# IOS平台5+SDK技术白皮书

## 一 5+ SDK 集成

### 一 预备

#### 1 预备环境

IOS开发环境 Mac os ，Xcode 7.2或以上版本

下载HTML 5+ 离线打包SDK （http://ask.dcloud.net.cn/article/103）

#### 2 SDK 目录说明

HBuilder-Hello：离线打包演示应用

HBuilder-Integrate： 插件开发和5+SDK集成示例应用

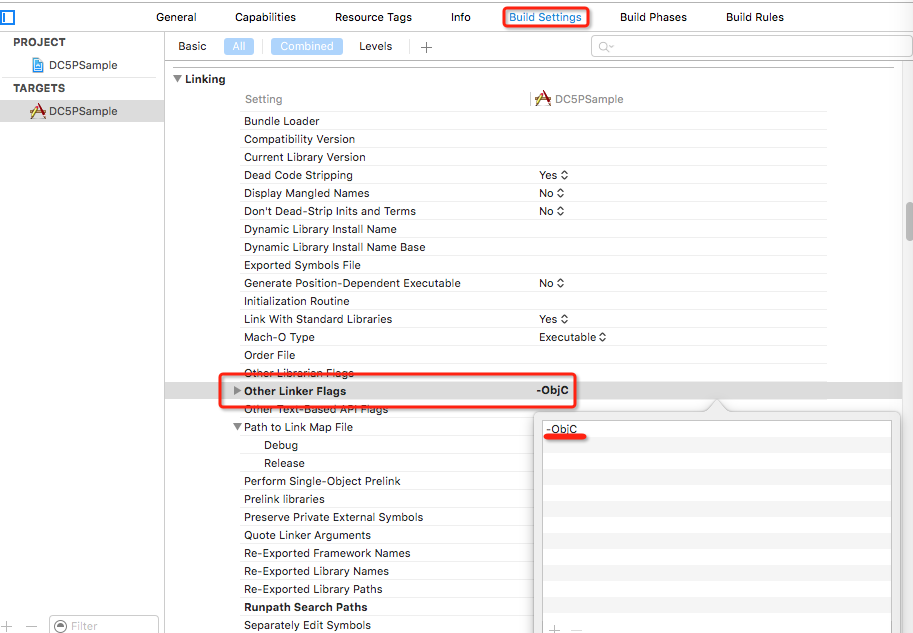
Feature-iOS.xls：iOS平台各扩展Feature API对应的库文件列表；

