隐藏 loading</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.show').addEventListener('click', function() {
    AlipayJSBridge.call('showTitleLoading');
  });

  document.querySelector('.hide').addEventListener('click', function() {
    AlipayJSBridge.call('hideTitleLoading');
  });
});
</script>
```

1.6.5.14. 隐藏标题栏加载中

此接口用于隐藏标题栏加载框。

hideTitleLoading 接口的使用方法

```
AlipayJSBridge.call('hideTitleLoading');
```

代码示例

显示/隐藏全局加载框：

```
<h1>单击下面按钮查看效果</h1>
<a href="javascript:void(0)" class="btn show">显示 loading</a>
<a href="javascript:void(0)" class="btn hide">隐藏 loading</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.show').addEventListener('click', function() {
    AlipayJSBridge.call('showTitleLoading');
  });

  document.querySelector('.hide').addEventListener('click', function() {
    AlipayJSBridge.call('hideTitleLoading');
  });
});
</script>
```

1.6.6. 工具类

1.6.6.1. 获取容器的启动参数

此接口仅用于获取打开离线包时传递的启动参数，`pushWindow` 传递的参数请使用 `AlipayJsBridge.startupParams` 获取。

getStartupParams 接口的使用方法

```
AlipayJSBridge.call('getStartupParams', {
  key: ['url', 'xxx'] // 可选，根据 key 值过滤返回结果，不填返回全部
}, function(result) {
  console.log(result);
});
```

代码示例

```
<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}

ready(function() {
  // 有 key 参数的情况
  AlipayJSBridge.call('getStartupParams', {
    key: ['url']
  }, function(result) {
    alert(JSON.stringify(result));
  });

  // 没有参数的情况
  AlipayJSBridge.call('getStartupParams', function(result) {
    alert(JSON.stringify(result));
  });
});
</script>
```

API 说明

```
getStartupParams
```

入参

属性	类型	描述	必填	默认值
key	Array	根据传的 key 来获得对应 key 的值。	N	null

出参

返回对应的启动参数，例如：`{url: 'https://taobao.com', xx: '其它启动参数'}`。

- 如果没有输入参数，则返回所有的 `startupParams` 参数。

- 如果有入参，根据入参返回对应的值。
- 如果启动参数中没有对应 key 值，则返回中不带这个 key，不作为报错处理。

错误码描述

错误码	描述
2	参数异常，key 为空数组，或者其它类型。
12	未知错误。

1.6.6.2. 截屏

此接口用于截屏。

snapshot 的使用方法

```
AlipayJSBridge.call('snapshot', function(result) {
  console.log(result.success);
});
```

代码示例

```
<h1>点击下面按钮查看不同截图效果</h1>

<a href="javascript:void(0)" class="btn screen">截取屏幕并保存到相册</a>
<a href="javascript:void(0)" class="btn viewport">viewport 截图返回 fileURL</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.screen').addEventListener('click', function() {
    AlipayJSBridge.call('snapshot', function(result) {
      alert(JSON.stringify(result));
    });
  });

  document.querySelector('.viewport').addEventListener('click', function() {
    AlipayJSBridge.call('snapshot', {
      range: 'viewport',
      dataType: 'fileURL',
      saveToGallery: false
    }, function(result) {
      alert(JSON.stringify(result));
    });
  });
});
</script>
```

API 说明

```
AlipayJSBridge.call('snapshot', {
  range, saveToGallery, dataType, imageFormat, quality,
  maxWidth, maxHeight
}, fn)
```

入参

属性	类型	描述	必填	默认值
range	string	快照范围： • screen：当前客户端整个屏幕。 • viewport：网页可见区域。 • document：整个网页。 重要 document 会截取全部网页，在安卓手机上，网页很多时候会存在浏览器内存溢出情况，请使用 `screen` 参数。	N	“screen”
saveToGallery	bool	是否保存到相册。	N	true
dataType	string	结果数据格式： • dataURL：base64 编码的图片数据。 • fileURL：图片在文件系统中的 URL（图片存放于临时目录中，退出时被清除）。 • none：不返回数据（用于保存到相册的情况）。	N	“none”
imageFormat	string	jpg、png。	N	“jpg”
quality	int	jpg 的图片质量，取值为 1 到 100。	N	75
maxWidth	int	图片的最大宽度，过大将被等比缩小。	N	-
maxHeight	int	图片的最大高度，过大将被等比缩小。	N	-
fn	function	回调函数。	N	-

出参

回调函数带入的参数 `result: {success, fileUrl, dataURL}` 。

属性	类型	描述
success	bool	是否处理成功。
fileUrl	string	图片在文件系统中的 URL。
dataURL	string	base64 编码的图片数据。

错误码

错误码	描述
10	相册保存失败。
11	图片文件保存失败。

1.6.6.3. RPC 调用

本文介绍的是 RPC 接口的使用方法。

> **说明**
>
> 由于 JS 传入的 JSON 数据无法包含数据类型，在 Native 层转为字典时可能会由于数据类型问题导致误差，如果是数字类型的精确值，尽量使用字符串来进行传递。例如：`{"value":9.45}` 会被 native 转为 `{"value":9.449999999999999}` 然后上发到服务端。应该改为使用 `{"value":"9.45"}` 来传递。

RPC 接口的使用方法

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

代码示例

```
<h1>点击按钮发起 RPC 请求</h1>

<a href="javascript:void(0)" class="btn rpc">发起请求</a><br/>
<a href="javascript:void(0)" class="btn rpcHeader">发起有响应头返回的请求</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.rpc').addEventListener('click', function() {
    AlipayJSBridge.call('rpc', {
      operationType: 'alipay.client.xxxx',
      requestData: [],
      headers: {}
    }, function(result) {
      alert(JSON.stringify(result));
    });
  });

  document.querySelector('.rpcHeader').addEventListener('click', function() {
    AlipayJSBridge.call('rpc', {
      operationType: 'alipay.client.xxxx',
      requestData: [],
      headers: {},
      getResponse: true
    }, function(result) {
      alert(JSON.stringify(result));
    });
  });
});
</script>
```

API 说明

```
AlipayJSBridge.call('rpc', {
  operationType:,
  requestData:,
  headers
}, fn);
```

入参

属性	类型	描述	必填	默认值
operationType	string	RPC 服务名称。	Y	-
requestData	array	RPC 请求的参数。需要开发者根据具体 RPC 接口进行构造。	N	-
headers	object	RPC 请求设置的 headers。	N	{}

gateway	string	网关地址。	N	alipay 网关
compress	boolean	是否支持 request gzip 压缩。	N	true
disableLimitView	boolean	RPC 网关被限流时是否禁止自动弹出统一限流弹窗。	N	false
timeout	int	RPC 超时时间，单位为秒。 框架统一设置，策略较复杂。 • iOS 端 Wi-Fi 环境 20s，其它环境 30s。 • Android 端 Wi-Fi/4G 环境 12s 到 42s 之间。其它环境 32s 到 60s 之间	N	-
getResponse	boolean	获取 RPC 响应头。 **重要** 设置为 true 时，响应数据会多一层嵌套，可用于数据回流上报获取 traceId、entityId。	N	false
fn	function	回调函数。	N	-

出参

回调函数带入的参数 `result: {error }` 。

属性	类型	描述
error	string	错误码

错误码

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

RPC 原生错误码

错误码	描述
1000	成功。
0	未知错误。

1	客户端找不到通讯对象。
2	客户端没有网络（JSAPI 做了转换，返回 10）。
3	客户端证书错误。
4	客户端网络连接超时。
5	客户端网络速度过慢。
6	客户端请求服务端未返回。
7	客户端网络 IO 错误。
8	客户端网络请求调度错误。
9	客户端处理错误。
10	客户端数据反序列化错误，服务端数据格式有误。
11	客户端登录失败。
12	客户端登录账号切换。
13	请求中断错误，例如线程中断时网络请求会被中断。
14	客户端网络缓存错误。
15	客户端网络授权错误。
16	DNS 解析错误。
17	operationType 不在白名单。
1001	拒绝访问。
1002	调用次数超过限制：“系统繁忙，请稍后再试。”
2000	登录超时，请重新登录。
3000	缺少操作类型或者此操作类型不支持。
3001	请求数据为空：系统繁忙，请稍后再试。

3002	数据格式有误。
4001	服务请求超时，请稍后再试。
4002	远程调用业务系统异常：网络繁忙，请稍后再试。
4003	创建远程调用代理失败：网络繁忙，请稍后再试。
5000	未知错误：“抱歉，暂时无法操作，请稍后再试。”
6000	RPC-服务找不到。
6001	RPC-目标方法找不到。
6002	RPC-参数数目不正确。
6003	RPC-目标方法不可访问。
6004	RPC-JSON 解析异常。
6005	RPC-调用目标方法时参数不合法。
6666	RPC-业务异常。
7000	没有设置公钥。
7001	验签的参数不够。
7002	验签失败。
7003	验签时间戳校验失败。
7004	验签 RPC 接口 operationType 参数为空。
7005	productId 参数为空。
7006	验签接口 did 参数为空。
7007	验签接口请求发送时间参数 t 为空。
7008	验签接口 IMEI（客户端设备标识）参数为空。
7009	验签接口 IMSI（客户端用户标识）为空。
7010	验签接口 API 版本号为空。

7011	验签接口用户没有权限。
7012	验签接口 RPC 没有对外开放。
7013	验签接口 productId 没有注册或者获取密钥为空。
7014	验签接口加签数据为空。
7015	验签接口签约无效。
7016	验签接口请求登录 RPC 传入 sid 为空。
7017	验签接口请求登录 RPC 传入 sid 无效。
7018	验签接口请求登录 RPC 传入 token 无效。
7019	验签接口请求登录 RPC 获取 alipayuserid 为空。
8001	etag：响应数据没有变化。

RPC 自定义 gateway

可在 RPC 调用中指定请求的网关地址。

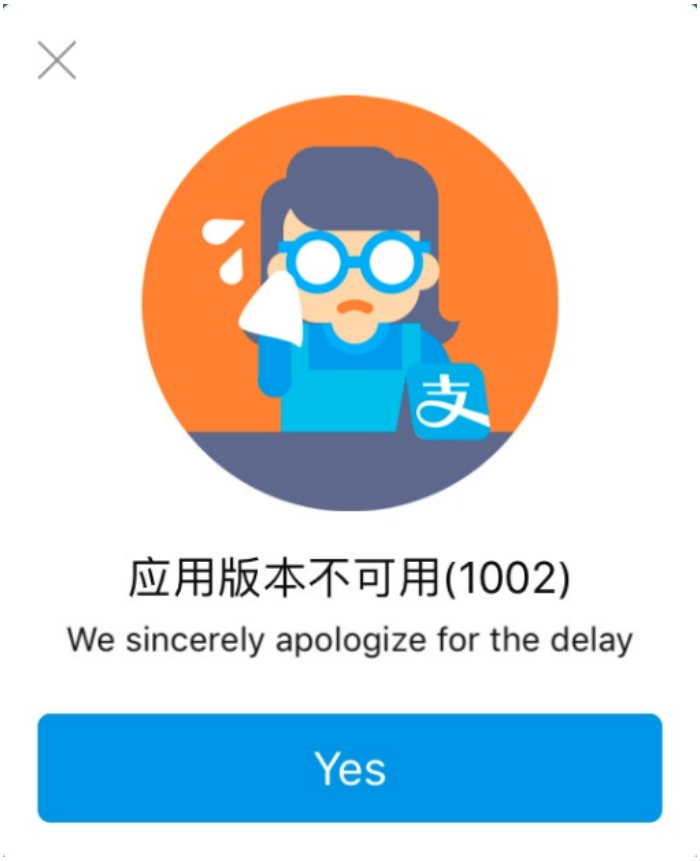
RPC 限流逻辑

容器版本	disableLimitView	行为	回调参数
<=9.9.5	true	静默	1002
<=9.9.5	false	Alert	1002
>=9.9.6	true	静默	1002
>=9.9.6	false	网关处理	100201

行为类型	描述
静默	无。
Alert	弹出统一限流框，如下图。
Toast	弹出系统 Toast，如果用户关闭系统则没有。

网关处理	根据网关的 RPC 配置，静默 Alert Toast。

RPC 限流弹框



常见问题

Q：出现如下报错信息：`ESLint: 'AlipayJSBridge' is not defined`，如何解决？

A：AlipayJSBridge 未定义的问题有如下两种解决方案：

- 方案一：`window.AlipayJSBridge.call('rpc');`
- 方案二：

```
const { AlipayJSBridge } = window;
AlipayJSBridge.call('rpc');
```

1.6.6.4. 上报埋点

此接口为前端能用到的最原始的埋点接口。

remoteLog 接口的使用方法

```
AlipayJSBridge.call('remoteLog', {
  bizType: "Nebula",     // 业务类型
  logLevel: 1,           // 1 - high, 2 - medium, 3 - low
  actionId: "event",     // 埋点类型，固定为 "event"
  seedId: "Login",       // 埋点唯一标识
  param1: "",
  param2: "",
  param3: "",
  param4: {key1:"value1",key2:"value2"}, // 自定义参数
});
```

> **说明**
>
> - 如果您需要添加自定义埋点参数，可以通过 `key:"value"` 的格式添加至上述代码的 `param4` 中，例如： `key1:"value1"` 。
> - 添加多个自定义埋点参数时，在 `param4` 中添加的内容格式如下： `param4:"key1:"value1",key2:"value2",key3:"value3"` 。

代码示例

```
<h1>点击按钮会记录相关信息</h1>

<a href="javascript:void(0)" class="btn read">点一点</a>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('.read').addEventListener('click', function() {
    AlipayJSBridge.call('remoteLog', {
      type: "behavior",
      bizType: "Nebula",
      spmId: "a1.b2",
      logLevel: 1, // 1 - high, 2 - medium, 3 - low
      actionId: "event"
      seedId: "xxx",
      param1: "xxx",
      param2: "xxx",
      param3: "xxx",
      param4: "xxx",
    });
  });
});
</script>
```

API 说明

```
AlipayJSBridge.call('remoteLog', {
  type, seedId, ucId, bizType, logLevel,
  actionId, spmId, param1, param2, param3, param4
});
```

入参

属性	类型	描述	必填	默认值

type	string	埋点类型： • monitor：监控类型。 • monitorWithLocation：监控类型，自动在 param4 里带上经纬度。 • behavior：行为类型。 • behaviorAuto：自动行为类型。 • performance：性能类型。 • error：异常类型，9.6.8 版本开始支持。 • 135：135 业务相关，9.9 版本开始支持。	N	“monitor”
seedId	string	埋点 ID。	Y	“”
bizType	float	业务类型标识，该参数传值时，会生成单独的日志文件。	N	-
logLevel	int	1 - high 2 - medium 3 - low，低级别的日志可能会被限流。	N	-
actionId	string	埋点类型，固定为 event。	Y	“”
spmId	string	spm 编码。当编有 spmId 时，忽略 seedId。	Y	“”
param1	string	埋点参数 1。	N	“”
param2	string	埋点参数 2。	N	“”
param3	string	埋点参数 3。	N	“”
param4	string	埋点参数 4。	N	“”

1.6.6.5. 设置 AP 数据

`setAPDataStorage` 接口用于将一个字符串保存到客户端统一存储，字符串长度不得超过 200×1024。

> **说明**
>
> - 底层存储服务组件在 iOS 和 Android 中实现不一致。Android 统一存储组件不支持 `type=user` 属性，为了与前端接口一致，当设置 `type=user` 时，Android 底层会设置为 `key=key + "_" +MD5(userId + userId + userId)` 并进行存储。业务用客户端取的时候也要对 key 做相应处理。
> - 在 10.1.60 及以下版本的基线中，客户端需要做适配工作，以使接口能够获得 userId，否则存储接口将无法按 `userId` 区分存储，参见下方 实现 H5LoginProvider 接口。
> - 在 10.1.68 及以上版本的基线中，`userId` 默认使用 `MPLogger.setUserId` 中的值，若实现 `H5LoginProvider`，则取用 `H5LoginProvider`。

实现 H5LoginProvider 接口

Android

实现 `H5LoginProvider` 接口，并将实例类设置到 `H5ProviderManager` 中。

代码示例

```
package com.mpaas.nebula.provider;

import android.os.Bundle;

import com.alipay.mobile.common.logging.api.LoggerFactory;
import com.alipay.mobile.nebula.provider.H5LoginProvider;

public class H5LoginProviderImpl implements H5LoginProvider {
    // 其他代码省略

    @Override
    public String getUserId() {
        // 此方法返回 userId 即可
        return LoggerFactory.getLogContext().getUserId();
    }

    // 其他代码省略
}
```

设置 H5LoginProvider

```
H5Utils.setProvider(H5LoginProvider.class.getName(), new H5LoginProviderImpl());
```

setAPDataStorage 接口的使用方法

```
AlipayJSBridge.call('setAPDataStorage', {
  type: "common",
  business: "customBusinessKey",
  key: "customKey",
  value: "customValue"
}, function(result) {
  alert(JSON.stringify(result));
});
```

代码示例

```
<button id="J_saveDataBtn" class="btn">保存数据</button>
<button id="J_getDataBtn" class="btn">查看数据</button>
<button id="J_removeDataBtn" class="btn">删除数据</button>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('#J_saveDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('setAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey",
      value: "customValue"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);

  document.querySelector('#J_getDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('getAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);

  document.querySelector('#J_removeDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('removeAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);
}, false);
</script>
```

API 说明

```
AlipayJSBridge.call('setAPDataStorage', {
  type, business, key, value
});
```

入参

属性	类型	描述	必填	默认值
type	String	用户维度存储（ `user` ）或公共存储（ `common` ）。	N	“common”

business	String	自定义的业务标识，可与相应的客户端存取代码约定。 在 Android 中，该业务标识与创建 APSharedPreferences 时所传入的 `GROUP_ID` 对应。	N	“NebulaBiz”
key	String	自定义数据的 key。	Y	“”
value	String	需要存储的值，仅支持字符串类型。JSON 数据需要先字符串化。	Y	“”

出参

回调函数带入的参数 `result: {success}` 。

属性	类型	描述
success	bool	是否保存成功。

错误码

错误码	描述
11	字符串长度超出限制。

1.6.6.6. 获取 AP 数据

此接口用于从统一存储中获取数据，仅支持字符串类型。

> **说明**
> 如果在 iOS 端获取的数据是由 native 端直接存储的，确保使用 setString 接口存储，否则此接口获取数据会失败。

getAPDataStorage 接口的使用方法

```
AlipayJSBridge.call('getAPDataStorage', {
  type: "common",
  business: "customBusinessKey",
  key: "customKey",
}, function(result) {
  alert(JSON.stringify(result));
});
```

代码示例

```
<button id="J_saveDataBtn" class="btn">保存数据</button>
<button id="J_getDataBtn" class="btn">查看数据</button>
<button id="J_removeDataBtn" class="btn">删除数据</button>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('#J_saveDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('setAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey",
      value: "customValue"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);

  document.querySelector('#J_getDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('getAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);

  document.querySelector('#J_removeDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('removeAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);
}, false);
</script>
```

API 说明

```
AlipayJSBridge.call('getAPDataStorage', {
  type, business, key
});
```

入参

属性	类型	描述	必填	默认值

type	string	用户维度存储（ `user` ）或公共存储（ `common` ），默认值为 `common` 。	N	“common”
business	string	自定义的业务标识，可与相应的客户端存取代码约定，默认值为 `NebulaBiz` 。 在 Android 中，该业务标识与创建 APSharedPreferences 时所传入的 `GROUD_ID` 对应。	N	“”
key	string	自定义数据的 key。	Y	“”

出参

回调函数带入的参数 `result: {data}` 。

属性	类型	描述
data	string	数据。
errorMessage	string	未找到该数据。

错误码

错误码	描述
11	未找到该数据。

1.6.6.7. 移除 AP 数据

此接口用于从统一存储中删除数据。

removeAPDataStorage 接口的使用方法

```
AlipayJSBridge.call('removeAPDataStorage', {
  type: "common",
  business: "customBusinessKey",
  key: "customKey",
}, function(result) {
  alert(JSON.stringify(result));
});
```

代码示例

```
<button id="J_saveDataBtn" class="btn">保存数据</button>
<button id="J_getDataBtn" class="btn">查看数据</button>
<button id="J_removeDataBtn" class="btn">删除数据</button>

