WebView

目录结构：

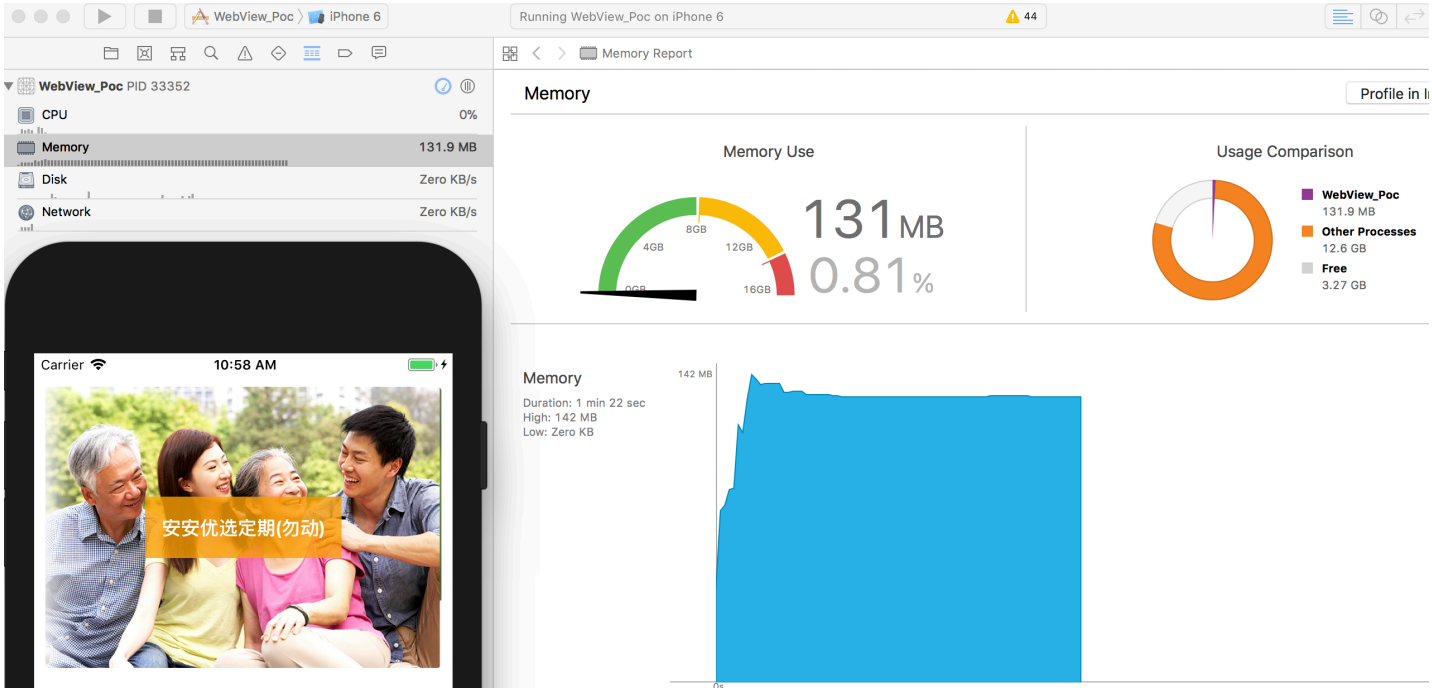
一.背景介绍

随着App业务的扩展和追求发布的及时性，App使用WebView承载业务成为常态。如活动页，使用H5能快速发布，避免了耗时的审核流程。UIWebView自iOS2就有，iOS8出现了WKWebView，毫无疑问WKWebView将逐步取代UIWebView。通过以往使用发现UIWebView占用过多内存，且内存峰值更是夸张。WKWebView内置于WebKit库中，采用跨进程方案，网页加载速度较UIWebView有较大提升。