**注意：HBuilder运行环境相关库文件已全部包含在HBuilder-Hello演示应用中**

#### 3 工程添加5+SDK必须的配置

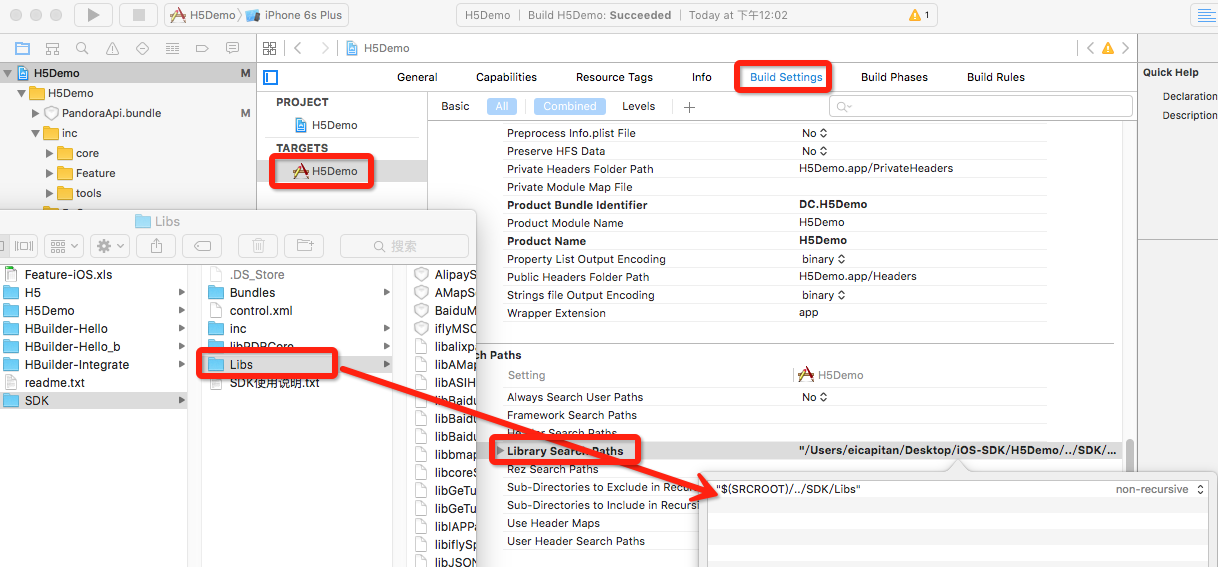
##### 1 引入5+SDK静态库

（1）首先在工程的Build Setting -> Linking -> Other Linker Flags 双击该项在打开的下拉列表中添加“-ObjC”

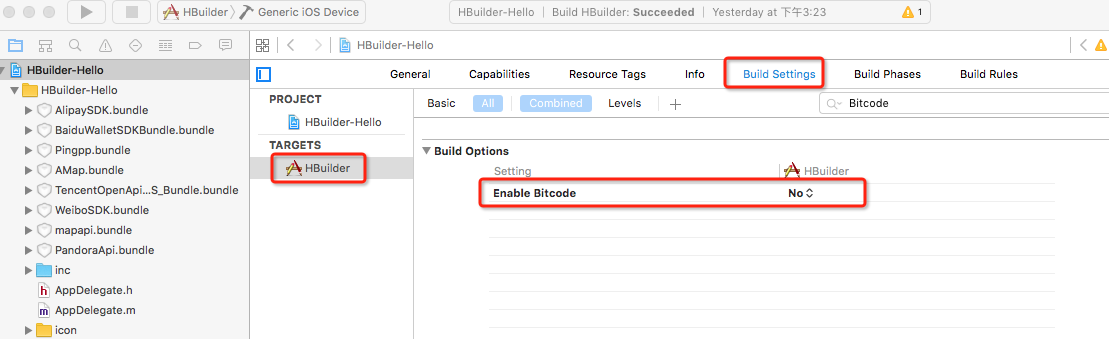


（2）然后在SDK/libs目录下选中要添加的库文件拖动到Xcode工程内

（3）设置静态库搜索路径，在工程的target->Build Settings->Library Search Paths,双击打开下拉列表，鼠标左键按住静态库所在目录，拖到下拉列表内即可



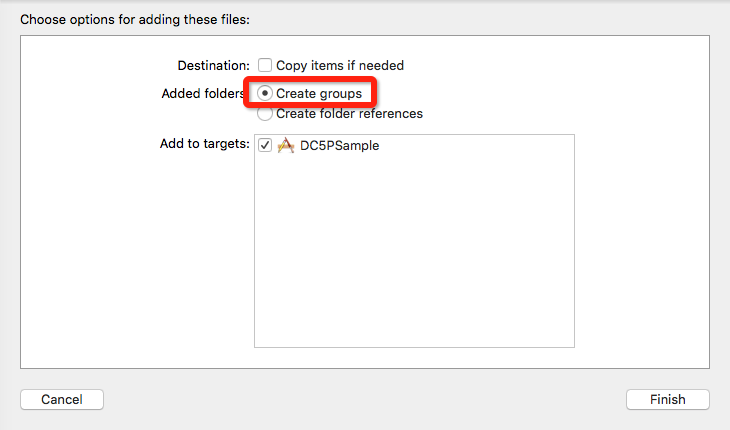
(4)设置Enable BitCode 在工程里 target->Build Settings- 搜索框中输入 Bitcode 将Enable Bitcode 设置为 NO