<script>
function ready(callback) {
  // 如果 jsbridge 已经注入则直接调用
  if (window.AlipayJSBridge) {
    callback && callback();
  } else {
    // 如果没有注入则监听注入的事件
    document.addEventListener('AlipayJSBridgeReady', callback, false);
  }
}
ready(function() {
  document.querySelector('#J_saveDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('setAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey",
      value: "customValue"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);

  document.querySelector('#J_getDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('getAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);

  document.querySelector('#J_removeDataBtn').addEventListener('click', function(e) {
    AlipayJSBridge.call('removeAPDataStorage', {
      type: "common",
      business: "customBusinessKey",
      key: "customKey"
    }, function(result) {
      alert(JSON.stringify(result));
    });
  }, false);
}, false);
</script>
```

API 说明

```
AlipayJSBridge.call('removeAPDataStorage', {
  type, business, key
});
```

入参

属性	类型	描述	必填	默认值

type	string	用户维度存储（ `user` ）或公共存储（ `common` ），默认值为 `common` 。	N	“common”
business	string	自定义的业务标识，可与相应的客户端存取代码约定， 默认值为 `NebulaBiz` 。 在 Android 中，该业务标识与创建 APSharedPreferences 时所传入的 `GROUD_ID` 对应。	N	“”
key	string	自定义数据的 key。	Y	“”

出参

回调函数带入的参数 `result: {success}` 。

属性	类型	描述
success	bool	删除是否成功。

1.7. 使用教程

1.7.1. Android 使用教程 - AAR

1.7.1.1. 总览

H5 容器支持原生 AAR 接入和组件化接入两种接入方式。

如果想要像使用其他 SDK 一样简单地接入并使用 mPaaS，推荐使用原生 AAR 接入方式。原生 AAR 接入方式是指采用原生 Android AAR 打包方案，更贴近 Android 开发者的技术栈。开发者无需了解 mPaaS 相关的打包知识，通过 mPaaS Android Studio 插件即可将 mPaaS 集成到开发者的项目中。该方式降低了开发者的接入成本，能够让开发者更轻松地使用 mPaaS。

为了方便您快速熟悉并掌握原生 AAR 接入方式，本教程以 **原生 AAR 接入方式** 为例，指导您快速接入 H5 容器组件并使用 H5 离线包。

本教程一共包含以下五个部分：

1. 在 Android Studio 创建应用。
2. 在 mPaaS 控制台创建应用。
3. 原生 AAR 方式接入工程。
4. 使用 H5 容器。
5. 使用 H5 离线包。

您将学会

- 如何创建一个通过点击按钮弹出 Toast 的安卓应用。
- 如何采用原生 AAR 方式接入 mPaaS。
- 如何使用 mPaaS H5 容器服务。
- 如何使用 mPaaS H5 离线包服务。

您将需要

1. 配置开发环境（本教程中 Windows 下的开发环境为例进行说明）。
2. 网络浏览器（建议您使用 Chrome 浏览器）。
3. 一部安卓手机（系统版本为安卓 4.3 或更新版本）及配套的数据线。您也可以选择使用模拟器进行调试，本教程以模拟器为例。

1.7.1.2. 在 Android Studio 创建应用

在本节您将创建一个通过点击按钮弹出 Toast 的应用，并获得 APK 格式的安装包。

该过程主要分为四个步骤：

1. 创建工程
2. 编写代码
3. 创建签名文件并给工程添加签名
4. 在手机上安装应用

如果您已经有了一个原生的 Android 开发工程并完成了签名，那么您可以跳过本教程，直接 在 mPaaS 控制台创建应用。

创建工程

1. 打开 Android Studio，点击 **File** > **New** > **New Project**。
2. 在弹出的新建工程窗口中，选择 **Empty Activity**，点击 **Next**。
3. 输入 **Name**、**Package name**（可以使用默认值）、**Save location**。在此处 **Name** 以 **H5 Application** 为例。选择 **Minimum SDK** 为 **API 18: Android 4.3 (Jelly Bean)**。

   > **说明**
   > API 18: Android 4.3（Jelly Bean）是 mPaaS 支持的最低版本，您在实际生产中可以根据需要进行选择。

4. 点击 **Finish**，即可完成 **创建工程**。

编写代码

1. 打开 `activity_main.xml` 文件，参照如下代码添加按钮。

```
<Button
    android:id="@+id/button"
    android:layout_width="101dp"
    android:layout_height="50dp"
    android:layout_marginStart="142dp"
    android:layout_marginTop="153dp"
    android:layout_marginBottom="151dp"
    android:text="Button"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

2. 打开 `MainActivity` 类，添加按钮的点击事件。

```
    findViewById(R.id.button).setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            Toast.makeText(MainActivity.this, "Hello mPaaS!", Toast.LENGTH_SHORT).show();
        }
    });
```

3. 编译成功后，您已完成 **编写代码**。

创建签名文件并给工程添加签名

1. 在 Android Studio 中点击 **Build** > **Generate Signed Bundle / APK**。
2. 在弹出的窗口中选择 **APK**，点击 **Next**。
3. 选择 **Create new**。
4. 填入相应信息后，点击 **OK**，即可完成创建签名。您可在指定的 **Key store path** 中获得生成的签名文件。
5. 内容自动填充后，点击 **Next** 开始对工程添加签名。
6. 根据需要选择 **Build Variants**，随后勾选 **V1（Jar Signature）** 加密版本。V1（Jar Signature）为必选项，V2（Full APK Signature）可按需选择。
7. 点击 **Finish**。打包完成后在工程文件夹下的 `debug` 文件夹（`~\MyHApplication\app\debug`）中，即可获得该应用签名后的 APK 安装包。在本教程中，安装包名为 `app-debug.apk`。

在手机上安装应用

1. 连接手机到电脑，并开启手机的 USB 调试模式。
2. 运行工程。
3. 点击 **BUTTON**，弹出 Toast，即表示应用安装成功且实现了预期功能。至此，您已完成 **在手机上安装应用** 。

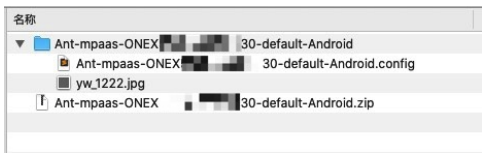
1.7.1.3. 在 mPaaS 控制台创建应用

本文介绍的是在 mPaaS 控制台创建应用的操作步骤。

1. 打开网络浏览器，登录 mPaaS 控制台 。
2. 创建 mPaaS 应用。
3. 输入项目名并单击 **确定**。
4. 在 **应用列表** 页，单击 **H5 Application（应用名称）**。
5. 在 **应用详情** 页，单击 

，打开 **配置应用** 页。
6. 在 **配置应用** 页，单击 

，打开 **代码配置** 页。在 **代码配置** 页，输入 **Package Name（应用包名）**（此处以 **com.example.h5application** 为例），上传编译并添加签名后的 APK 安装包。关于快速生成签名后的 APK 相关信息，请参见 生成控制台用签名 APK。在 **代码配置** 页，填写完成后，点击 **下载配置**，即可获取 mPaaS 的配置文件。配置文件是一个压缩包文件。该压缩包包含一个 `.config` 文件以及一个 `yw_1222.jpg` 加密图片。



1.7.1.4. 原生 AAR 方式接入工程

本文介绍如何通过原生 AAR 方式接入工程。

1. 在 Android Studio 中选择 **mPaaS** > **原生 AAR 接入**。
2. 在界面右侧弹出的窗口中，选择 **导入 App 配置** 下方的 **开始导入**。
3. 在弹出的 **导入 mPaaS 配置文件** 窗口中，选择 **我已经从控制台上下载配置文件，准备导入到工程** ，点击 **Next**。
4. 选择在控制台创建 mPaaS 应用后下载的 配置文件，点击 **Finish**。
5. 随后会提示配置文件导入成功。
6. 点击界面右侧 **接入/升级基线** 下方的 **开始配置**。
7. 在弹出的 **选择 mPaaS 基线版本** 窗口中，选择 10.1.68 基线，点击 **OK**，即可接入 mPaaS SDK。

> **说明**
> 再次点击 **开始配置** 可升级基线。

8. 点击界面右侧 **配置/更新组件** 下方的 **开始配置**。
9. 在弹出的组件列表中，勾选 **H5 容器** ，并点击 **OK**，即可将 H5 容器组件添加至工程。至此您已完成通过原生 AAR 方式接入工程。

1.7.1.5. 使用 H5 容器

您可使用 H5 容器完成以下操作。

- 在应用内打开一个在线网页
- 前端调用 Native 接口
- 前端调用自定义 JSAPI
- 自定义 H5 页面的 TitleBar

在应用内打开一个在线网页

1. 给工程添加自定义的类 `MyApplication` ，该类继承自 `Application` 。
2. 在自定义的 `MyApplication` 类中进行初始化，初始化方法如下：

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

    MP.init(this);
}
```

详情可参考：初始化 mPaaS。

3. 在 `app/src/main/AndroidManifest.xml` 文件中，添加 `android:name=".MyApplication"` 。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.h5application">

    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

4. 在 `activity_main.xml` 文件中，重新设置 Button 样式并修改 Button 的 id 为 `start_url_btn` 。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/start_url_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:background="#108EE9"
        android:gravity="center"
        android:text="启动一个在线页面"
        android:textColor="#ffffff"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

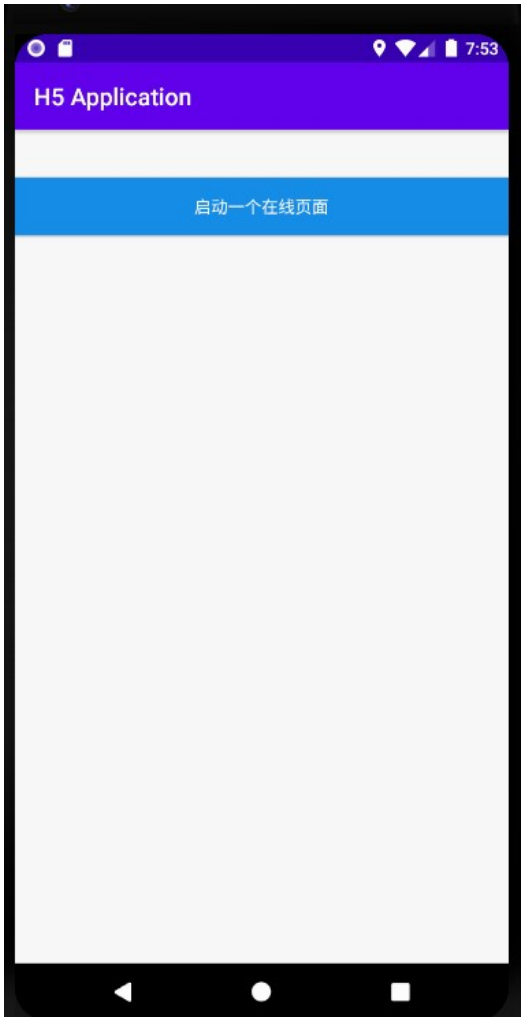
5. 在 `MainActivity` 类重写单击按钮事件，实现打开蚂蚁科技官网的功能。实现代码如下所示：

```
        findViewById(R.id.start_url_btn).setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                MPNebula.startUrl("https://tech.antfin.com/");
            }
        });
```

6. 在工程主 Module 下的 `build.gradle(:app)` 中添加以下配置：

```
apply plugin: 'com.android.application'
apply plugin: 'com.alipay.apollo.baseline.config'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "com.example.h5application"
        minSdkVersion 18
        targetSdkVersion 26
        ndk{
            abiFilters 'armeabi'
        }
        multiDexEnabled true
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation platform("com.mpaas.android:$mpaas_artifact:$mpaas_baseline")
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.mpaas.android:nebula'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

}
```

7. 编译工程后，在手机上安装应用。打开应用后界面如下：



8. 单击按钮后即可应用内打开金融科技官网首页，即说明接口调用成功。至此，您已完成 **在应用内打开一个在线网页** 。

前端调用 Native 接口

1. 在开发前端页面时，可以通过 Nebula 容器提供的 bridge，通过 JSAPI 的方式与 Native 进行通信，获取 Native 处理的相关信息或数据。Nebula 容器内预置了部分基础的 JSAPI 能力，您可以在 H5 页面的 js 文件中，直接通过 `AlipayJSBridge.call` 的方式进行调用。示例如下：

```
AlipayJSBridge.call('alert', {
  title: '原生 Alert Dialog',
  message: '这是一个来自原生的 Alert Dialog',
  button: '确定'
}, function (e) {
  alert("单击了按钮");
});
```

> **说明**
>
> `https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-default/20200121/0.0.0.8_all/nebula/fallback/mPaaSComponentTestWebview.html` 是已经写好的前端页面，您可以调用此页面以体验前端调用 Native 接口的功能。

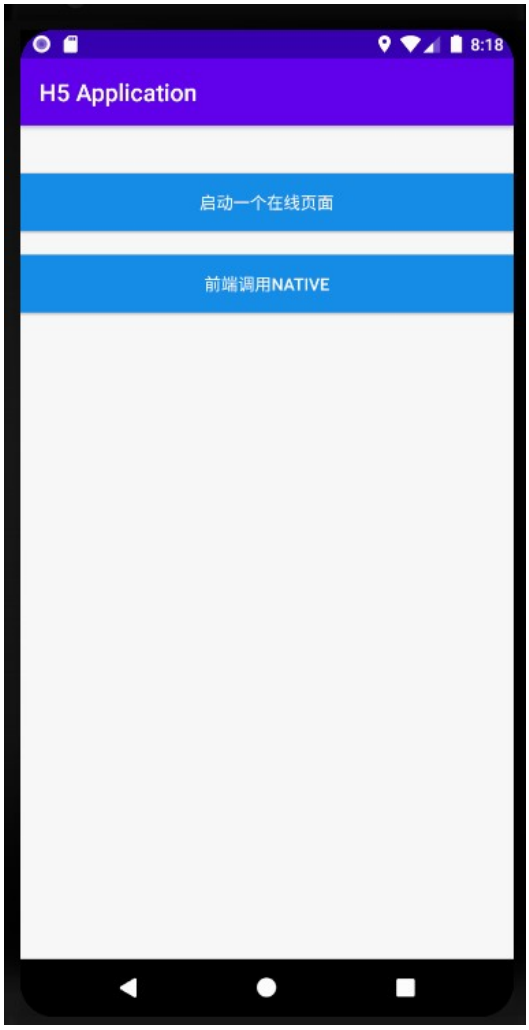
2. 在 `activity_main.xml` 文件中，新增按钮 Button，Button id 设置为 `h5_to_native_btn` 。

```
<Button
    android:id="@+id/h5_to_native_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="前端调用Native"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/start_url_btn" />
```

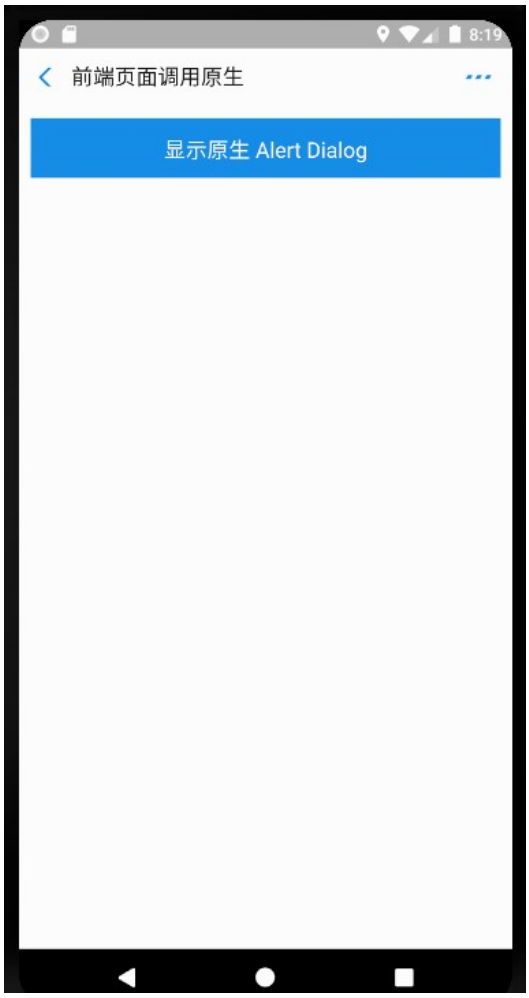
3. 在 `MainActivity` 类定义单击按钮 `h5_to_native_btn` 后的行为，实现打开前端页面的功能。实现代码如下所示：

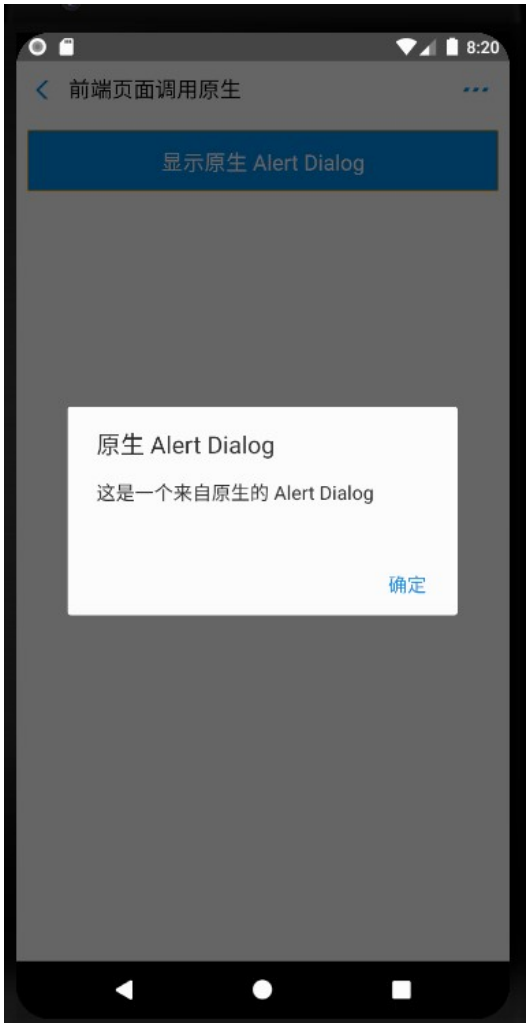
```
    findViewById(R.id.h5_to_native_btn).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            MPNebula.startUrl("https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/80000000/1.0.XX.XX_all/nebula/fallback/h5_to_native.html");
        }
    });
```

4. 编译工程后，在手机上安装应用。打开应用后界面如下：



5. 单击按钮后即可打开前端页面，单击按钮 **显示原生 Alert Dialog**，会弹出原生的警示框，警示框的标题是 **原生 Alert Dialog**，消息框的内容是**这是一个来自原生的 Alert Dialog**；单击警示框的 **确定** 按钮，会再弹出一个无标题警示框，内容是 **点击了按钮**。说明接口调用成功。至此，您已完成 **前端调用 Native 接口** 。




8:20
前端页面调用原生
显示原生 Alert Dialog
原生 Alert Dialog
这是一个来自原生的 Alert Dialog
确定

前端调用自定义 JSAPI

1. 构建一个自定义类 `MyJSApiPlugin` ，用来定义自定义的 JSAPI。

```
package com.example.h5application;

import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5SimplePlugin;

public class MyJSApiPlugin extends H5SimplePlugin {
    private static final String API = "myapi";
    @Override
    public void onPrepare(H5EventFilter filter) {
        super.onPrepare(filter);
        filter.addAction(API);
    }
    @Override
    public boolean handleEvent(H5Event event, H5BridgeContext context) {
        String action = event.getAction();
        if (API.equalsIgnoreCase(action)) {
            JSONObject params = event.getParam();
            String param1 = params.getString("param1");
            String param2 = params.getString("param2");
            JSONObject result = new JSONObject();
            result.put("success", true);
            result.put("message", API + " with " + param1 + "," + param2 + " was handled by native."
);
            context.sendBridgeResult(result);
            return true;
        }
        return false;
    }
}
```

2. 在工程中注册自定义的 JSAPI：`MyJSApiPlugin`。推荐在应用启动的时候注册。此处我们注册在 `MyApplication` 中。

```
public class MyApplication extends Application {
        @Override
        protected void attachBaseContext(Context base) {
            super.attachBaseContext(base);
            // 建议判断下是否主进程，只在主进程初始化
            QuinoxlessFramework.setup(this, new IInitCallback() {
                @Override
                public void onPostInit() {
                    // 在这里开始使用 mPaaS 功能
        //调用registerCustomJsapi()完成自定义JSAPI的注册。
                    registerCustomJsapi();
                }
            });
        }