##### 2 引入5+SDK头文件

打开5+SDK所在目录，拖动inc目录到Xcode工程

文件夹类型选择“Create Groups”



##### 3 引入5+SDK资源文件

PandoraApi.bundle文件：5+SDK所需要的资源，此文件必须添加否则5+SDK将无法正常运行

control.xml文件：文件中配置了默认启动应用的APPID，如使用离线打包方式则需要添加此文件，如使用Widget或者Webview方式集成则不需要添加此文件

<key>NSAppTransportSecurity</key> <dict>     <key>NSAllowsArbitraryLoads</key>     <true/> </dict>

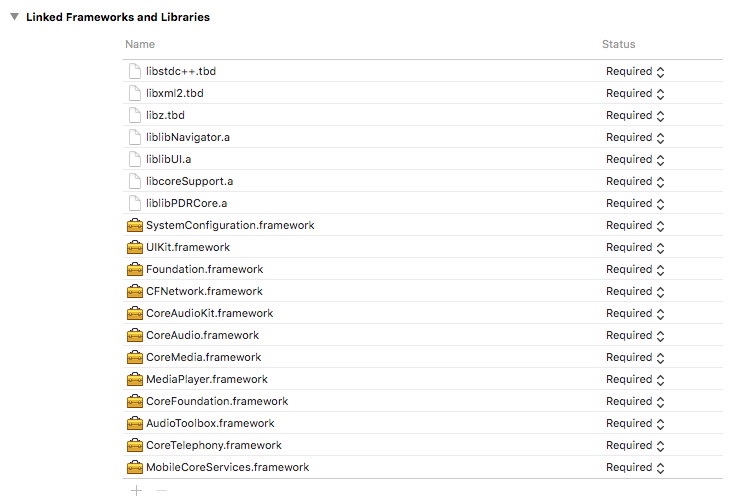
##### 4 info.plist文件配置

在Info.plist中添加App Transport Security Settings类型Dictionary。

　　在App Transport Security Settings下添加Allow Arbitrary Loads类型Boolean,值设为YES

##### 5 添加系统库

在General -> Linked Framewoks and Libraries 添加依赖库



##### 6 添加5+ WebApp

如是离线打包工程则需要将应用资源拷贝到Xcode工程内，并需要同时修改control.xml文件

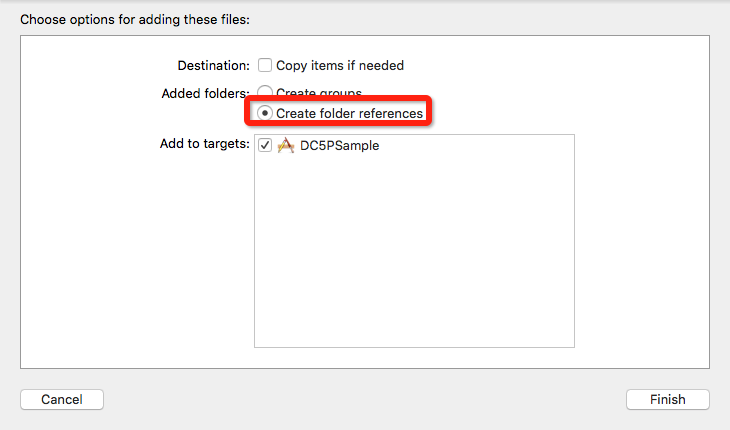
（1）拷贝应用资源到Xcode工程

离线打包需要将应用资源拷贝到Xcode工程内，

注意:Pandora目录为folder references引入方式，修改后不会自动更新,需要在模拟器或设备删除应用后再重新运行 注意:拷贝到Pandora目录的www目录中manifest.json需要删除注释

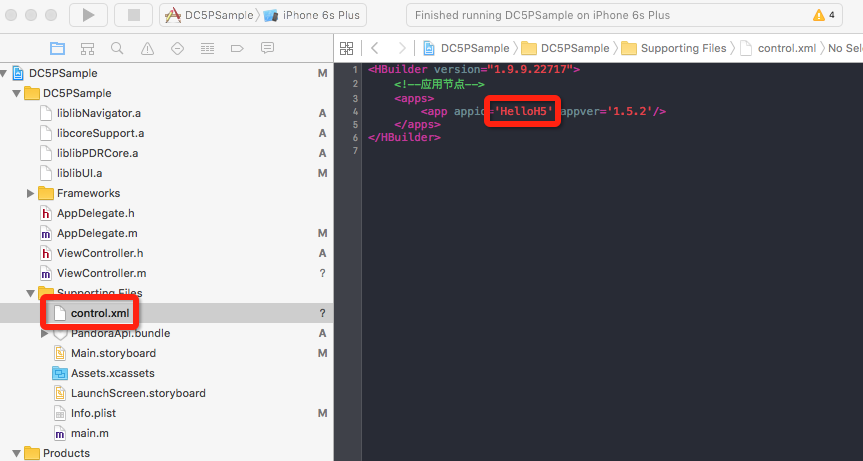
在工程目录下创建 “Pandora/apps/[appid]/www”目录，并将所有应用资源拷贝到其下的www目录中。

拖动“Pandora”目录到Xcode工程内，并选择“Create folder referneces”



（2）修改control.xml文件

打开工程目录下的control.xml文件，修改appid值



### 二 集成

如果需要在Xcode工程内引入5+SDK，可以使用Widget集成方式或者使用Webview集成方式，Widget集成方式支持所有的H5+扩展功能，WebView集成功能支持除plus.webview API外所有的扩展功能。

在使用中，如果需要显示多个H5页面，建议使用Widget集成方式，如果只有一个H5页面，建议使用WebView集成方式

#### 集成5+SDK必须要引入的库文件和资源文件

资源文件：

|  |  |
| --- | --- |
| 文件名 | 文件所在路径 |
| PandoraApi.bundle | SDK/Bundles/ |

注意：如使用离线打包还需要引入control.xml文件并配置默认启动的APPID

5+静态库文件

|  |  |
| --- | --- |
| 文件名 | 文件所在路径 |
| libcoreSupport.a | SDK/Libs/ |
| liblibPDRCore.a | SDK/Libs/ |
| liblibUI.a | SDK/Libs/ |

系统库

JavaScriptCore.framework

WebKit.framework

CoreTelephony.framework

MobileCoreServices.framework

SystemConfiguration.framework

MediaPlayer.framework

AudioToolbox.framework

Security.framework

QuartzCore.framework

CFNetwork.framework

Foundation.framework

CoreFoundation.framework

CoreGraphics.framework

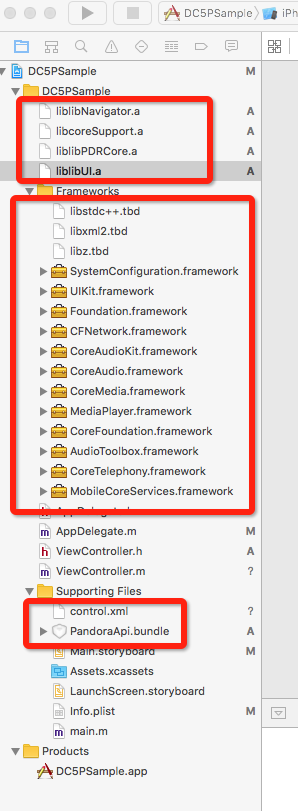
UIKit.framework

libstdc++.6.tbd

libxml2.tbd

libz.tbd

libsqlite3.0.tbd



#### WebApp集成方式

1 设置5+SDK运行模式

在工程的AppDelegate类的添加代码，当应用启动时设置5+SDK的运行模式

-(BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions

{

// 设置当前SDK运行模式

return [PDRCore initEngineWihtOptions:launchOptions withRunMode:PDRCoreRunModeAppClient];

}

1. 在应用退出时添加销毁5+SDK运行环境

- (void)applicationWillTerminate:(UIApplication \*)application

{

// Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.