    @Override
    public void onCreate() {
        super.onCreate();
        QuinoxlessFramework.init();
    }

    private void registerCustomJsapi(){
        MPNebula.registerH5Plugin(
                // 插件的 class name
                MyJSApiPlugin.class.getName(),
                // 填空即可
                "",
                // 作用范围，填 "page" 即可
                "page",
                // 注册的 jsapi 名称
                new String[]{"myapi"});
    }
}
```

3. 在前端页面中，调用该自定义 JSAPI。示例如下：

```
AlipayJSBridge.call('myapi', {
  param1: 'JsParam1',
  param2: 'JsParam2'
}, function (result) {
  alert(JSON.stringify(result));
});
```

> **说明**
>
> `https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-default/80000001/1.0.XX.XX_all/nebula/fallback/custom_jsapi.html` 是已经写好的前端页面，您可以调用此页面以体验前端调用 **自定义 JSAPI** 接口的功能。

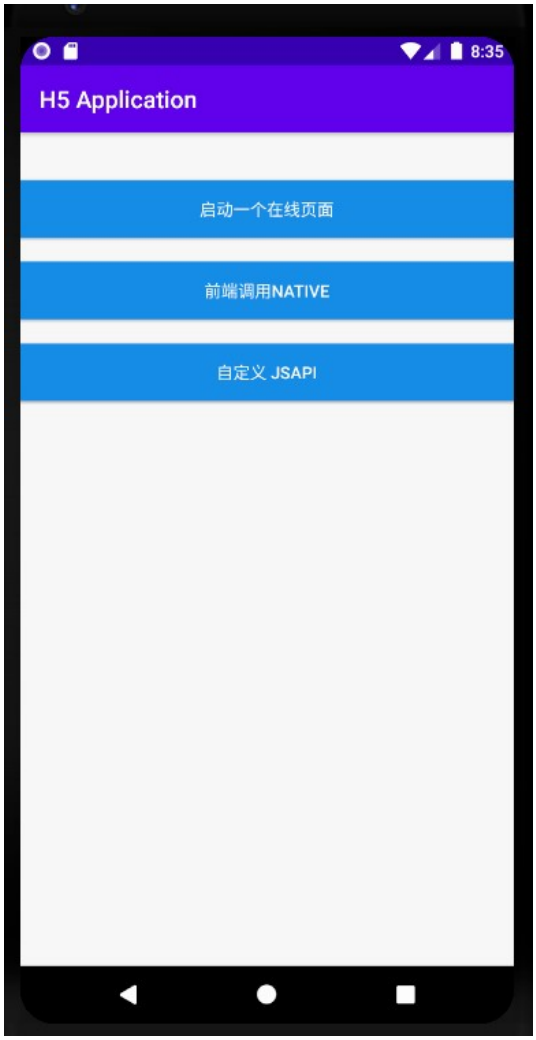
4. 在 `activity_main.xml` 文件中，新增按钮 button，`button id` 为 `custom_jsapi_btn`。

```
<Button
    android:id="@+id/custom_jsapi_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="自定义 JSAPI"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/h5_to_native_btn" />
```

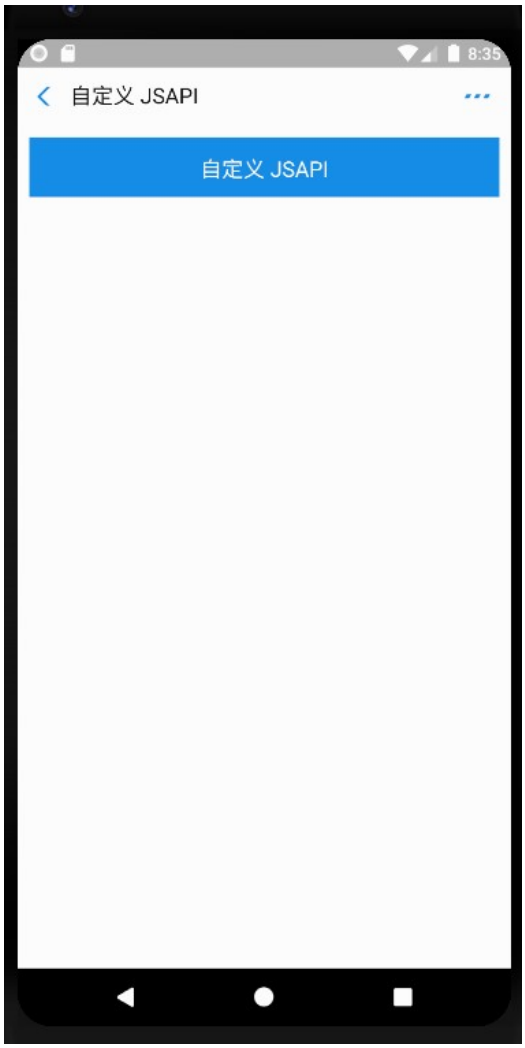
5. 在 `MainActivity` 类定义单击按钮 `custom_jsapi_btn` 后的行为，实现前端调用自定义 JSAPI 接口的功能。实现代码如下所示：

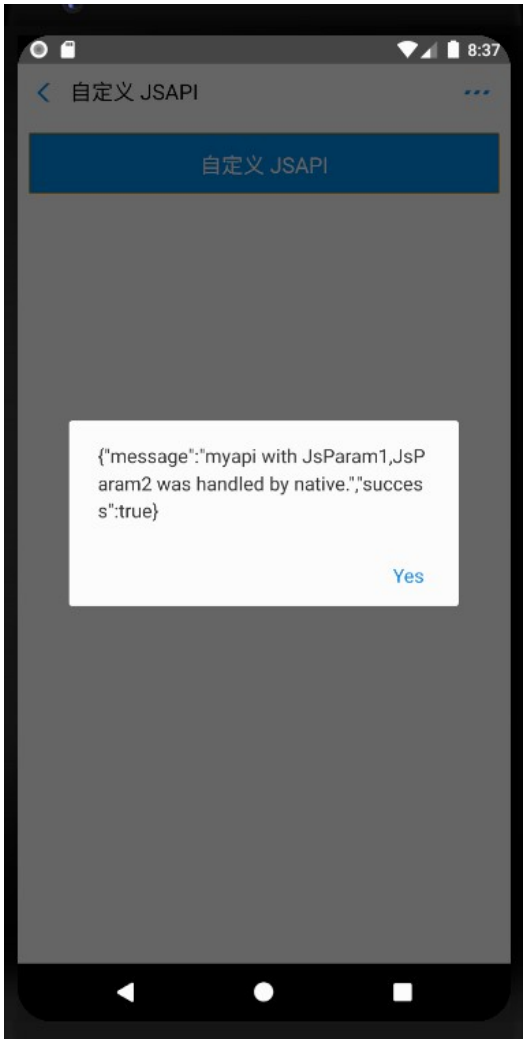
```
findViewById(R.id.custom_jsapi_btn).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        MPNebula.startUrl("https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/80000001/1.0.XX.XX_all/nebula/fallback/custom_jsapi.html");
    }
});
```

6. 编译工程后，在手机上安装应用。打开应用后界面如下：



7. 单击按钮后即可打开前端页面，单击按钮 **自定义 JSAPI**，会打开包含了一个按钮 **自定义 JSAPI** 的前端页面。单击该 **自定义 JSAPI** 按钮，会再弹出一个无标题警示框，内容按照自定义 API 定义的功能处理了的前端调用时传入的参数。至此，您已完成 **前端调用自定义 JSAPI 接口** 。





自定义 H5 页面的 TitleBar

H5 容器提供的方法可以设置自定义的标题栏，您可以继承 mPaaS 提供的默认标题栏 `MpaasDefaultH5TitleView` ，然后根据自己的需求，重写其中的一些方法。当然，您也可以自己实现 `H5TitleView` 。在本教程中我们使用 `MpaasDefaultH5TitleView` 。

1. 构建一个 `H5ViewProvider` 实现类，在 `createTitleView` 方法中返回您定义的 `H5TitleView` 。

```
package com.example.h5application;

import android.content.Context;
import android.view.ViewGroup;

import com.alipay.mobile.nebula.provider.H5ViewProvider;
import com.alipay.mobile.nebula.view.H5NavMenuView;
import com.alipay.mobile.nebula.view.H5PullHeaderView;
import com.alipay.mobile.nebula.view.H5TitleView;
import com.alipay.mobile.nebula.view.H5WebContentView;
import com.mpaas.nebula.adapter.view.MpaasDefaultH5TitleView;

public class H5ViewProviderImpl implements H5ViewProvider {
    @Override
    public H5TitleView createTitleView(Context context) {
        return new MpaasDefaultH5TitleView(context);
    }
    @Override
    public H5NavMenuView createNavMenu() {
        return null;
    }
    @Override
    public H5PullHeaderView createPullHeaderView(Context context, ViewGroup viewGroup) {
        return null;
    }
    @Override
    public H5WebContentView createWebContentView(Context context) {
        return null;
    }
}
```

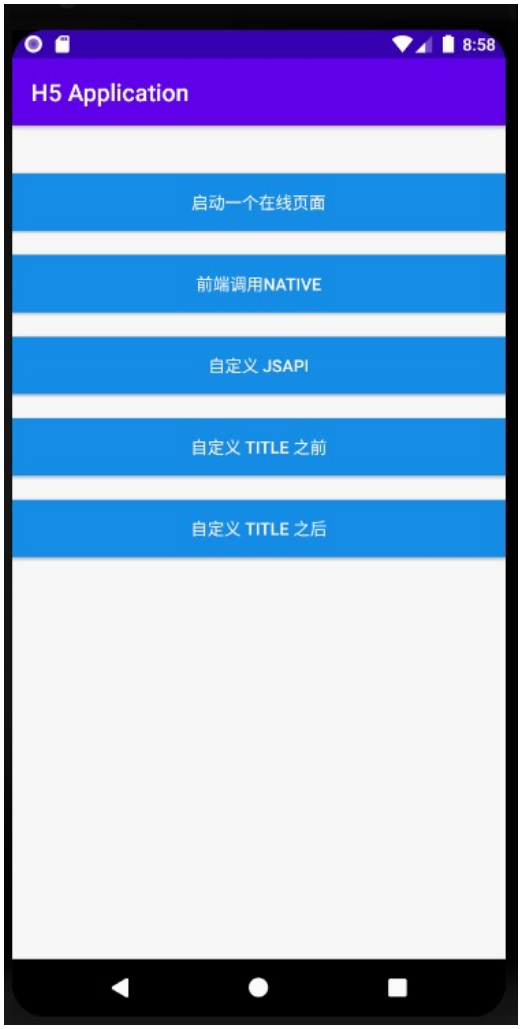
2. 在 `activity_main.xml` 文件中，新增按钮 button， `button id` 为 `custom_title_btn` 。

```
<Button
<Button
    android:id="@+id/custom_title_btn_before"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="自定义 Title 之前"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/custom_jsapi_btn" />

<Button
    android:id="@+id/custom_title_btn_after"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="自定义 Title 之后"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/custom_title_btn_before" />
```

3. 在 `MainActivity` 类定义单击按钮 `custom_title_btn` 后的行为，将自定义View Provider设给容器，并打开一个在线网页。实现代码如下所示：

```
        findViewById(R.id.custom_title_btn_before).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                MPNebula.startUrl("https://www.cloud.alipay.com/docs/2/49549");
            }
        });
        findViewById(R.id.custom_title_btn_after).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 设置自定义 title（设置一次即可）
                MPNebula.setCustomViewProvider(new H5ViewProviderImpl());
                // 随意启动一个地址，title 已经改变
                MPNebula.startUrl("https://www.cloud.alipay.com/docs/2/49549");
            }
        });
```

4. 编译工程后，在手机上安装应用。打开应用后界面如下：



5. 分别单击按钮 **自定义 TITLE 之前** 和 **自定义 TITLE 之后** 均会打开同一在线网页，可以看到两个页面的 titlebar 的颜色、字体颜色都发生了变化。至此，您已完成 **自定义 H5 页面的 TitleBar**。
   - 自定义 TitleBar 之前

- 自定义 TitleBar 之后

代码示例

点此下载 此教程中使用的代码示例。

1.7.1.6. 使用 H5 离线包

本文主要介绍如何使用 H5 离线包。

H5 离线包的使用可以分为以下四个部分：

1. 发布离线包
2. 预置离线包
3. 启动离线包
4. 更新离线包

在本教程中为了说明和演示 H5 离线包的功能，采取了从发布到预置，再到启动，最后完成更新的流程。

但是这一流程并非是使用 H5 离线包的必要条件。在实际生产中，您可以自由地根据需要进行使用。

发布离线包

本节介绍了发布离线包的操作流程。

前提条件

您需要准备一个前端 App 的 zip 包，如果您没有自己的前端离线包，您可以下载我们为您准备好的 示例离线包。

操作步骤

1. 先在控制台的应用中配置离线包信息，请参考 配置离线包。
2. 生成您自己的前端 App 的离线包（或者使用我们的示例离线包），请参考 生成离线包。

3. 在控制台上创建该离线包并上传，请参考 创建离线包。
4. 将配置好的离线包发布到您的客户端 App 中，请参考 发布离线包。

预置离线包

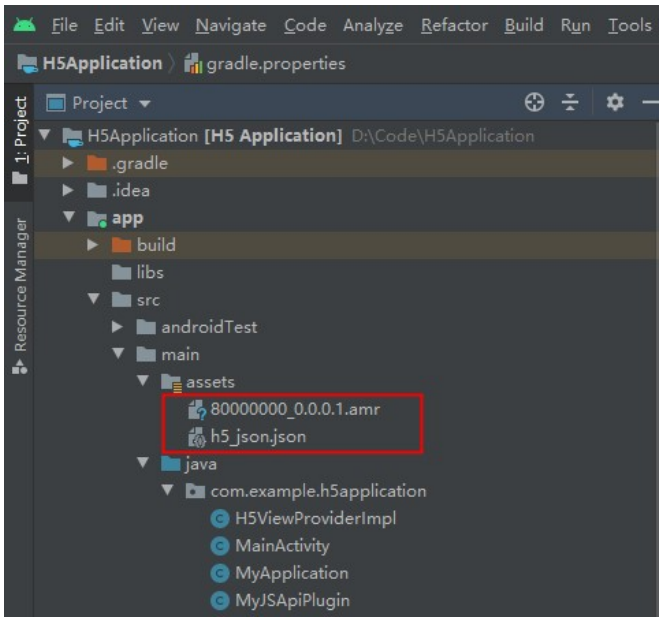
本节介绍了预置离线包的操作流程。

前提条件

您已经在 mPaaS 控制台发布了离线包。

操作步骤

1. 在控制台下载离线包 AMR 文件和离线包配置文件到本地。
2. 将下载到的离线包 AMR 文件和离线包配置文件放在工程中的 `assets` 目录下。



3. 将离线包预置到应用内。推荐在 APP 启动的时候注册，在本教程中在 `MyApplication` 类中进行注册。至此，您已经完成预置离线包。

```
public class MyApplication extends Application {

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

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

    private void loadOfflineNebula() {
        new Thread(new Runnable() {
            @Override
            public void run() {
            // 此方法为阻塞调用，请不要在主线程上调用内置离线包方法。如果内置多个 amr 包，要确保文件已存在，如不存在，
会造成其他内置离线包失败。
            // 此方法仅能调用一次，多次调用仅第一次调用有效。
            MPNebula.loadOfflineNebula("h5_json.json", new
MPNebulaOfflineInfo("80000000_1.0.0.0.amr", "80000000", "1.0.0.0"));
            }
        }).start();
    }
}
```

启动离线包

本节介绍了更新离线包的操作流程。

前提条件

您在客户端中已经预置了离线包。

操作步骤

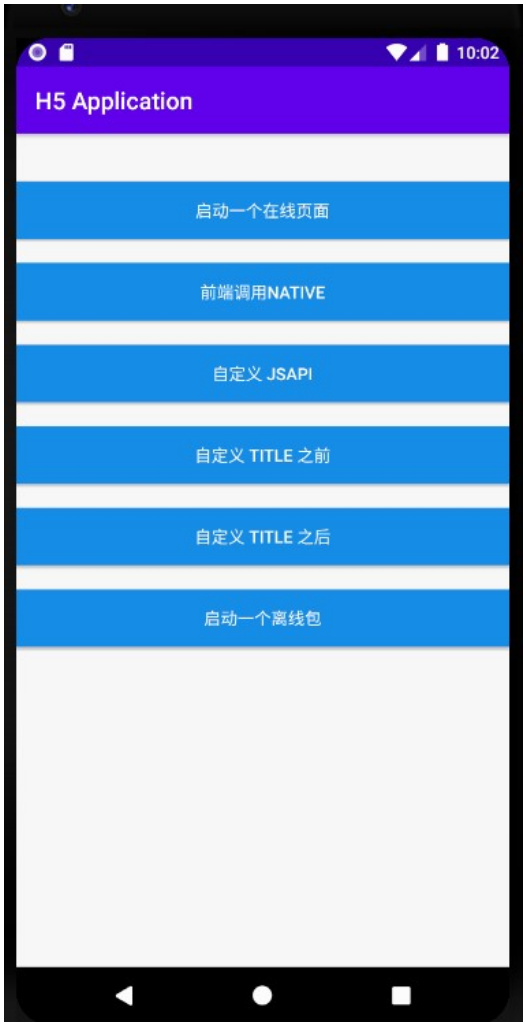
1. 在 activity_main.xml 文件中，新增按钮，按钮的 `id` 为 `start_app_btn` 。

```
<Button
    android:id="@+id/start_app_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="启动一个离线包"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/custom_title_btn_after" />
```

2. 在 `MainActivity` 类定义单击按钮 `start_app_btn` 后的行为，启动离线包。其中传入的参数“80000000”为离线包的 APPID。

```
findViewById(R.id.start_app_btn).setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
     MPNebula.startApp("80000000");
 }
});
```

3. 编译工程后，在手机上安装应用。打开应用后界面如下。



4. 单击按钮 **启动一个离线包** ，即可打开离线包中预置的网页。至此，您已完成 **启动离线包** 。

更新离线包

本节介绍了更新离线包的操作流程。

前提条件

您的客户端应用中已经预置了离线包，并且在 mPaaS 控制台上已经创建了新的离线包版本，也上传了新版本的离线包。

操作步骤

1. 在 activity_main.xml 文件中，新增按钮，按钮的 `id` 为 `update_app_btn` 。

```
<Button
    android:id="@+id/update_app_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="更新离线包"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/start_app_btn" />
```

2. 在 `MainActivity` 类定义单击按钮 `update_app_btn` 后的行为，更新离线包。其中传入的参数“80000000”为离线包的 APPID。

```
findViewById(R.id.update_app_btn).setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
     MPNebula.updateAllApp(new MpaasNebulaUpdateCallback() {
         @Override
         public void onResult(final boolean success, final boolean isLimit, String detailCode) {
             // success 为是否成功
             runOnUiThread(new Runnable() {
                 @Override
                 public void run() {
                     Toast.makeText(MainActivity.this, success ? "离线包更新成功" : "离线包更新失败",
Toast.LENGTH_SHORT).show();
                 }
             });
         }
     });
 }
});
```

3. 编译工程后，在手机上安装应用。打开应用后界面如下。

4. 单击按钮 **更新离线包**，即可打开更新离线包。在提示更新成功后，在单击按钮 **启动一个离线包**，即可以看到更新后的页面。至此，您已完成 **更新离线包**。

代码示例

点此下载 此教程中使用的代码示例。

1.7.2. iOS 使用教程

1.7.2.1. 总览

H5 容器是一款移动端 Hybrid 解决方案 SDK（Nebula SDK），提供了良好的外部扩展功能，拥有功能插件化、事件机制、JSAPI 定制和 H5App 推送更新管理能力。将 HTML、Javascript、CSS 等页面内的静态资源包打包到一个压缩包内，该压缩包即为离线包。使用离线包时通过预先下载该压缩包到本地，然后通过客户端打开，直接从本地加载离线包，从而最大程度地摆脱网络环境对 H5 页面的影响。本教程将带您一起体验和使用 H5 容器和离线包的基本能力。

本教程一共分为以下 4 个部分：

- 在 mPaaS 控制台创建应用并下载配置文件。
- 在 Xcode 创建工程。
- 使用 H5 容器。
- 使用 H5 离线包。

您将学会

- 如何创建一个 iOS 应用。
- 如何使用 mPaaS H5 容器服务。
- 如何使用 mPaaS H5 离线包服务。

您将需要

1. 配置开发环境。参考 准备配置 获取更多信息。
2. 可用的网络访问。
3. 网络浏览器（建议您使用 Chrome 浏览器）。

1.7.2.2. 在 mPaaS 控制台创建应用并下载配置文件

本文介绍的主要是在 mPaaS 控制台创建应用的操作步骤。

1. 打开网络浏览器，登录 mPaaS 控制台 。
2. 创建 mPaaS 应用。
3. 输入项目名称并点击 **确定**。
4. 进入刚创建的应用，单击 **代码配置** > **iOS** > **立即下载**。输入 **Bundle ID**（此处以 com.mpaas.demo 为例），单击 **保存**。 最后，点击 **下载配置** 下载配置文件。

1.7.2.3. 在 Xcode 创建工程

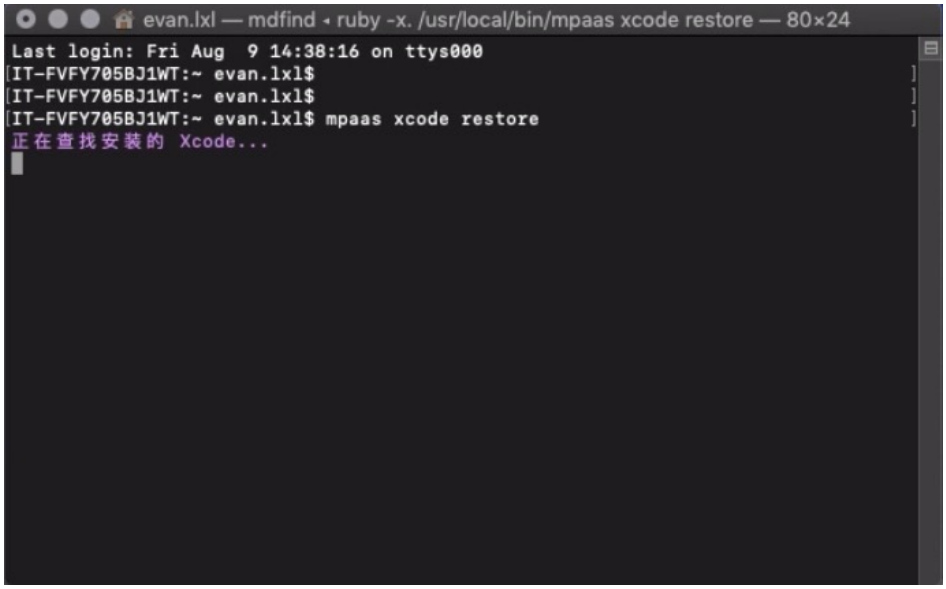
在本节您将创建一个显示 **Hello World!** 的应用。

点击 视频教程，可以查看对应的视频教程。

创建工程

1. 打开 Xcode。
2. 点击 **Edit** > **mPaaS** > **新建 mPaaS 工程** ，输入 **项目名称** 和 **项目路径**。 在此处项目名称以 **MyH5Application** 为例。
3. 导入从控制台下载到的 **配置文件**。Xcode 会自动解析并填充。
4. 选择 App 布局。
5. 选择 **H5容器&离线包** 项目组件。
6. 工程创建成功。
7. 单击 **OK**。
8. 恢复 Xcode 签名。

> **说明**
> 此步骤非必须步骤。当您发现在编译工程后模拟器无响应时，可恢复 Xcode 签名后再次尝试。



9. 点击 **Run**, 工程能够编译通过并且在模拟器中进行安装。 至此，您已完成创建工程。

1.7.2.4. 使用 H5 容器

您可使用 H5 容器完成以下操作。

- 初始化容器
- 在应用内打开一个在线网页
- 前端调用 Native 接口
- 前端调用自定义 JSAPI
- 自定义H5 页面的 TitleBar

初始化容器

为了使用 Nebula 容器，您需要在程序启动完成后进行初始化。在 `MyH5Application/MPaaS/Targets/MyH5Application/APMobileFramework/DTFrameworkInterface+MyH5Application.m` 中， `DTFrameworkInterface` 中的 `Category` 的 `beforeDidFinishLaunchingWithOptions` 方法中，调用以下接口进行初始化。

示例如下：

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)la
unchOptions
{
       // 初始化容器
        [MPNebulaAdapterInterface initNebula];

}
```

在应用内打开一个在线网页

在客户端代码中添加完容器初始化逻辑后，就可以调用 Nebula 容器提供的接口，启动一个在线网页。

单击 视频教程，查看对应的视频教程。

1. 在 `MyH5Application/Sources/DemoViewController.m` 中添加代码。添加一个按钮，单击按钮调用接口打开在线网页。

```
#import "DemoViewController.h"

@interface DemoViewController ()

@end

@implementation DemoViewController

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

    UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
    button.frame = CGRectMake(30, 150, [UIScreen mainScreen].bounds.size.width-60, 44);
    button.backgroundColor = [UIColor blueColor];
    [button setTitle:@"在线 URL" forState:UIControlStateNormal];
    [button addTarget:self action:@selector(openOnline)
forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button];

}

- (void)openOnline
{
    [[MPNebulaAdapterInterface shareInstance]
startH5ViewControllerWithParams:@{@"url":@"https://tech.antfin.com"}];
}

@end
```

2. 单击按钮后即可在应用内打开金融科技官网首页。至此，您已完成 **在应用内打开一个在线网页** 。

前端调用 Native 接口

在开发前端页面时，可以通过 Nebula 容器提供的 bridge，通过 JSAPI 的方式与 Native 进行通信，获取 Native 处理的相关信息或数据。Nebula 容器内预置了部分基础的 JSAPI 能力（详情参见 链接），您可以在 H5 页面的 js 文件中，直接通过 `AlipayJSBridge.call` 的方式进行调用。示例如下：

```
AlipayJSBridge.call('alert', {
  title: '原生 Alert Dialog',
  message: '这是一个来自原生的 Alert Dialog',
  button: '确定'
}, function (e) {
  alert("单击了按钮");
});
```

> **说明**
>
> 单击前往试用的前端页面，您可以调用此页面以体验前端调用Native接口的功能。

单击 视频教程，查看对应的视频教程。

1. 在 `MyH5Application/Sources/DemoViewController.m` 中添加代码。 添加一个按钮button1。

```
 UIButton *button1 = [UIButton buttonWithType:UIButtonTypeCustom];
    button1.frame = CGRectOffset(button.frame, 0, 80);
    button1.backgroundColor = [UIColor blueColor];
    [button1 setTitle:@"前端调用 native" forState:UIControlStateNormal];
[button1 addTarget:self action:@selector(openJsApi) forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button1];
```

2. 在 `MyH5Application/Sources/DemoViewController.m` 中，给button1添加实现代码。如下所示：

```
- (void)openJsApi
{
[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url":@"https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-default/80000000/1.0.XX.XX_all/nebula/fallback/h5_to_native.html"}];
}
```

3. 编译工程后，在手机上安装应用。打开应用后界面如下所示。

4. 单击按钮后即可打开前端页面，单击按钮 **显示原生 Alert Dialog**，会弹出原生的警示框，警示框的标题是 **原生 Alert Dialog**，消息框的内容是 **这是一个来自原生的 Alert Dialog** ；单击警示框的 **确定** 按钮，会再弹出一个无标题警示框，内容是 **点击了按钮** 。说明接口调用成功。至此，您已完成 **前端调用 Native 接口** 。

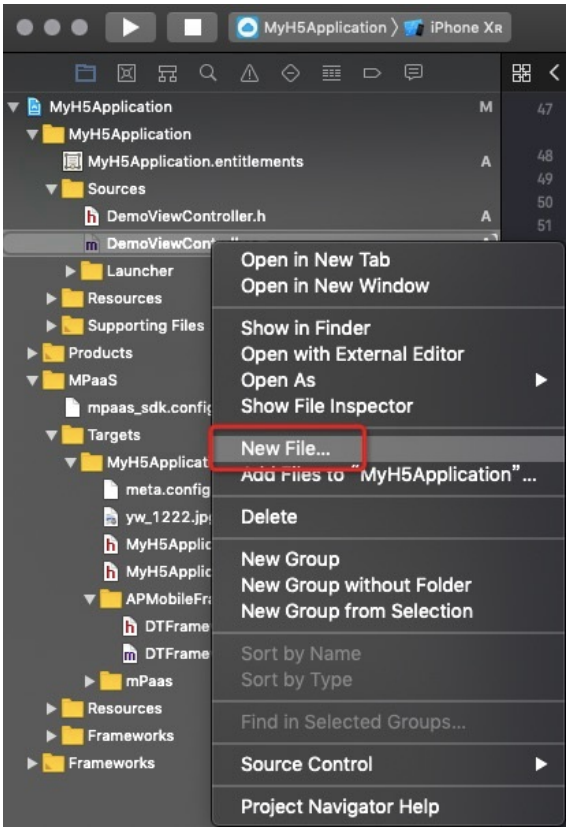
前端调用自定义 JSAPI

除了 Nebula 容器预置的基础 JSAPI 能力外，还可以根据以下方式自定义一个 JSAPI。

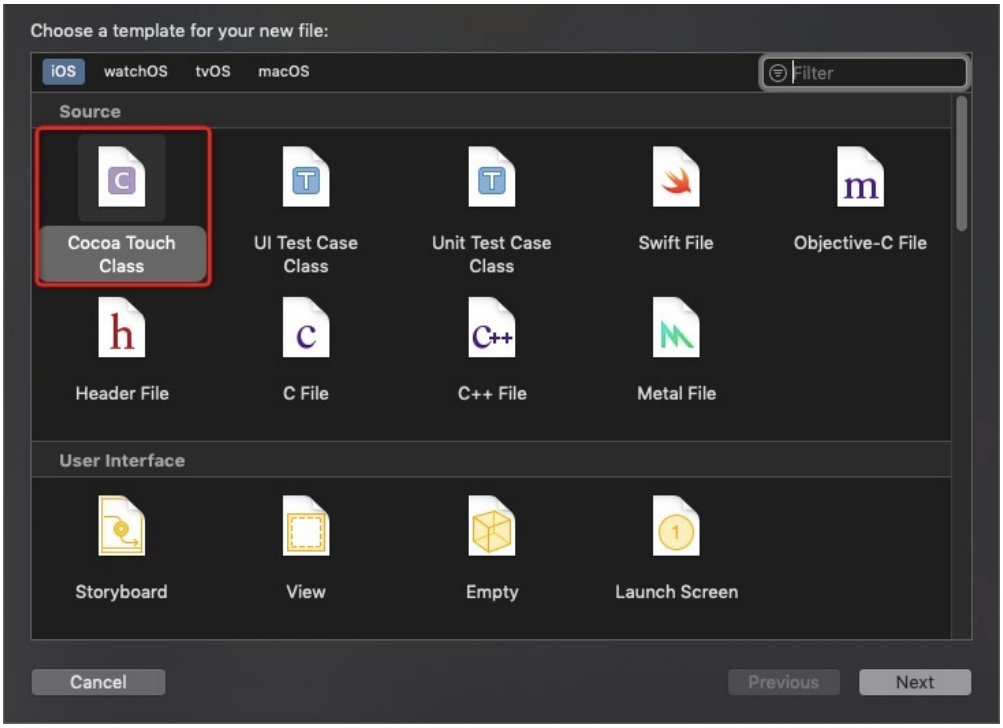
单击 视频教程，查看对应的视频教程。

1. 创建 JSAPI 类。

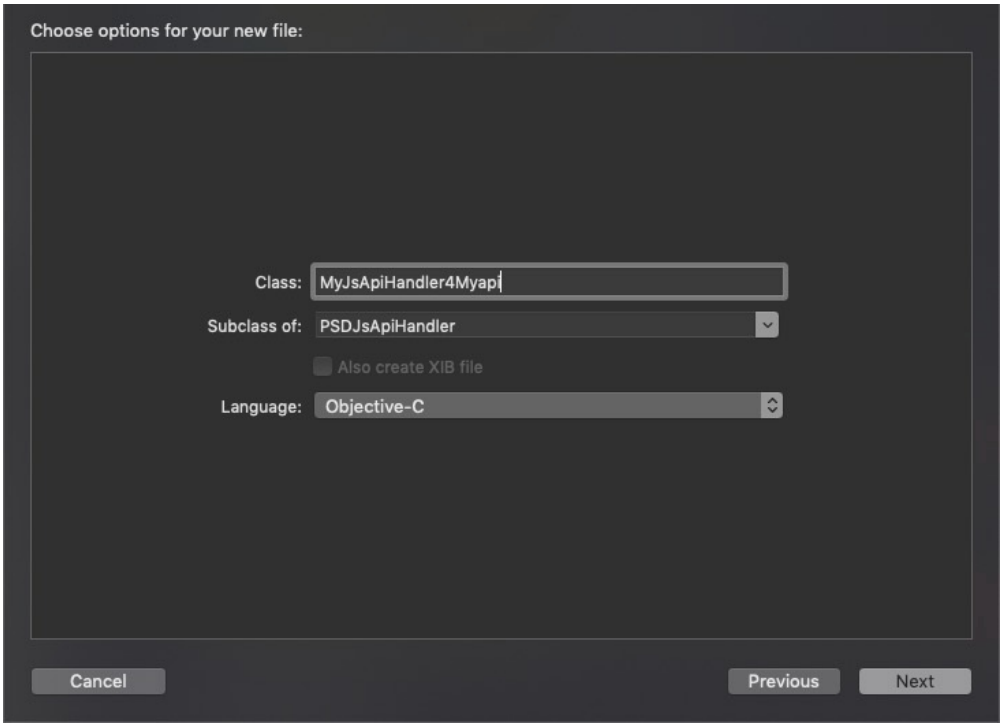
i. 第一步，操作如下图所示。



ii. 第二步，操作如下图所示。



iii. 第三步，操作如下图所示。



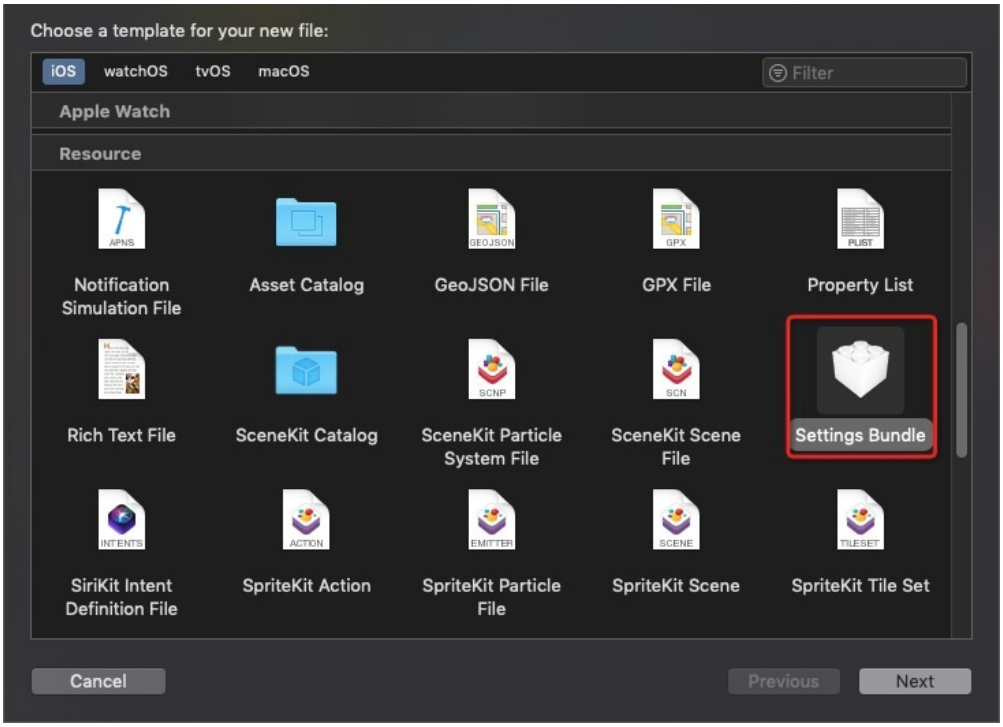
iv. 第四步，添加代码。

```
@implementation MyJsApiHandler4Myapi

- (void)handler:(NSDictionary *)data context:(PSDContext *)context callback:
(PSDJsApiResponseCallbackBlock)callback
{
    [super handler:data context:context callback:callback];

    NSString *userId = @"admin";
    if ([userId length] > 0) {
        callback(@{@"success":@YES, @"userId":userId});
    } else {
        callback(@{@"success":@NO});
    }
}

@end
```

2. 向 Nebula 容器注入您自定义的 JSAPI。

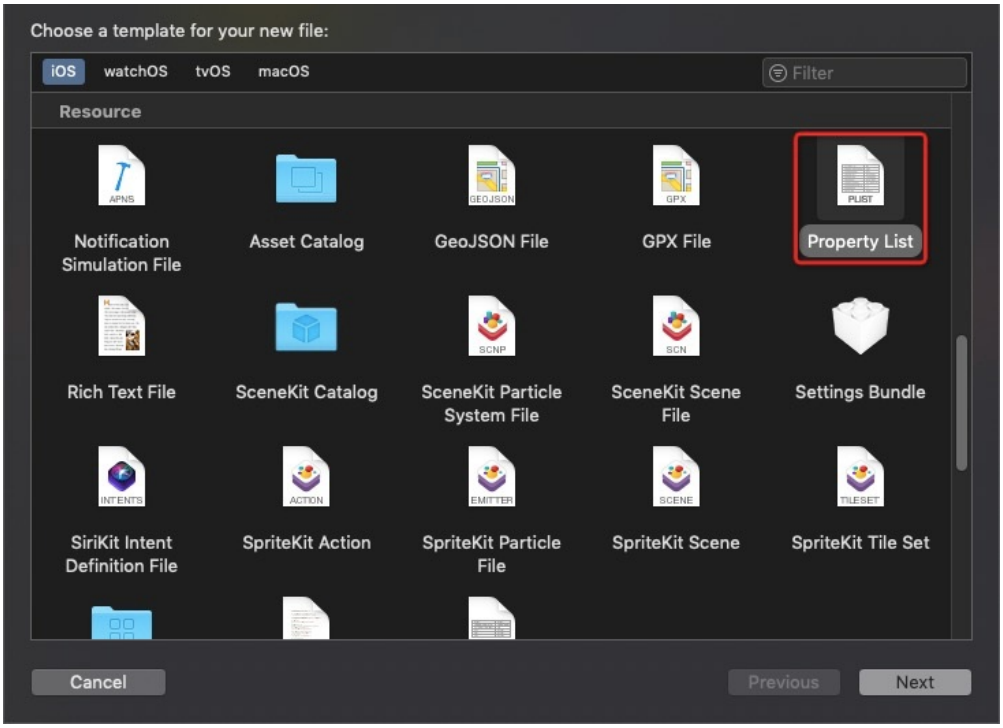
i. 在 `MyH5Application/MyH5Application/Resources` 下新建一个 bundle。

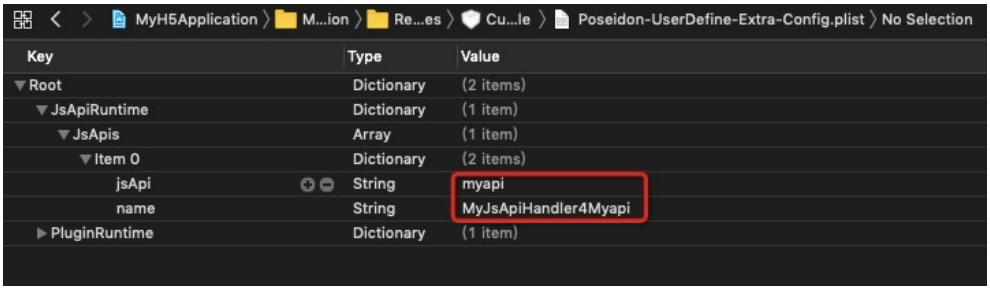
ii. 按下图所示进行操作。



iii. 将 bundle 重命名为 `CustomJsApi.bundle`，并将其内容清空。

iv. 在 `MyH5Application/MyH5Application/Resources` 中新建一个 plist 文件，重命名为 `Poseidon-UserDefine-Extra-Config.plist` 。在 JsApis 数组注册上一步中创建的 `JSAPI` 类。注册的 `JSAPI` 是一个字典 Dictionary 类型，其中 `JSAPI` 表示在 H5 页面中调用的 JSAPI 接口名 `myapi` ； `name` 表示上一步创建的类名 `MyJsApiHandler4Myapi` 。将 plist 文件拖拽到 `CustomJsApi.bundle` 中。