[PDRCore destoryEngine];

}

1. 启动5+运行环境

-(void)Start5pEngineAsWidget

{

// 设置WebApp所在的目录，该目录下必须有mainfest.json

NSString\* pWWWPath = [[[NSBundle mainBundle] bundlePath] stringByAppendingPathComponent:@"Pandora/apps/HelloH5/www"];

// 如果路径中包含中文，或Xcode工程的targets名为中文则需要对路径进行编码

//NSString\* pWWWPath2 = (NSString \*)CFURLCreateStringByAddingPercentEscapes( kCFAllocatorDefault, (CFStringRef)pTempString, NULL, NULL, kCFStringEncodingUTF8 );

// 设置5+SDK运行的View

[[PDRCore Instance] setContainerView:\_containerView];

// 传入参数可以在页面中通过plus.runtime.arguments参数获取

NSString\* pArgus = @"id=plus.runtime.arguments";

// 启动该应用

pAppHandle = [[[PDRCore Instance] appManager] openAppAtLocation:pWWWPath withIndexPath:@"index.html" withArgs:pArgus withDelegate:nil];

// 如果应用可能会重复打开的话建议使用restart方法

//[[[PDRCore Instance] appManager] restart:pAppHandle];

}

#### 3 WebView集成方式

1 设置5+SDK运行模式

在工程的AppDelegate类的添加代码，当应用启动时设置5+SDK的运行模式

-(BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions

{

// 设置当前SDK运行模式

return [PDRCore initEngineWihtOptions:launchOptions withRunMode: PDRCoreRunModeWebviewClient];

}

1. 在应用退出时添加销毁5+SDK运行环境

- (void)applicationWillTerminate:(UIApplication \*)application

{

// Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.

[PDRCore destoryEngine];

}

1. 启动5+运行环境

-(void)Start5pEngineAsWebView

{

PDRCore\* pCoreHandle = [PDRCore Instance];

if (pCoreHandle != nil)

{

// 单页面集成时可以设置打开的页面是本地文件或者是网络路径

NSString\* pFilePath = [NSString stringWithFormat:@"file://%@/%@", [NSBundle mainBundle].bundlePath, @"Pandora/apps/HelloH5/www/plugin.html"];

// 如果路径中包含中文，或Xcode工程的targets名为中文则需要对路径进行编码

//NSString\* pFilePath = (NSString \*)CFURLCreateStringByAddingPercentEscapes( kCFAllocatorDefault, (CFStringRef)pTempString, NULL, NULL, kCFStringEncodingUTF8 );

CGRect StRect = CGRectMake(0, 20, self.view.frame.size.width, self.view.frame.size.height - 20);

appFrame = [[PDRCoreAppFrame alloc] initWithName:@"WebViewID1" loadURL:pFilePath frame:StRect];

// 单页面运行时设置Document目录

NSString\* pStringDocumentpath = [NSString stringWithFormat:@"%@/Pandora/apps/HelloH5/www/", [NSSearchPathForDirectoriesInDomains(NSLibraryDirectory, NSUserDomainMask, YES) objectAtIndex:0]];

[pCoreHandle.appManager.activeApp.appInfo setWwwPath:pStringDocumentpath];

[pCoreHandle.appManager.activeApp.appWindow registerFrame:appFrame];

[\_containerView addSubview:appFrame];

}

}

#### 扩展插件的配置

所有扩展插件配置都需要配置5+必须的基础库，配置方法请参考前文，这里不再重复说明。

具体插件配置需要的文件请参考Feature-iOS.xls文件

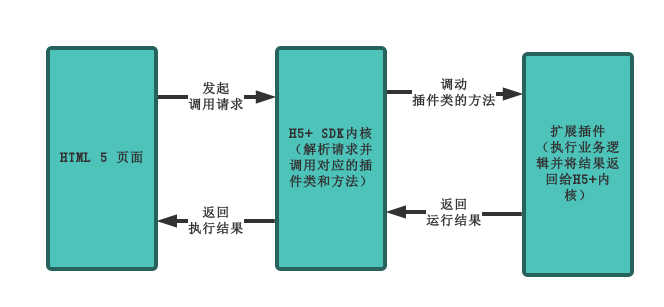
## 二 扩展

5+SDK支持用户自行编写扩展插件扩充5+SDK的功能，通常的一个扩展插件分为两部分：

JS代码部分：在H5页面中执行，触发扩展插件的原生代码方法。

原生代码部分：按照5+扩展插件接口规范编写代码，由在页面中运行的插件JS方法触发调用，并将执行结果返回给JS层。

### 扩展插件的开发

插件的JS代码首先要定义一个插件的JS名，5+内核会根据注册的插件JS别名和原生类名的对应关系找到对应的原生类。并触发对应的方法。

#### JS代码的编写：

5+内核是根据插件在JS层的别名，在配置文件中找到对应的原生类名，动态生成原生类对象，同时根据JS层传入的原生方法名，触发对应的原生插件方法。

5+内核会管理扩展插件的生命周期，所有的5+插件在运行期只有唯一实例，开发者在实现扩展插件时不要销毁自己的实例对象。

插件的方法根据返回结果的方式不同分为同步方法和异步方法，

异步方法：异步方法在调用后不会阻塞后续JS代码的执行，页面中需要设置回调方法来获取原生层的执行结果。

同步方法：同步方法在调用时会阻塞后续JS代码的执行，需要等待原生方法执行完才能继续执行，JS层可直接获取到原生层的执行结果。

异步方法的写法：

要异步触发原生层的方法需要在5+页面中调用方法”plus.bridge.exec()”

plus.bridge.exec(String PluginJSName, String PluginFunName, Array Argyments)

参数说明：

PluginJSName：（String）扩展插件的JS别名，5+内核根据这个参数找到对应的原生类

PluginFunName：（String）要调用的原生类的方法名

Argyments：（Array）列表对象，原生方法中需要的参数，原生层获取参数的顺序和JS层传入的顺序一致。

使用方法：

JS 层如果需要获取原生层的执行结果，可以调用5+提供plus.bridge.callbackid方法对回调函数进行封装，并将封装后生成的ID作为参数传递给原生层。原生层在方法执行完成后同样通过5+提供的原生API触发页面中对应的回调函数获取执行结果。

// JS 层代码示例

function H5PAsyncCallFunction(arguments,successCallback, errorCallback)

{

var success = typeof successCallback !== 'function' ? null : function(args)

{

successCallback(args);

},

fail = typeof errorCallback !== 'function' ? null : function(code)

{

errorCallback(code);

};

callbackID = plus.bridge.callbackId(success, fail);

return plus.bridge.exec(“PluginJSName”, "PluginTestFunction", [callbackID, arguments]);

} }

// 原生层代码

- (void)PluginTestFunction:(PGMethod\*)commands

{

if ( commands ) {

// CallBackid 异步方法的回调id，H5+ 会根据回调ID通知JS层运行结果成功或者失败

NSString\* cbId = [commands.arguments objectAtIndex:0];

// 用户的参数会在第二个参数传回

NSString\* pArgument1 = [commands.arguments objectAtIndex:1];

// 运行Native代码结果和预期相同，调用回调通知JS层运行成功并返回结果

// PDRCommandStatusOK 表示触发JS层成功回调方法

// PDRCommandStatusError 表示触发JS层错误回调方法

// 如果方法需要持续触发页面回调，可以通过修改 PDRPluginResult 对象的keepCallback 属性值来表示当前是否可重复回调， true 表示可以重复回调 false 表示不可重复回调 默认值为false