3. 向容器注册自定义 ISAPI 的路径。在 `MyH5Application/MPaaS/Targets/MyH5Application/APMobileFramework/DTFrameworkInterface+MyH5Application.m` 中，使用下面的接口，初始化容器。在采用自定义 JSAPI 的时候，需要采用以下方法，通过 plist 将自定义的类注入容器，将自定义的路径 `NSS string` 传给初始化方法。

```
 - (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *
)launchOptions
 {
 //        [MPNebulaAdapterInterface initNebula];
     NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"CustomJsApi.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
     [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:@""
customPresetAppPackagePath:@""customPluginsJsapisPath:pluginsJsapisPath];

 }
```

4. 在 `MyH5Application/Sources/DemoViewController.m` 中添加代码，新增按钮 button2。

```
 UIButton *button2 = [UIButton buttonWithType:UIButtonTypeCustom];
     button2.frame = CGRectOffset(button1.frame, 0, 80);
     button2.backgroundColor = [UIColor blueColor];
     [button2 setTitle:@"前端调用自定义 JSAPI" forState:UIControlStateNormal];
     [button2 addTarget:self action:@selector(openCusJsApi)
forControlEvents:UIControlEventTouchUpInside];
 [self.view addSubview:button2];
```

5. 在 `MyH5Application/Sources/DemoViewController.m` 中添加代码，定义 `openCusJsApi` 。实现代码如下所示：

```
 - (void)openCusJsApi
 {
 [[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url":@"https://mcube-p
rod.oss-cn-hangzhou.aliyuncs.com/570DA89281533-
default/80000001/1.0.XX.XX_all/nebula/fallback/custom_jsapi.html"}];
 }
```

> **说明**
>
> 单击前往**试用的前端页面**，您可以调用此页面以体验前端调用自定义 JSAPI 接口的功能。在该前端页面中，通过以下方法调用该自定义 JSAPI。示例如下所示。

```
AlipayJSBridge.call('myapi', {
  param1: 'JsParam1',
  param2: 'JsParam2'
}, function (result) {
  alert(JSON.stringify(result));
});
```

6. 编译工程后，在手机上安装应用。打开应用后界面如下所示。

7. 单击 **前端调用自定义 JSAPI** 按钮后即可打开前端页面，再单击按钮 **自定义 JSAPI**，会打开包含了一个按钮 **自定义 JSAPI** 的前端页面。单击该 **自定义 JSAPI** 按钮，会再弹出一个无标题警示框，内容按照自定义 API 定义的功能处理了的前端调用时传入的参数。至此，您已完成 **前端调用自定义 JSAPI 接口** 。

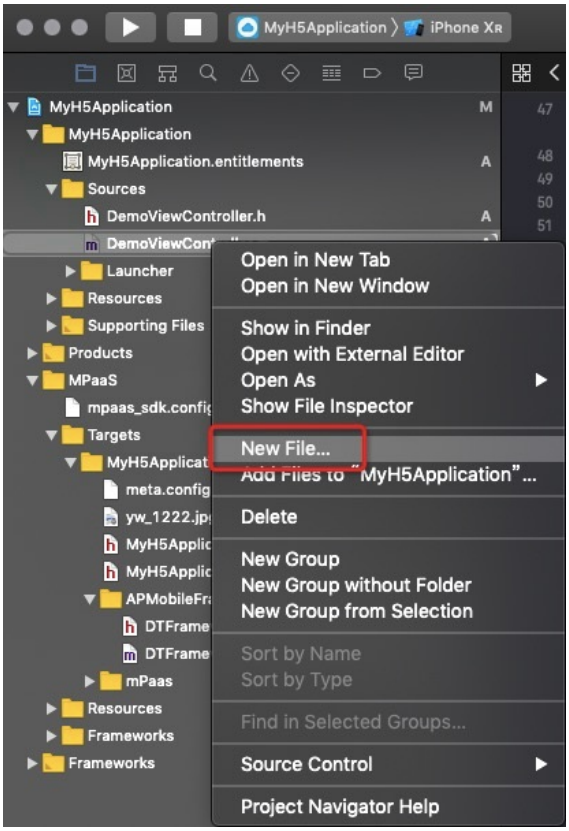
自定义 H5 页面的 TitleBar

H5 页面导航栏样式为白底黑字蓝按钮，您可以通过自定义一个 Plugin，来修改导航栏样式。

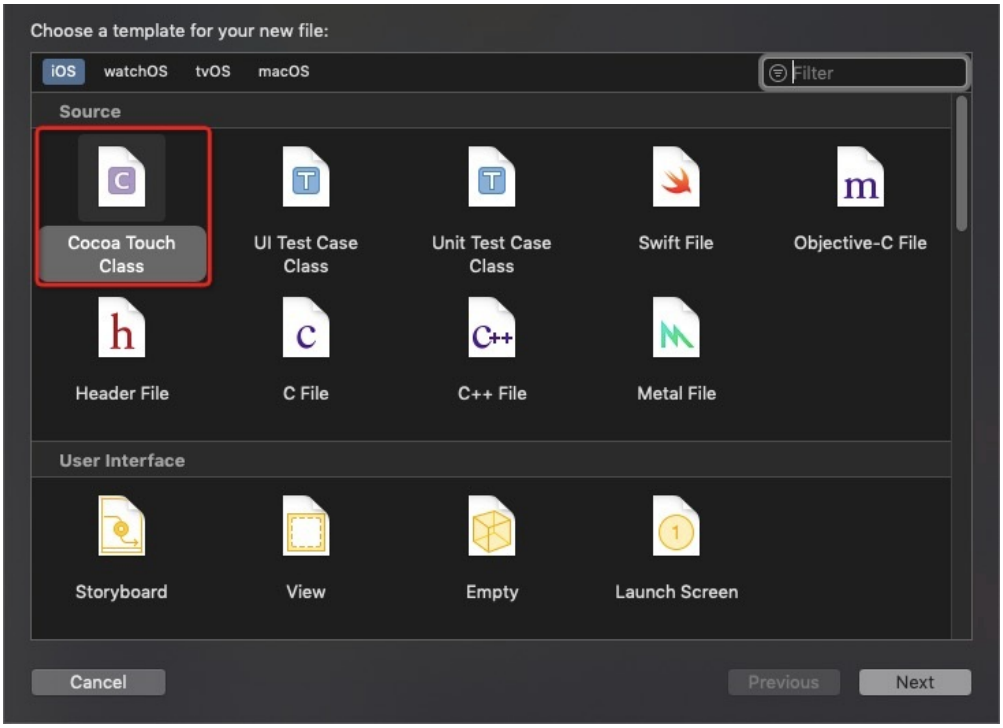
单击 视频教程，查看对应的视频教程。

1. 创建 Plugin 类。

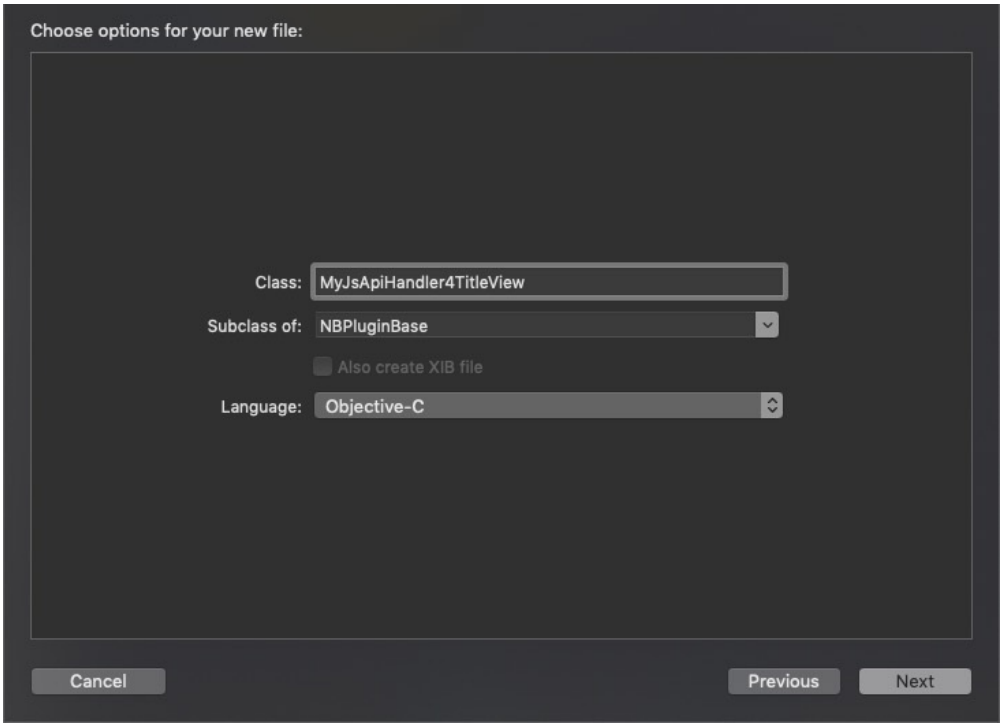
i. 按下图所示进行操作。



ii. 按下图所示进行操作。



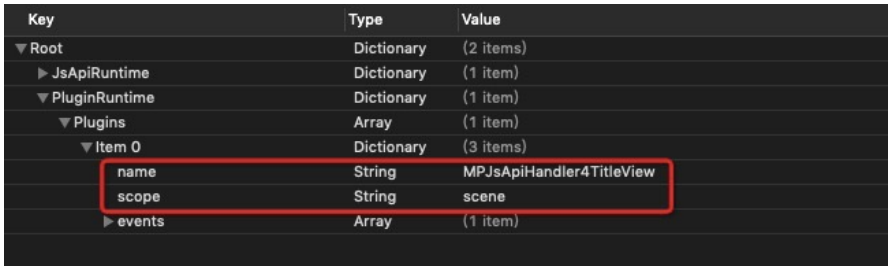
iii. 按下图所示进行操作。



iv. 添加代码。

```
- (void)pluginDidLoad
{
    self.scope = kPSDScope_Scene;

    // -- 返回区域
    [self.target addEventListener:kNBEvent_Scene_NavigationItem_Left_Back_Create_Before
withListener:self useCapture:NO];

    [super pluginDidLoad];
}

- (void)handleEvent:(PSDEvent *)event
{
    [super handleEvent:event];

    if ([kNBEvent_Scene_NavigationItem_Left_Back_Create_Before isEqualToString:event.eventType]) {
        // 在默认返回按钮基础上，修改样式
        NSArray *leftBarButtonItems =
event.context.currentViewController.navigationItem.leftBarButtonItems;
        if ([leftBarButtonItems count] == 1) {
            if (leftBarButtonItems[0] && [leftBarButtonItems[0] isKindOfClass:[AUBarButtonItem
class]]) {
                // 在默认返回按钮基础上，修改返回箭头和文案颜色
                AUBarButtonItem *backItem = leftBarButtonItems[0];
                backItem.backButtonColor = [UIColor redColor];
                backItem.titleColor = [UIColor redColor];
            }
        }
        [event preventDefault];
        [event stopPropagation];
    }
}

- (int)priority
{
    return PSDPluginPriority_High;
}
```

2. 向 Nebula 容器注入您自定义的 JSAPI。 打开 `Poseidon-UserDefine-Extra-Config.plist` 文件，在 Plugins 数组注册上一步中创建的 `MPJsApiHandler4TitleView` 类。注册的 Plugins 是一个字典 Dictionary 类型，其中 `name` 表示上一步创建的类名 `MyJsApiHandler4TitleView` 。 `Scope` 表示处理范围是页面（scene）。



3. 向容器注册自定义 plugin 的路径。在 `MyH5Application/MPaaS/Targets/MyH5Application/APMobileFramework/DTFrameworkInterface+MyH5Application.m` 中，使用下面的接口，初始化容器。在采用自定义 JSAPI 的时候，需要采用以下方法，通过 plist 将自定义的 plugin 类注入容器。

> **说明**
> 在上一步调用自定义 JSAPI 时已完成注册，可跳过本步骤。

```
 - (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *
)launchOptions
 {
 //        [MPNebulaAdapterInterface initNebula];
     NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"CustomJsApi.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
     [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:@""
customPresetAppPackagePath:@""customPluginsJsapisPath:pluginsJsapisPath];

 }
```

4. 在 `MyH5Application/Sources/DemoViewController.m` 中添加代码，新增按钮button3。

```
 UIButton *button3 = [UIButton buttonWithType:UIButtonTypeCustom];
 button3.frame = CGRectOffset(button2.frame, 0, 80);
 button3.backgroundColor = [UIColor blueColor];
 [button3 setTitle:@"自定义导航栏" forState:UIControlStateNormal];
 [button3 addTarget:self action:@selector(customNavigatorBar)
forControlEvents:UIControlEventTouchUpInside];
 [self.view addSubview:button3];
```

5. 在 `MyH5Application/Sources/DemoViewController.m` 中添加代码，定义 `customNavigatorBar` 。实现代码如下所示。

```
 - (void)customNavigatorBar
 {
     [[MPNebulaAdapterInterface shareInstance]
startH5ViewControllerWithParams:@{@"url":@"https://tech.antfin.com"}];
 }
```

6. 编译工程后，在手机上安装应用。打开应用后界面如下所示。

7. 单击 **自定义导航** 按钮后即可打开金融科技官网，此时返回按钮已经根据自定义调整样式。至此，您已完成 **自定义导航**。

1.7.2.5. 使用 H5 离线包

参考 在 Xcode 创建工程，创建新工程。我们基于此工程，使用 H5 离线包。

发布离线包

首先，在使用 H5 离线包之前，需要先准备一个前端 App 的 `zip` 包。如果没有自己的前端离线包，可以下载我们为您准备好的 示例离线包。

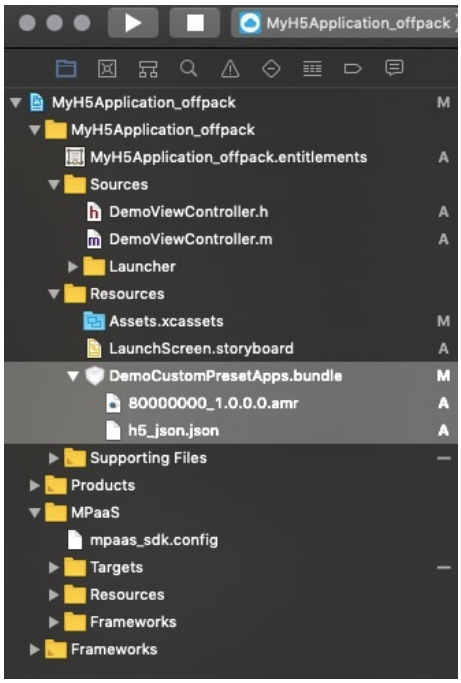
1. 在控制台的应用中配置离线包信息，参考 配置离线包。
2. 生成您自己前端 App 的离线包（或者使用我们的示例离线包），参考 生成离线包。
3. 在控制台上创建该离线包并上传，参考 创建离线包。
4. 将配置好的离线包发布到您的客户端 App 中，参考 发布离线包。

单击 视频教程，查看对应的视频教程。

预置离线包

1. 在控制台上下载离线包 AMR 文件，以及离线包配置文件到本地。

2. 在 `MyH5Application_offpack/Resources` 下创建 `DemoCustomPresetApps.bundle`，将 bundle 下的内容清空，并将下载到的离线包 AMR 文件和离线包配置文件添加到此 bundle 中。



3. 向容器注册预置的离线包。将包含预置离线包信息的 plist 文件和离线包的路径分别赋给 `presetApplistPath` 和 `appPackagePath`，通过 `initNebulaWithCustomPresetApplistPath` 进行初始化。至此，您已经完成 **预置离线包**。

```
 - (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *
)launchOptions
 {

     NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"DemoCustomPresetApps.bundle/h5_json.json"] ofType:nil];
     NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"DemoCustomPresetApps.bundle"] ofType:nil];
     [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath
customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:@""];
 }
```

单击 视频教程，查看对应的视频教程。

启动离线包

1. 在 `MyH5Application_offpack/Sources/DemoViewController.m` 中添加代码。添加按钮 button4，单击该按钮以调用接口打开离线包页面。

```
     UIButton *button4 = [UIButton buttonWithType:UIButtonTypeCustom];
     button4.frame = CGRectMake(30, 150, [UIScreen mainScreen].bounds.size.width-60, 44);
     button4.backgroundColor = [UIColor blueColor];
     [button4 setTitle:@"打开离线包页面" forState:UIControlStateNormal];
     [button4 addTarget:self action:@selector(startOffPack)
forControlEvents:UIControlEventTouchUpInside];
 [self.view addSubview:button4];
```

2. 在 `MyH5Application_offpack/Sources/DemoViewController.m` 中，给 button4 添加实现代码启动离线包。其中传入的参数 `80000000` 为离线包的 APPID。具体代码如下所示。

```
 - (void)startOffPack
 {
     [[MPNebulaAdapterInterface shareInstance]
startH5ViewControllerWithNebulaApp:@{@"appId":@"80000000"}];
 }
```

3. 编译工程后，在手机上安装应用。打开应用后界面如下所示。

4. 单击按钮 **打开离线包页面** ，即可打开离线包中预置的网页，如下图所示。至此，您已完成 **启动离线包** 。

单击 视频教程，查看对应的视频教程。

更新离线包

1. 在 `MyH5Application_offpack/Sources/DemoViewController.m` 中添加代码。 添加按钮 button5，单击按钮调用接口打开离线包页面。

```
 UIButton *button5 = [UIButton buttonWithType:UIButtonTypeCustom];
 button5.frame = CGRectOffset(button4.frame, 0, 80);
 button5.backgroundColor = [UIColor blueColor];
 [button5 setTitle:@"更新离线包" forState:UIControlStateNormal];
 [button5 addTarget:self action:@selector(updateOffPack)
forControlEvents:UIControlEventTouchUpInside];
 [self.view addSubview:button5];
```

2. 在 `MyH5Application_offpack/Sources/DemoViewController.m` 中，给 button5 添加实现代码以更新离线包。

```
- (void)updateOffPack
{
    [[MPNebulaAdapterInterface shareInstance] requestNebulaAppsWithParams:@{@"80000000":@"*"} finish
:^(NSDictionary *data, NSError *error) {
        NSString *result = @"后台无新包发布";
        if (!error && [data[@"data"] count] > 0) {
            result = [NSString stringWithFormat:@"%@", data[@"data"]];
        }

        dispatch_async(dispatch_get_main_queue(), ^{
            UIAlertView* alertView = [[UIAlertView alloc] initWithTitle:@"离线包已更新" message:result
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
            [alertView show];
        });
    }];
}
```

3. 编译工程后，在手机上安装应用。打开应用后界面如下所示。

4. 此时单击 **打开离线包页面** 按钮，会显示以下页面。

5. 单击 **更新离线包** 按钮，则会弹出 toast 提示 **后台无新包发布**。

6. 更新离线包文件，在控制台中上传并发布。在本例中更新的内容是将按钮显示文字更新为 **更新 20190826**。

> **说明**
>
> 在控制台上传新版本离线包以更新客户端离线包时，需要填写支持的客户端范围。此范围由最低版本和最高版本两个版本号确定。只有在此版本范围内的客户端，才能够得到推送的新版本离线包。客户端版本信息可以在客户端工程的 `info.plist` 文件的 `product version` 获得。