PDRPluginResult \*result = [PDRPluginResult resultWithStatus:PDRCommandStatusOK messageAsString:pArgument1];

// 通知JS层Native层运行结果

[self toCallback:cbId withReslut:[result toJSONString]];

}

}

同步方法的写法：

要同步触发原生层方法需要在5+页面中调用plus.bridge.execSync方法触发原生层方法的执行。

plus.bridge.execSync(String PluginJSName, String PluginFunName, Array Argyments)

参数说明：

PluginJSName：（String）扩展插件的JS别名，5+内核根据这个参数找到对应的原生类

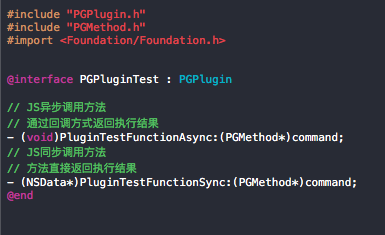
PluginFunName：（String）要调用的原生类的方法名

Argyments：（Array）列表对象，原生方法中需要的参数，原生层获取参数的顺序和JS层传入的顺序一致。

扩展方法JS代码可以通过在HTML页面中引用或原生代码注册两种方式加入到页面中。页面引用JS代码需要注意的是扩展方法需要在页面触发plusready事件时候才能调用。如使用5+原生的API注册JS，5+内核会给每一个支持5+扩展的页面注入该插件的JS，页面中不需要再重复引用，同样扩展方法需要在页面触发plusready事件之后才能调用。（加一个跳转跳到原生注册插件）

#### 原生代码接口规范：

扩展插件必须继承自PGPlugin类，扩展方法只有一个参数类型为PGMethod对象，异步方法的返回值类型为void，同步方法的返回值为NSData\*类型。



参数获取：

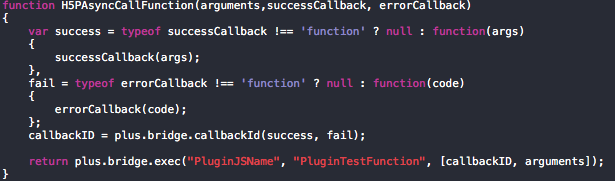
JS层传入的参数通过PGMethod对象的argyments属性获取，arguments属性为NSArray类型，参数获取的顺序和JS层传入的顺序一致。

结果返回：

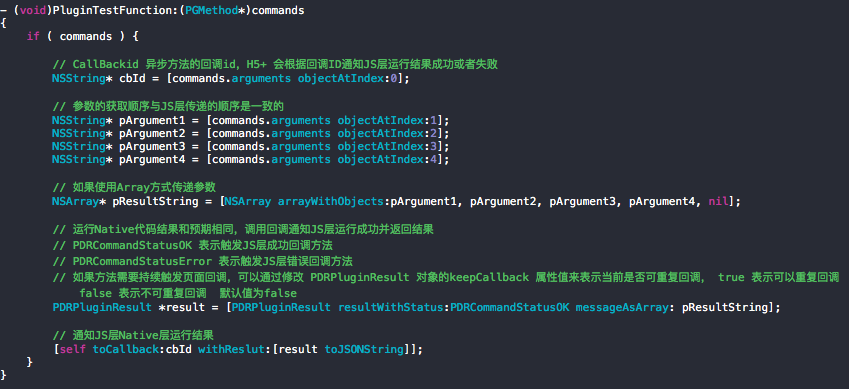
5+为返回插件的运行结果提供了一条封装好的通道，

JS层：通过调用plus.bridge.callbackid方法将成功和失败的回调方法进行封装，并返回一个ID。在调用plus.bridge.exec方法时将返回的回调ID作为参数传递给原生方法。

JS代码示例



原生代码示例



#### 向5+内核注册扩展插件：

通过配置注册插件：

在Xcode工程中打开PandoraAPI.bundle/feature.plist文件，添加一项key值为扩展插件的JS别名，类型为Dictionary。

在该项下添加“class”项，类型为String值为扩展插件的原生类名，再添加“globle”项，类型为Boolean值为NO，如下图所示



通过代码注册插件:

也可以通过5+原生API注册扩展插件

PDRCore类提供了接口用来注册扩展插件

注意：插件注册必须在5+应用启动或者5+页面创建之前调用

接口说明：

- (int)regPluginWithName:(NSString\*)pluginName

impClassName:(NSString\*)impClassName

type:(PDRExendPluginType)pluginType

javaScript:(NSString\*)javaScript;

注册第三方扩展的插件

参数说明：

pluginName 插件名称JS文件中定义的名字

impClassName 插件对应的实现类名

pluginType 插件类型 详情: `PDRExendPluginType`

javaScript js实现 为javascript文本

返回值：

int 0 成功

- (int)regPluginWithName:(NSString\*)pluginName

impClassName:(NSString\*)impClassName

type:(PDRExendPluginType)pluginType

javaScriptPath:(NSString\*)javaScript;

注册第三方扩展的插件

参数说明：

pluginName 插件名称JS文件中定义的名字

impClassName 插件对应的实现类名

pluginType 插件类型 详情: `PDRExendPluginType`

javaScript js实现 为javascript文件路径

返回值：

int 0 成功

代码请参考HBuilder-Integrate工程

### 扩展插件监听系统消息

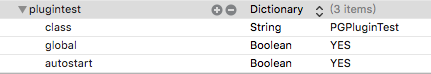
PGPlugin类可以监听系统消息，扩展插件可以通过实现消息监听方法获取应用运行事件



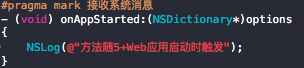
### 扩展插件启动初始化

有些扩展插件需要在应用启动是进行一些处理，如推送插件需要在应用启动时向APS服务器申请设备推送的token。插件有随应用启动的需求可按照如下步骤配置，可实现插件随5+WebApp启动时创建插件并触发监听方法。

1. info.plist文件中的插件项添加autostart属性，类型为Boolean值为YES。global属性值设置为true



1. 扩展插件实现- (void)onAppStarted:(NSDictionary\*)options方法



**注意：该方法会在使用WebApp集成方式集成5+SDK时触发，使用Webview方式集成时不会触发。**

## 四 常见问题

1. **H5页面如何同原生代码进行交互**

有两种方式可用来进行页面代码和原生代码交互

1 编写5+扩展插件，5+扩展插件有封装好的接口供交互,实现方法可参考

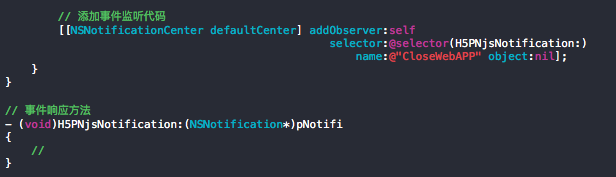
2 在页面中使用NJS代码发送消息，在原生代码中添加消息监听实现H5页面通知原生代码。在原生代码中通过获取PDRCoreAppFrame对象执行JS代码的方式传递参数到H5页面中。

NJS代码：

var notiClass = plus.ios.importClass("NSNotificationCenter");

notiClass.defaultCenter().postNotificationNameobject("CloseWebAPP",null);

原生代码：



1. **原生代码如何监听页面加载事件**

在代码中添加对以下几个事件的监听可获取页面加载状态。

///页面加载完成通知

extern NSString \*const PDRCoreAppFrameDidLoadNotificationKey;

///页面关闭通知

extern NSString \*const PDRCoreAppFrameDidCloseNotificationKey;

///页面将要加载通知

extern NSString \*const PDRCoreAppFrameWillLoadNotificationKey;

///页面开始加载通知

extern NSString \*const PDRCoreAppFrameStartLoadNotificationKey;

///页面加载失败通知

extern NSString \*const PDRCoreAppFrameLoadFailedNotificationKey;

3