7. 在应用中单击 **更新离线包** 按钮，此时应弹出 toast 提示 **离线包已更新** ，如下图所示。

8. 此时单击打开 **离线包页面** 按钮，会显示以下页面。

至此，您已经完成 **更新离线包**。

单击 视频教程，查看对应的视频教程。

1.8. 视频教程

1.8.1. Android 视频教程

1.8.1.1. 添加 H5 组件 SDK

本视频介绍如何添加 H5 容器组件到工程。

1.8.1.2. 添加打开在线网页及效果预览

本视频介绍如何在应用内打开一个在线页面，并展示预览效果。

1.8.1.3. 添加使用原生 API 的代码及效果预览

本视频介绍前端调用原生 API 的代码，并展示预览效果。

1.8.1.4. 添加使用自定义 JSAPI 的代码及效果预览

本视频介绍前端调用自定义 JSAPI 的代码，并展示预览效果。

1.8.1.5. 添加自定义 TitleBar 代码及效果预览

本视频介绍自定义 H5 页面的 TitleBar，并展示预览效果。

1.8.1.6. 发布离线包和下载 amr 文件

本视频介绍如何发布离线包和下载 amr 文件。

1.8.1.7. 预置离线包

本视频介绍预置离线包的相关操作。

1.8.1.8. 启动离线包及效果预览

本视频介绍如何启动离线包，并展示预览效果。

1.8.1.9. 更新离线包及效果预览

本视频介绍如何更新离线包，并展示预览效果。

1.8.2. iOS 视频教程

1.8.2.1. 创建工程并添加 H5 容器组件

本视频向您介绍创建工程并添加 H5 容器组件的操作过程。

1.8.2.2. 在应用内打开一个在线网页

本视频向您介绍在应用中打开一个在线页面的操作教程。

1.8.2.3. 前端调用 Native 接口

本视频向您介绍前端调用 Native 接口的操作教程。

1.8.2.4. 前端调用自定义 JSAPI

本视频介绍了前端调用自定义 JSAPI 的使用方法。

1.8.2.5. 自定义 H5 页面的 TitleBar

本视频为您介绍如何自定义 H5 页面的 TitleBar。

1.8.2.6. 发布离线包

本视频为您介绍如何发布离线包。

1.8.2.7. 预置离线包

本视频为您介绍预置离线包的使用教程。

1.8.2.8. 启动离线包

本视频为您介绍启动离线包的使用教程。

1.8.2.9. 更新离线包

本视频为您介绍更新离线包的使用教程。

1.9. 常见问题

1.9.1. Android 常见问题

本文介绍接入 Android 过程中常见的问题及相应的解决方案。

自定义 JSAPI 时，handleEvent 和 interceptEvent 有什么区别？

解答：如果监听容器自己处理的事件，需要将自定义 JSAPI 添加到 `handleEvent` 里，并返回 `true` 。返回 `true` ，表示事件将停止传递；返回 `false` ，表示事件将继续传递给其他插件。如果监听容器的其他事件，需要将自定义 JSAPI 添加到 `interceptEvent` 中。

自定义 JSAPI 时，已经添加了事件，为什么还要在 onPrepare 里添加一次？

解答：自定义 JSAPI 时，虽然已经在 `config.setEvents("event");` 中添加了事件，但是容器的插件是懒加载的，即在页面创建的时候加载。通过外部的 `config.setEvents` 来注入要监听的事件名称，当真正有 JS 调用的时候，才会去实例化对应的插件对象。真正实例化的插件的事件分发用的是插件内 `onPrepare` 的事件。所以，要确保 `config.setEvents("event")` 的事件和内部 `onPrepare` 的事件保持一致。

自定义 JSAPI 插件注册的 page、session 和 service 三者的区别是什么？

解答：page 对应一个 WebView。session 对应 mPaaS 应用的一个 App 对象。service 是全局的一个单例。

- 如果注册为 page 级别，每次创建 WebView 都会创建一个插件实例，对应的插件的 onRelease 在 WebView 销毁时回调。
- 如果注册为 session 级别，每次创建一个 App 对象都会创建一个插件。
- 如果注册为 service 级别，全局只创建一次插件，在第一次打开容器的时候创建。

验签是验证离线包的来源还是做完整性校验？

解答：在离线包发布平台，如果您配置了签名私钥，平台下发的 AMR 文件就会带上离线包的签名信息（通过私钥对离线包的 Hash 值加密后得到的密文）。验签的时候会使用在项目中预置的公钥进行签名的验证（解密上一步的密文得到 Hash 值，本地计算离线包的 Hash，判断两者是否相同）。该过程保证了离线包的内容完整，也保证了离线包的来源正确。

打开离线包 ProgressBar 无法隐藏的原因是什么？

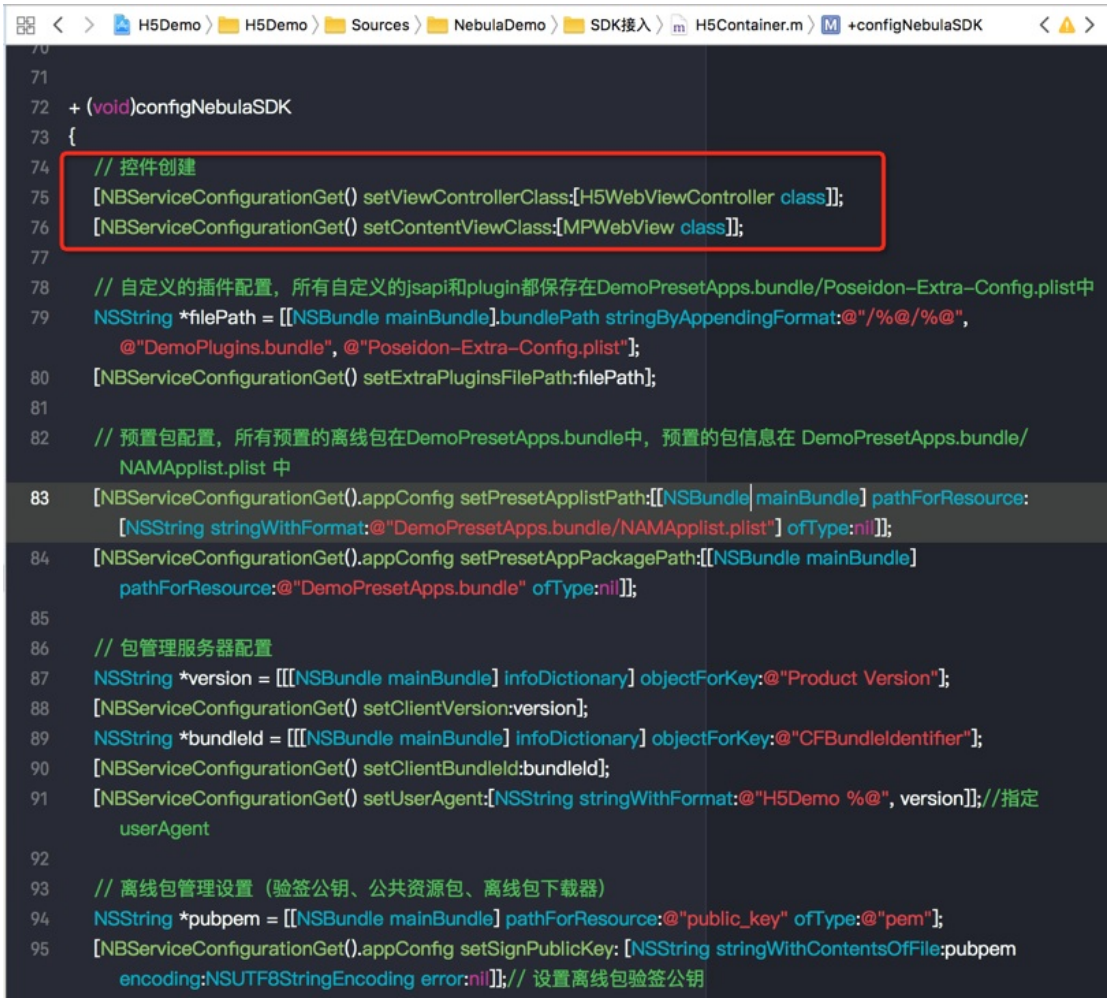
解答：这种情况可能是由于资源离线失败，页面 fallback 到了线上模式，在 fallback 模式下面 H5 容器强制限制 ProgressBar，无法通过设置 `SHOW_PROGRESS` 启动参数来隐藏页面加载进度条（Progress Bar）。

1.9.2. iOS 常见问题

本文介绍接入 iOS 过程中常见的问题及相应的解决方案。

注意事项

- H5 容器初始化时，最好指定所有 H5 页面的基类和 WebView 的基类，便于统一处理。



- 前端在开发离线包时，对应路径及文件名中不允许有中文字符，否则会导致客户端离线包解压失败。
- 离线包中各资源路径的绝对长度不要超过 100 字符，否则导致客户端 tar 包解压失败，页面白屏。

如何解决 H5 容器定位偏移问题

解答：在使用 mPaaS 容器的过程中可能会遇到 H5 容器定位偏移的问题，请参考以下方法进行设置更新：

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)la
unchOptions
{

    //跳过 LBS 定位
    [LBSmPaaSAdaptor sharedInstance].shouldSkipLBSLocation = YES;

    .......

}
```

预置离线包使用 H5_json.json 文件不生效

解答：在 10.1.32 基线中，只支持 plist 格式。在 10.1.60 基线中，plist 和 JSON 格式都支持。

如何获取已安装的离线包应用信息

解答：参考代码 `NSDictionary *installedApps = [NAMServiceGet() installApps:nil];` 。

如何强制更新全部离线包信息

解答：可以使用封装的 `requestAllNebulaApps` 方法进行全量更新。

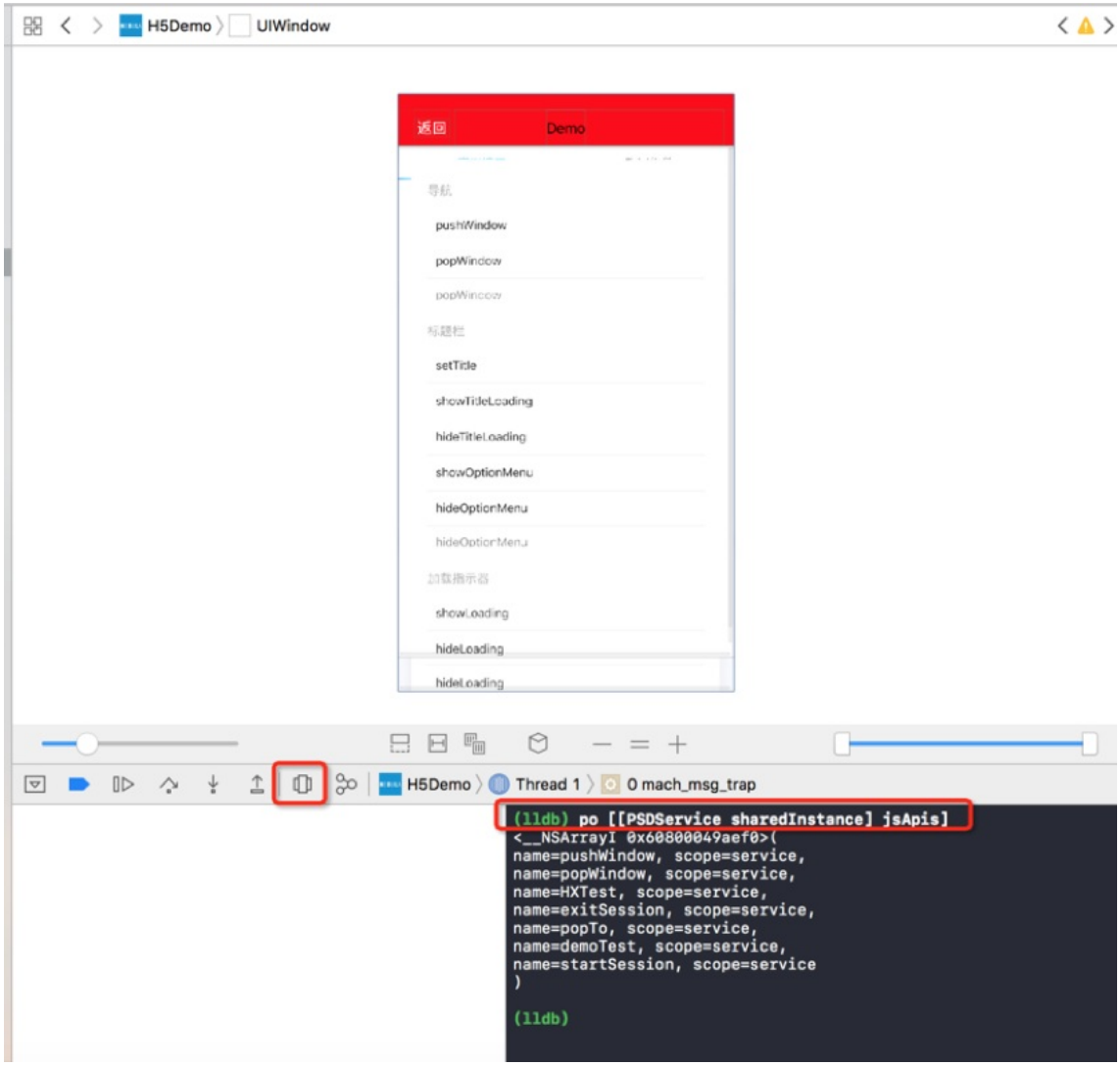
```
[[MPNebulaAdapterInterface shareInstance]requestAllNebulaApps:^(NSDictionary *data, NSError *error) {

 }];
```

在 iOS 13 中的 vux-ui pulldown 组件在 H5 容器中滑动卡死时如何处理

解答：这是因为触发了 `UIWebview` 在 iOS 系统中的 bug，可以考虑切换成 WKWebview 或更换前端组件。切换为 WKWebview 的方法请参考文档 mPaaS 10.1.60 适配 WKWebView。

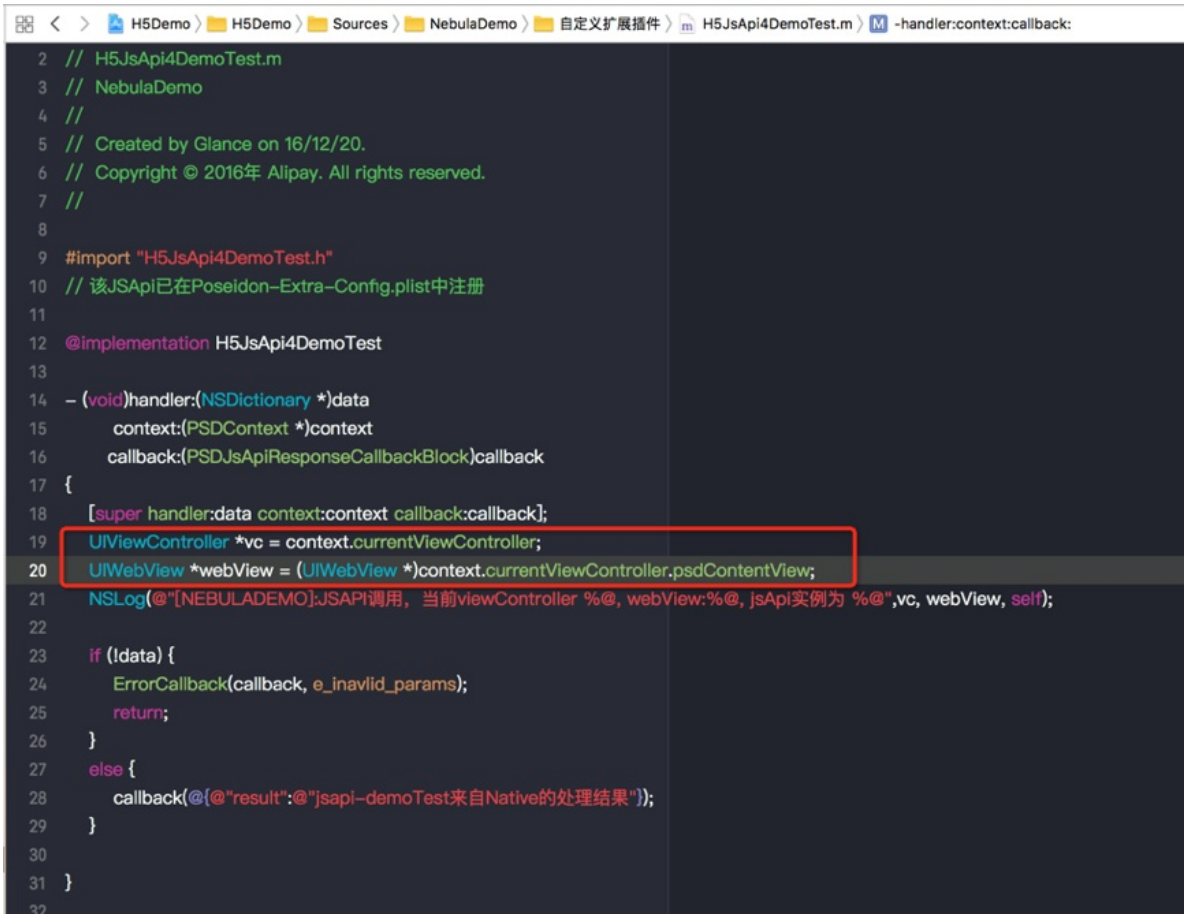
如何查看当前应用已注册的所有 JSAPI 和 Plugins

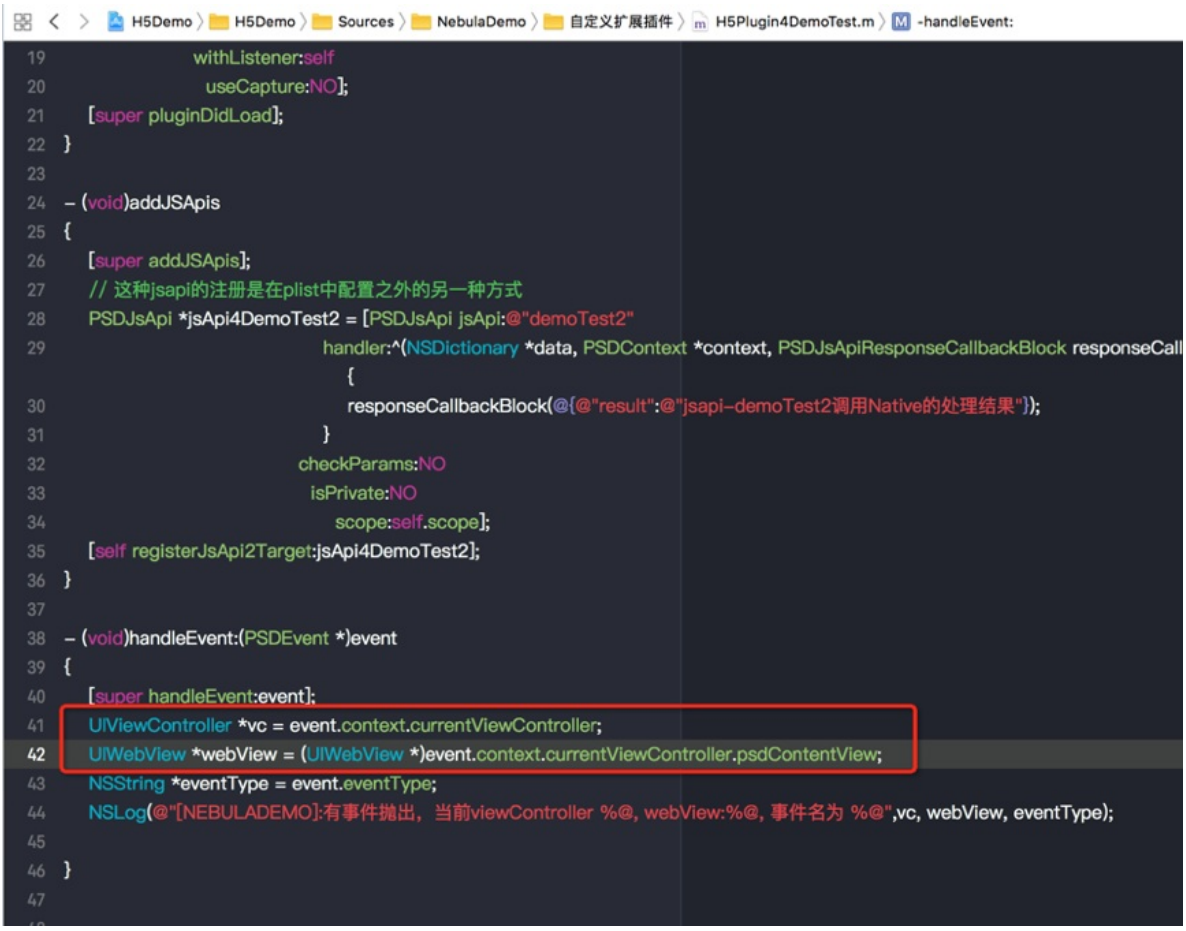
解答：打开一个 H5 页面，进入 Xcode View 层级页面，在 lldb 控制台使用 `po [[PSDService sharedInstance] jsApis]` 查看所有 JSAPI。同理使用 `po [[PSDService sharedInstance] plugins]` 查看所有注册的 Plugins。



如何在 JSAPI 或 Plugin 中获取当前 H5 页面的 UIViewController 和 Webview 对象

解答：在实际执行过程中，Plugin 可直接拿到参数 `event.context`，JSAPI 中可获取到 `context`，在上下文 `PSDContext` 对象中，可以取到您想要的所有信息或引用，比如当前控制器 `event.currentViewController` 的引用，当前 WebView 的引用 `context.currentViewController.psdContentView` 等等。

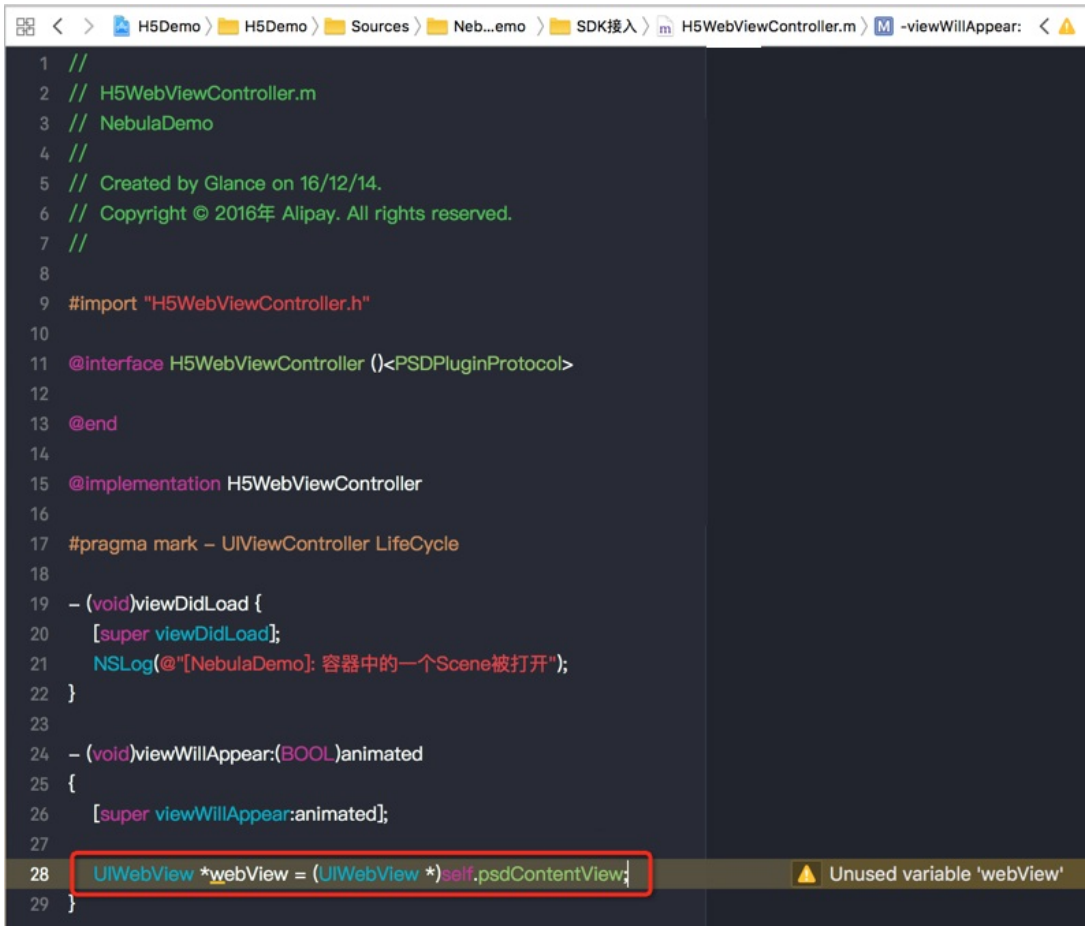

H5Demo 〉 H5Demo 〉 Sources 〉 NebulaDemo 〉 自定义扩展插件 〉 H5JsApi4DemoTest.m 〉 -handler:context:callback:
// H5JsApi4DemoTest.m
// NebulaDemo
//
// Created by Glance on 16/12/20.
// Copyright © 2016年 Alipay. All rights reserved.
//
#import "H5JsApi4DemoTest.h"
// 该JSApi已在Poseidon-Extra-Config.plist中注册
@implementation H5JsApi4DemoTest
- (void)handler:(NSDictionary *)data
context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];
UIViewController *vc = context.currentViewController;
UIWebView *webView = (UIWebView *)context.currentViewController.psdContentView;
NSLog(@"[NEBULADEMO]:JSAPI调用，当前viewController %@, webView:%@, jsApi实例为 %@",vc, webView, self);
if (!data) {
ErrorCallback(callback, e_inavlid_params);
return;
}
else {
callback(@{@"result":@"jsapi-demoTest来自Native的处理结果"});
}
}



如何获取当前页面的 WebView

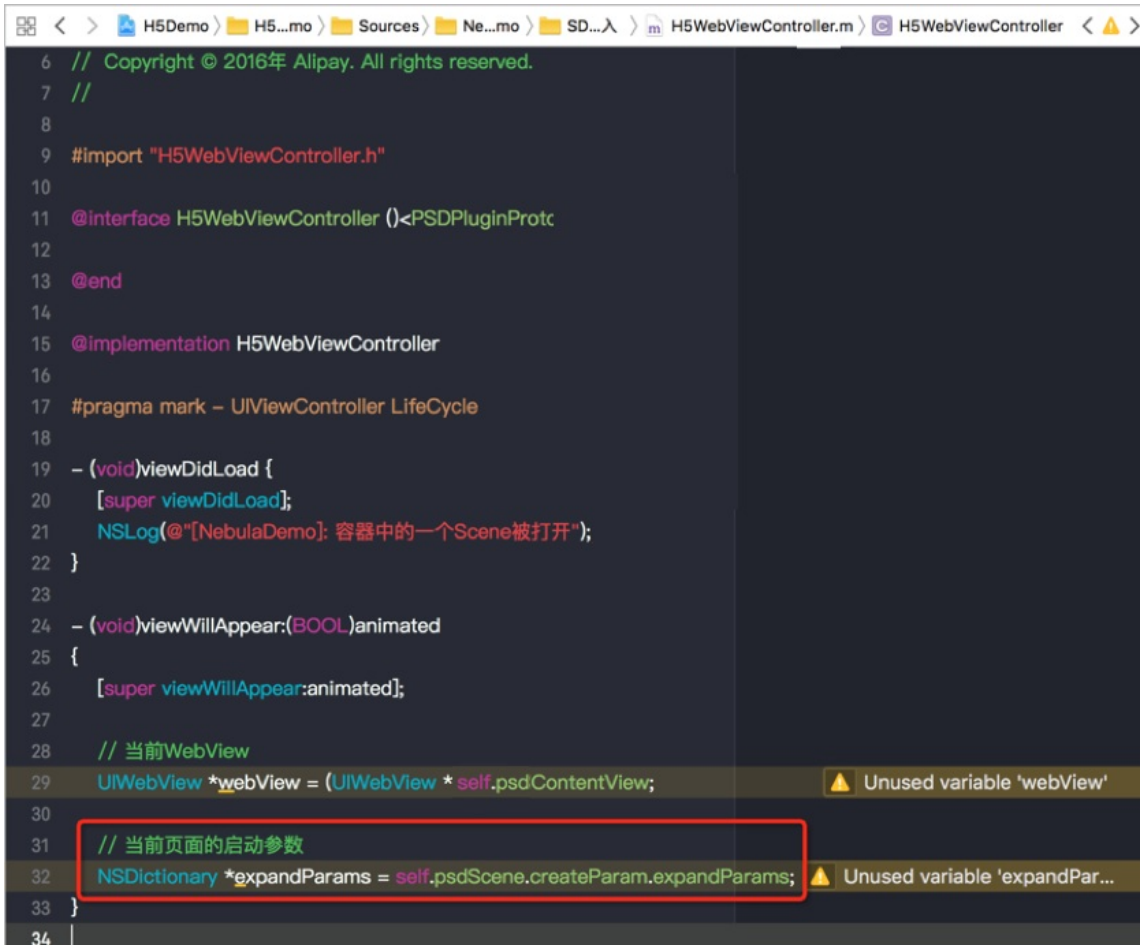
解答：当前 H5 页面的 WebView 是当前 VC 的一个属性，可通过 `vc.psdContentView` 获取，在 JSAPI 或 Plugin 中可通过上面的方法获取当前页面的 VC。

> **说明**
>
> 在 H5 页面基类获取当前页面的 WebView 时，请在 viewWillAppear 方法中。viewDidLoad 方法中 WebView 未创建，如果使用该方法，取到的值会为 nil。



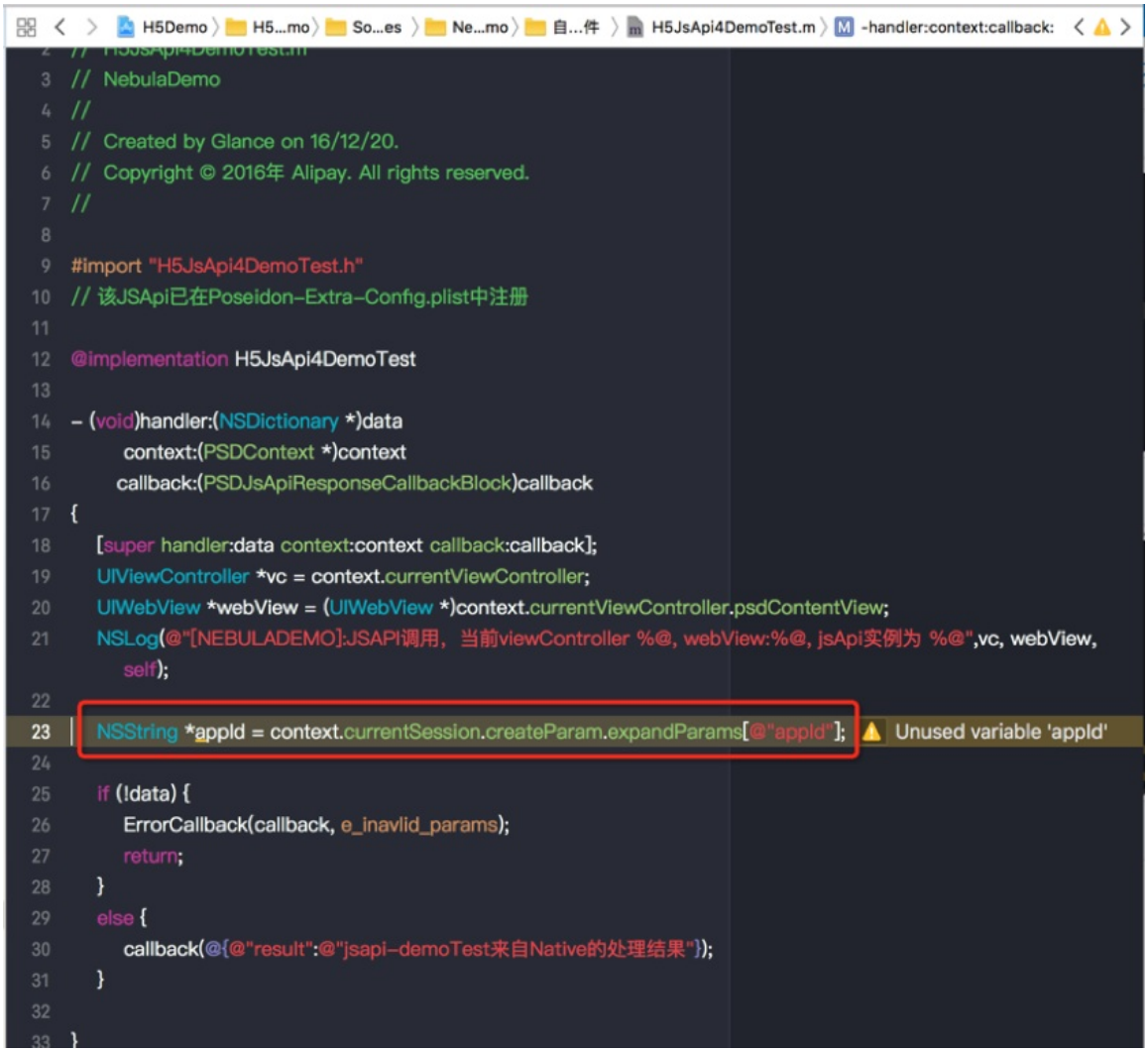
如何获取当前页面加载时前端传入的启动参数

解答：直接获取当前 VC 的 `psdScene.createParam.expandParams` 属性。



如何限制 JSAPI 只在某个特定离线包中起作用

解答：在 JSAPI 的实际执行方法中，获取当前页面所属离线包的 appId，决定是否执行逻辑即可。



如何拦截页面 URL

解答：您可以自定义 Plugin，通过监听事件来实现。

- 监听事件名称 `[PSDProxy addEventListener:kEvent_Proxy_Request_Start_Handler withListener:self useCapture:YES];` 。
- 拦截处理。

```
else if ([kEvent_Proxy_Request_Start_Handler isEqualToString:event.eventType]
        && [event isKindOfClass:[PSDProxyEvent class]] ){

        NSLog(@"-----kNBEvent_Scene_NavigationItem_Right_Setting_Click----");
        PSDProxyEvent *proxyEvent = (PSDProxyEvent*) event;
        NSMutableURLRequest *redirectReq = proxyEvent.request.mutableCopy;
        NSString *appId = event.context.currentSession.createParam.expandParams[@"appId"];

        NAMApp *app = [NAMServiceGet() findApp:appId version:nil];
        NSString *fallBackUrl = app.fallback_host;
        NSString *vhost = app.vhost;;
        NSString *url = redirectReq.URL.absoluteString;


       NSLog(@"url__%@_____fallBackUrl:%@",url,fallBackUrl);
       if ([url containsString:@"www.baidu.com"]) {
           [proxyEvent preventDefault];

 }}
```

如何在当前 H5 页面加载前，拦截当前页面 URL

解答：在 H5 页面的基类中，实现 UIWebView 的生命周期的代理方法中，监听 `kEvent_Navigation_Start` 事件，在页面加载之前拦截，获取当前页面的 WebView 和 URL 进行相关处理。

```
#pragma mark - 对应 UIWebViewDelegate 的委托实现

-(void)handleEvent:(PSDEEvent *)event
{
  if(![[event.context currentViewController] isEqual:self]){
    return;
  }
  if([kEvent_Navigation_Start isEqualToString:event.eventType]){
    BOOL shouldStart = [self handleContentViewShouldStartLoad:(id)event];

    if(!shouldStart){
      [event preventDefault];
    }
  }
  else if([kEvent_Page_Load_Start isEqualToString:event.eventType]){
    [self handleContentViewDidStartLoad:(id)event];
  }
  else if([kEvent_Page_Load_Complete isEqualToString:event.eventType]){
    [self handleContentViewDidFinishLoad:(id)event];
  }
  else if([kEvent_Navigation_Error isEqualToString:event.eventType]){
    [self handleContentViewDidFailLoad:(id)event];
  }
  else if([kEvent_Sence_NavigationItem_Left_Back_Click isEqualToString:event.eventType]){

  }

  -(BOOL)handleContentViewShouldStartLoad:(PSDNavigationEvent *)event{
    return YES;
  }
}
```

如何在 Native 手动调用 JSAPI

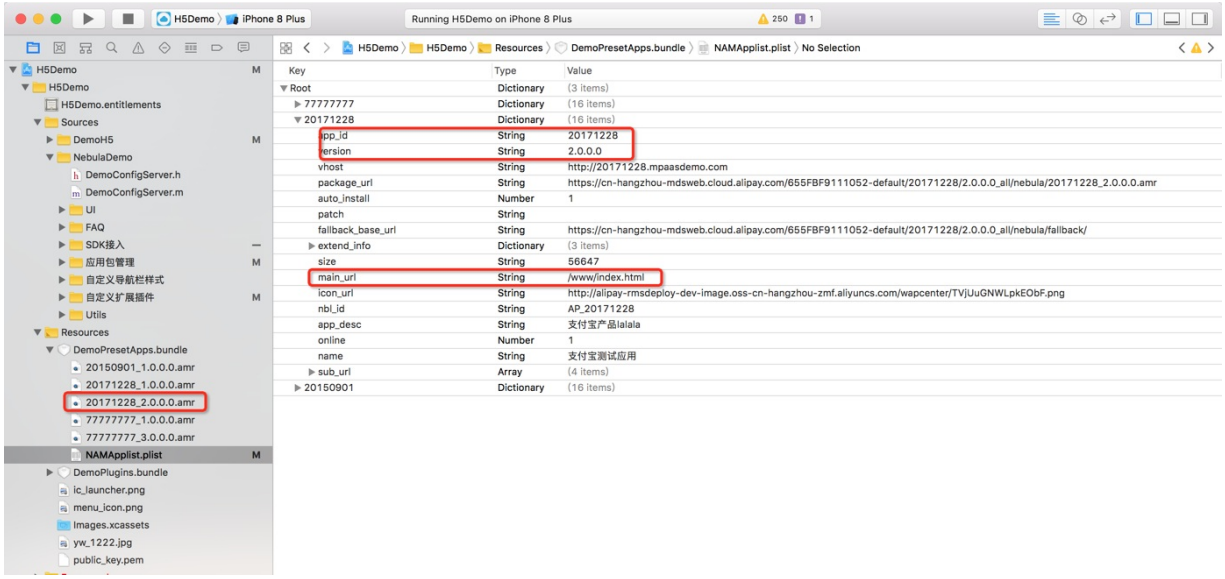
解答：有时 Native 端可能需要您在当前页面手动调用某个 JSAPI 接口，可通过调用当前 VC 的以下接口实现。

```
  if(self.navigationController){
    NSMutableArray *aar = [[self.navigationController viewControllers] mutableCopy];
    [arr removelastObject];

    [self.navigationController setViewControllers:aee animated:YES];
  }
  else{
    [self dismissViewControllerAnimated:YES completion:NULL];
  }
}

-(IBAction)btnRightItemClicked:(id)sender
{
  NSLog(@"[NebuiaDemo]:导航栏右侧按钮被点击":);

  //native 手动触发 JSAPI
  [self callHandler:@"HXTest" data:@{@"key":@"value"} responseCallback:^(id responseData){

  }];
}
```

为什么本地预置的离线包加载不生效

解答：预置资源包加载失败一般为预置包版本和包信息不匹配，测试本地预置离线包时，请先断开网络，避免离线包有更新，确保加载的是客户端本地预置的版本。

检查本地预置的离线包信息与 Plist 中配置的包信息是否一致，包括 app_id、version、main_url 等信息。

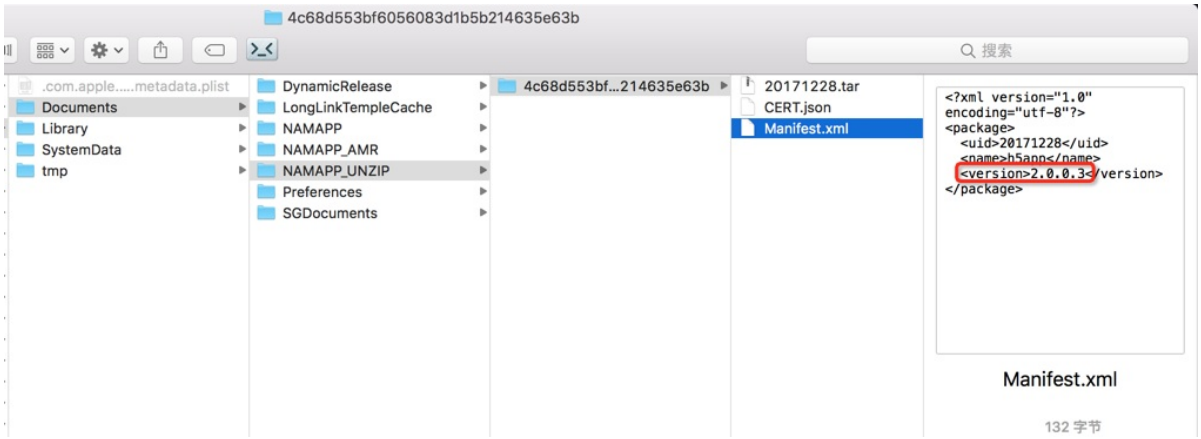


为什么控制台发布新版本离线包后客户端不能正常加载到新包

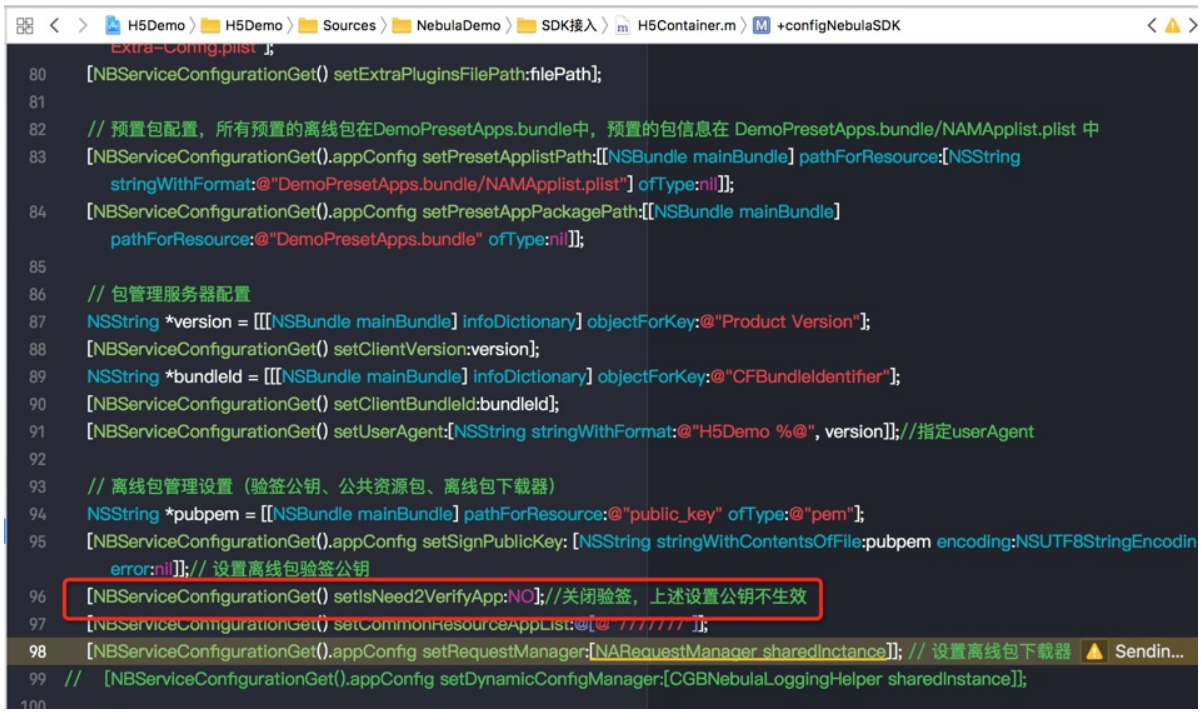
解答：在查看此问题解决方案前，确认您已理解 离线包更新原理，客户端不能正常加载新包，离线包渲染在任一阶段都有可能出错，下面将一一进行排查：

1. 查看全量更新离线包的 RPC 返回结果，在控制台搜索“bizType: 4”确认返回的离线包详情，确认已经拉取了控制台发布的最新包信息。
2. 上一步检查通过后，在加载离线包的 finish 回调方法里查看离线包信息，确保为控制台发布的最新包，且 error 的值为 nil，检查离线包的 app_id、version、main_url 等是否正确。

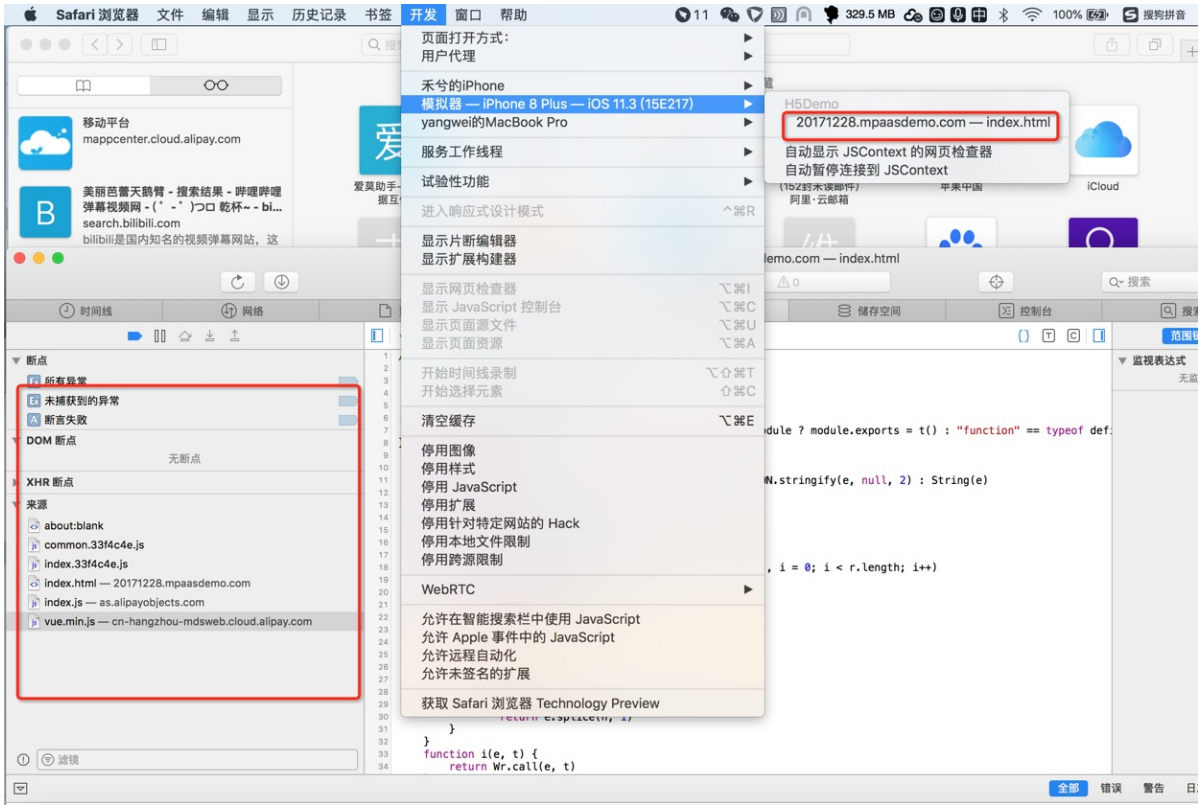
3. 上一步检查通过后，查看沙盒目录下离线包是否解压成功（若当前离线包有引用全局资源包中内容，此解压目录下同样需要有全局资源包）。



4. 若上一步中沙盒目录下无对应离线包，可先暂时关闭离线包验签，删除 App 重新运行。若关闭验签后加载正常，说明离线包加签私钥和客户端验签公钥不一致，请更新客户端对应的公钥信息。



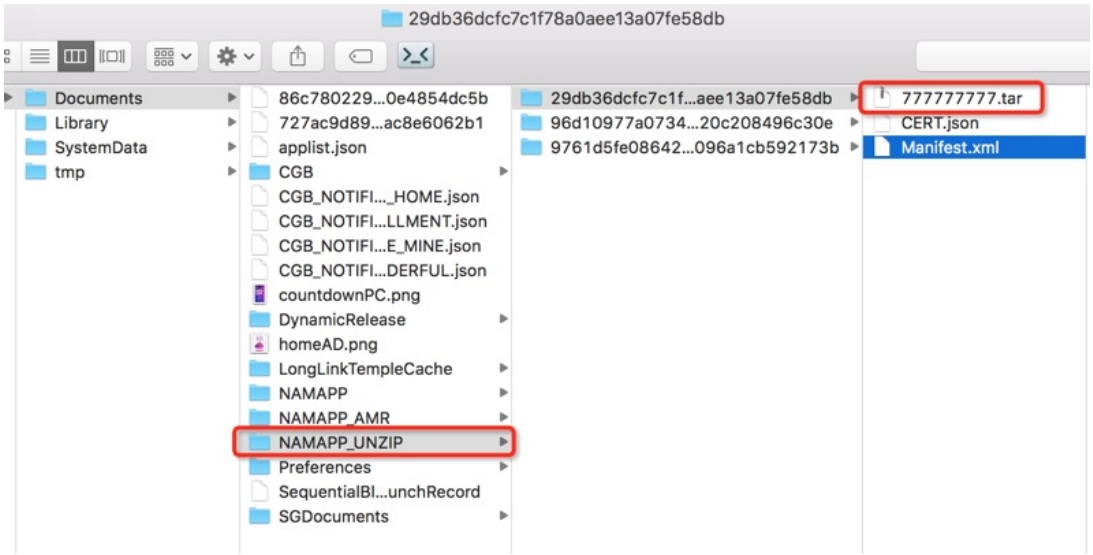
5. 若上述步骤检查都通过，离线包更新仍失败，可用 Safari 调试对应 H5 页面，具体查看报错原因。如全局资源包路径不正确，离线包加载的 URL 不存在等。



全局资源包加载失败

解答：当通过 Safari 调试判断出全局资源包加载失败时，可按下述步骤具体排查：

1. 检查全局资源包是否注册。
2. 上一步检查通过后，查看沙盒目录下离线包是否解压成功。



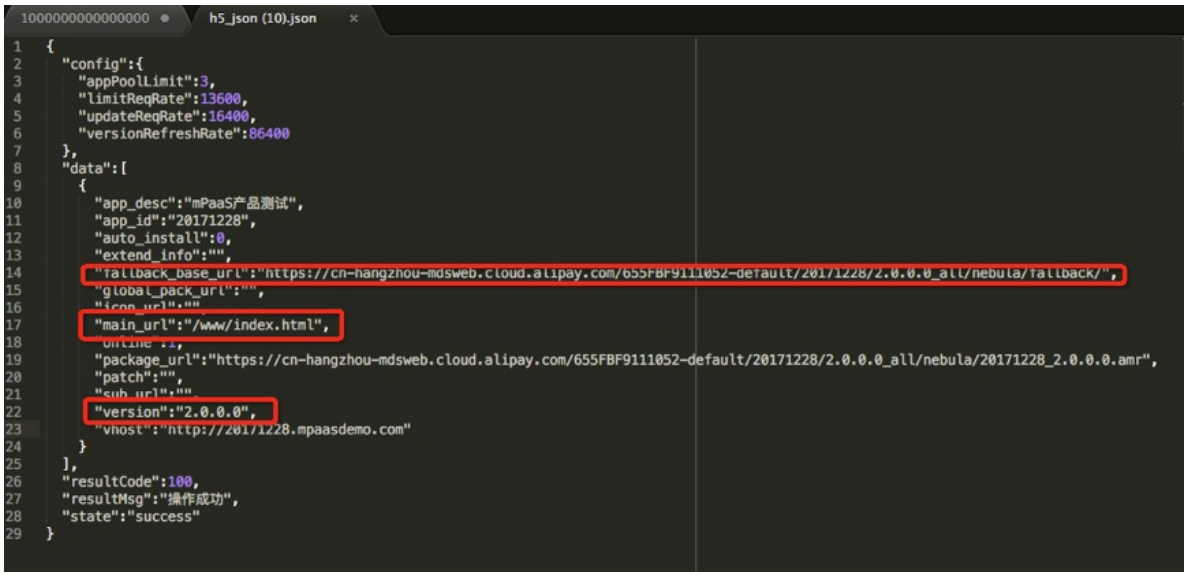
3. 检查引用的全局资源包中资源文件路径是否正常，且保证引用路径中无中文字符。

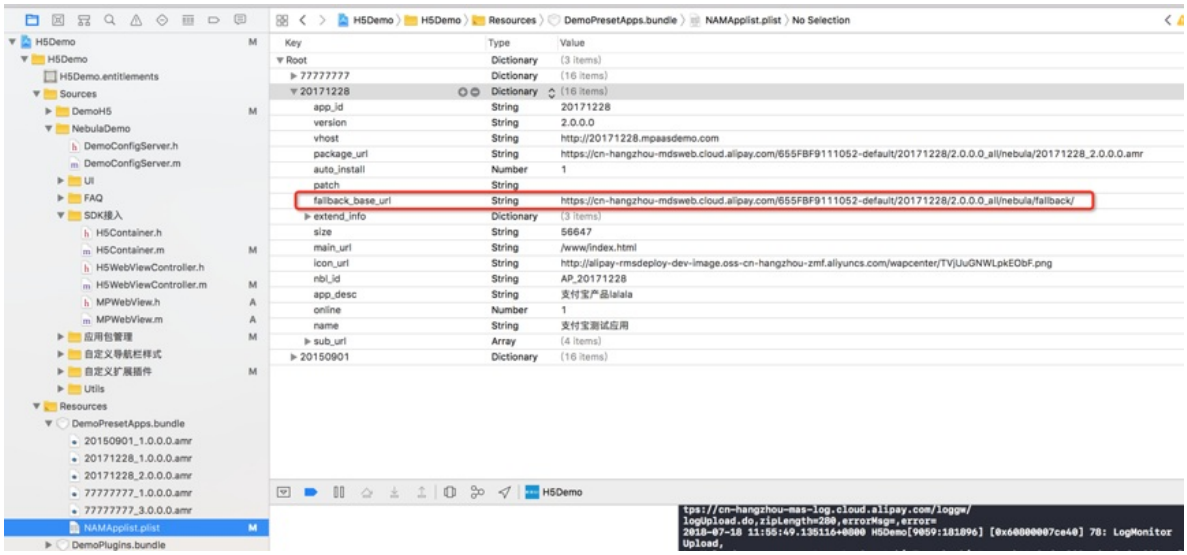
为什么打开某个离线包 H5 页面会白屏或出现 400 错误

解答：页面白屏或出现 400 错误，一般为本地离线包加载失败导致使用了在线 fallback 地址，而对应页面的 fallback 地址不存在，导致页面加载失败。

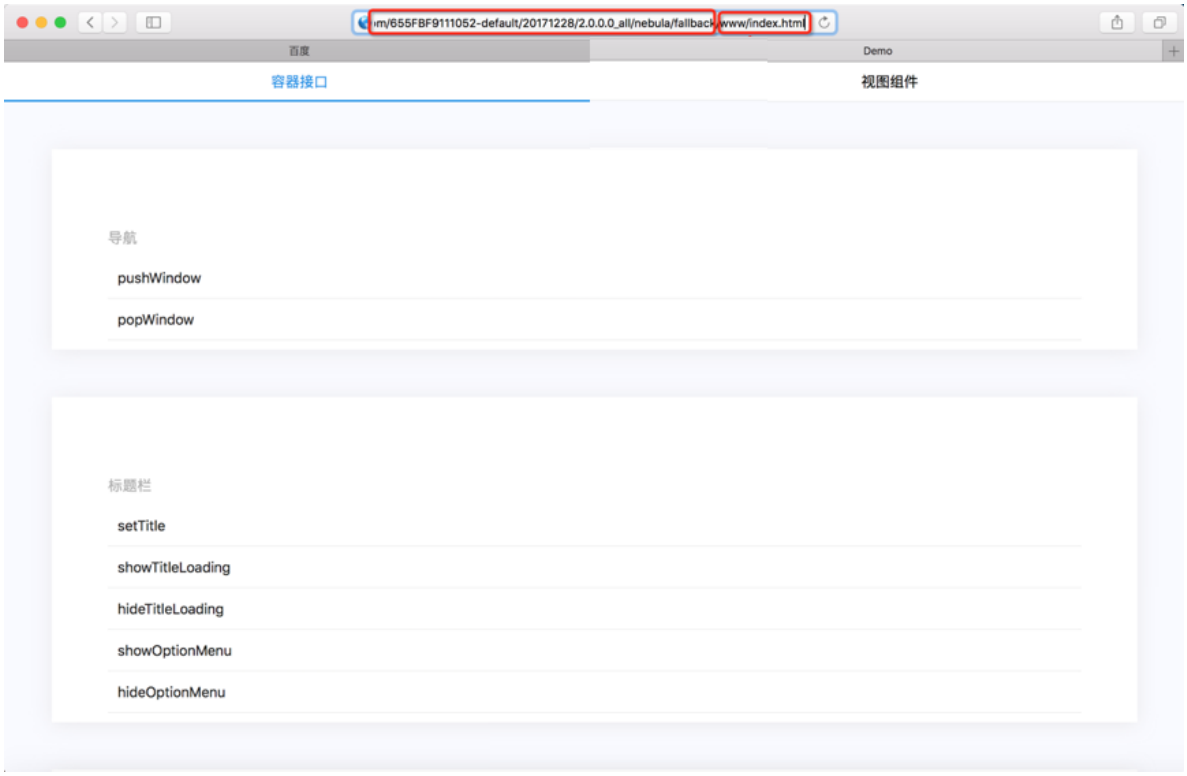
1. 按上述离线包问题排查客户端离线包加载失败的原因。

2. 针对加载在线 fallback 地址失败的问题，确认对应离线包版本生成正确，且已在控制台已上传，包括普通离线包和全局资源包，具体请参考 生成离线包。
3. 预置离线包若出现在线 fallback 地址失败的问题，先保证预置的离线包在控制台同样也上传。
4. 保证本地预置包信息中 `fallback_base_url` 与从控制台下载的 h5_json.json 配置文件中 `fallback_base_url` 一致。





5. 并且 `fallback_base_url + main_url` 拼接的地址在浏览器上可正常加载。



怎样禁止 H5 页面的手势侧滑返回功能

解答：支持由前端 H5 页面禁止和原生 H5 容器基类禁止。

1. 前端 H5 页面禁止：调用 setGestureBack JSAPI 实现。适用于某一个页面需要禁止手势侧滑返回的场景 `AlipayJSBridge.call('setGestureBack',{val:false});` 。
2. 原生 H5 容器基类禁止：在基类的 viewDidAppear 方法中调用系统禁止侧滑返回的接口，同时设置 guestBack 参数。适用于多个或所有 H5 页面需要禁止手势侧滑返回的场景。

```
-(void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];

    self.options.gestureBack = NO;
    if ([self.navigationController respondsToSelector:@selector(interactivePopGestureRecognizer)]) {
        self.navigationController.interactivePopGestureRecognizer.enabled = NO;
    }
}
```

如何判断当前页面是小程序中的页面

解答：获取当前页面所在的 session，调用 `isTinyAppWithSession` 接口判断。代码示例如下：

```
PSDSession *session = self.psdSession;
BOOL isTinyApp = [NBUtils isTinyAppWithSession:session];
```

H5 页面如何传递自定义参数

解答：根据传参方式，分为以下几种场景：

- 原生 - H5：调用 `startH5ViewControllerWithParams` 方法时传递 `[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithParams:@{@"url": @"https://tech.antfin.com", @"key1":@"value1"}];` 。
- 原生 - 离线包：调用 `startH5ViewControllerWithNebulaApp` 方法时传递 `[[MPNebulaAdapterInterface shareInstance] startH5ViewControllerWithNebulaApp:@{@"appId":@"70000000",@"param":@{@"key2":@"value2"}}];` 。
- H5 - H5：调用 `pushWindow` 时将自定义参数写在 `passData` 中。

```
AlipayJSBridge.call('pushWindow', {
// 要打开页面的 URL
url: 'https://m.taobao.com/',
// 打开页面的配置参数
param: {
  readTitle: true,    //自动读取 title
  showOptionMenu: false,    // 隐藏右边菜单
 transparentTitle:'always',
},
// 用于给新开的页面传递参数，可选
// 在新开的页面使用 AlipayJSBridge.startupParams 可以获取到 passData
passData: {
  key1: "key1Value",
  key2: "key2Value"
}
});
```

- H5 - 离线包：调用 startApp JSAPI 时将自定义参数写在 param 中。

```
AlipayJSBridge.call('startApp', {
appId: '70000000',
param: {
key1:'value1'
}
}, function(result) {
// noop
});
```

如何在 H5 页面获取传递的参数

解答：分为前端获取和原生获取两种场景：

- 前端获取：通过 `startupParams` 方法获取当前页面的启动参数。

  `AlipayJSBridge.startupParams`

- 原生获取：通过当前页面所在的 VC 对象获取。

```
// 当前页面的启动参数
NSDictionary *expandParams = self.psdScene.createParam.expandParams;
NSLog(@"[mpaas] expandParams: %@", expandParams);
```

如何拦截 JSAPI 调用

解答：可以自定义 Plugin，通过监听事件来实现。

- 监听事件名称：`kEvent_Invocation_Event_Start`。
- 拦截处理：获取到 JSAPI 的名称、传递的参数等。

```
else if([kEvent_Invocation_Event_Start isEqualToString:event.eventType]) {
      PSDInvocationEvent * invocationEvent = (PSDInvocationEvent *)event;
      NSString * apiName = invocationEvent.invocationName;
      if([apiName isEqualToString:@"setOptionMenu"] || [apiName isEqualToString:@"showOptionMenu"] )
{
          NSLog(@"wwww");
      }
  }
```

1.9.3. HarmonyOS NEXT 常见问题

本文介绍接入 HarmonyOS NEXT 过程中常见的问题及相应的解决方案。

鸿蒙是否支持 H5 调用相册、指纹、人脸等原生功能？

解答：注入 JSAPI 后即可在接入鸿蒙中通过 H5 调用相册、指纹、人脸等原生功能。

1.9.4. RPC 请求异常

如果通过 RPC 请求进行资源调用的过程中出现异常，请参考 无线保镖结果码说明 进行排查。