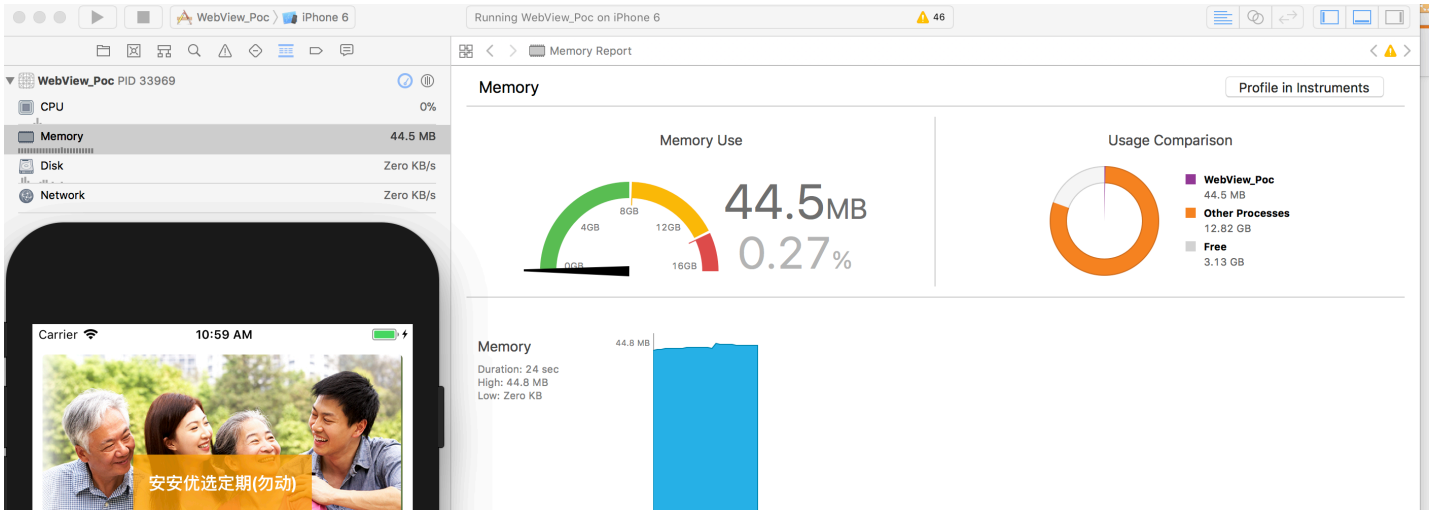
WKWebView的几点优势：

- 更多的支持HTML5的特性
- 官方宣称的高达60FPS的滚动刷新率以及内置手势
- Safari相同的JavaScript引擎
- 将UIWebViewDelegate与UIWebView拆分成了14类、3个协议
- 增加了加载进度属性：estimatedProgress

下图为UIWebView加载产品详情页面的内存消耗



下图为WKWebView加载产品详情页面的内存消耗



二.WebView加载过程

打开一个 H5 页面，大概加载过程如下：

- 初始化 webview
- 请求页面
- 下载数据
- 解析HTML
- 请求 JS/CSS 资源
- dom 渲染
- 解析 JS 执行
- JS 请求数据
- 解析渲染

- 下载渲染图片

一些简单的页面可能没有 JS 请求数据 这一步，但大部分功能模块应该是有的，根据当前用户信息，JS 向后台请求相关数据再渲染，是常规开发方式。

一般页面在 dom 渲染后能显示雏形，在这之前用户看到的都是白屏，等到下载渲染图片后整个页面才完整显示。

三.UIWebView与WKWebView对比

UIWebView使用WebKit内核，Javascript引擎使用JavascriptCore。在iOS8之前，Apple只允许开发者使用UIWebView加载Web页面，并且不允许开发者自行封装Javascript引擎。UIWebView以其内存占用极高、渲染超慢、Javascript执行效率低、存在leaks的风险著称

WKWebView同样使用WebKit内核，但是Javascript引擎使用Nitro。WKWebView拥有60FPS的渲染、刷新速度，内存占用较少，Javascript执行效率是JavascriptCore的数倍，并且WKWebView leaks较少。

创建UIWebView代码：

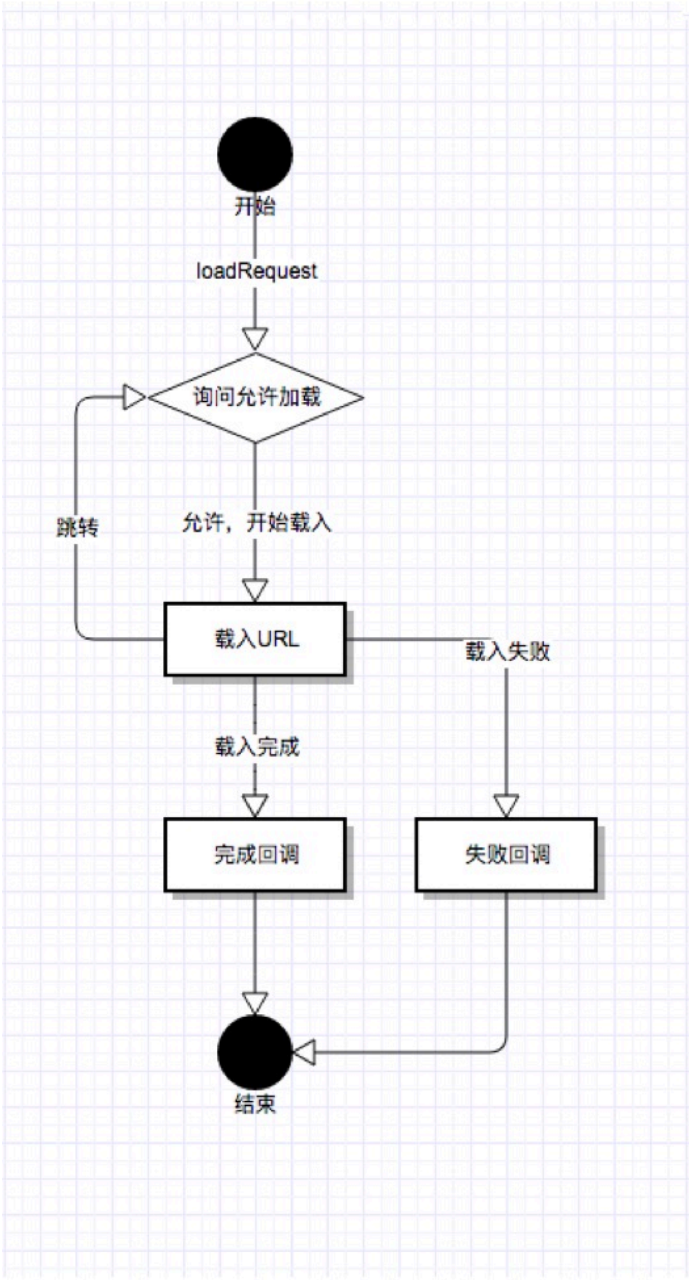
```
// 1.创建UIWebView
UIWebView *webView = [[UIWebView alloc] init];
// 2.创建请求
NSURLRequest *url = [NSURLRequest requestWithURL:[NSURL URLWithString:@"https://ww
w.baidu.com"]];
// 3.加载请求
[webView loadRequest:url];
// 4.视图添加
[self.view addSubview:webView];
```

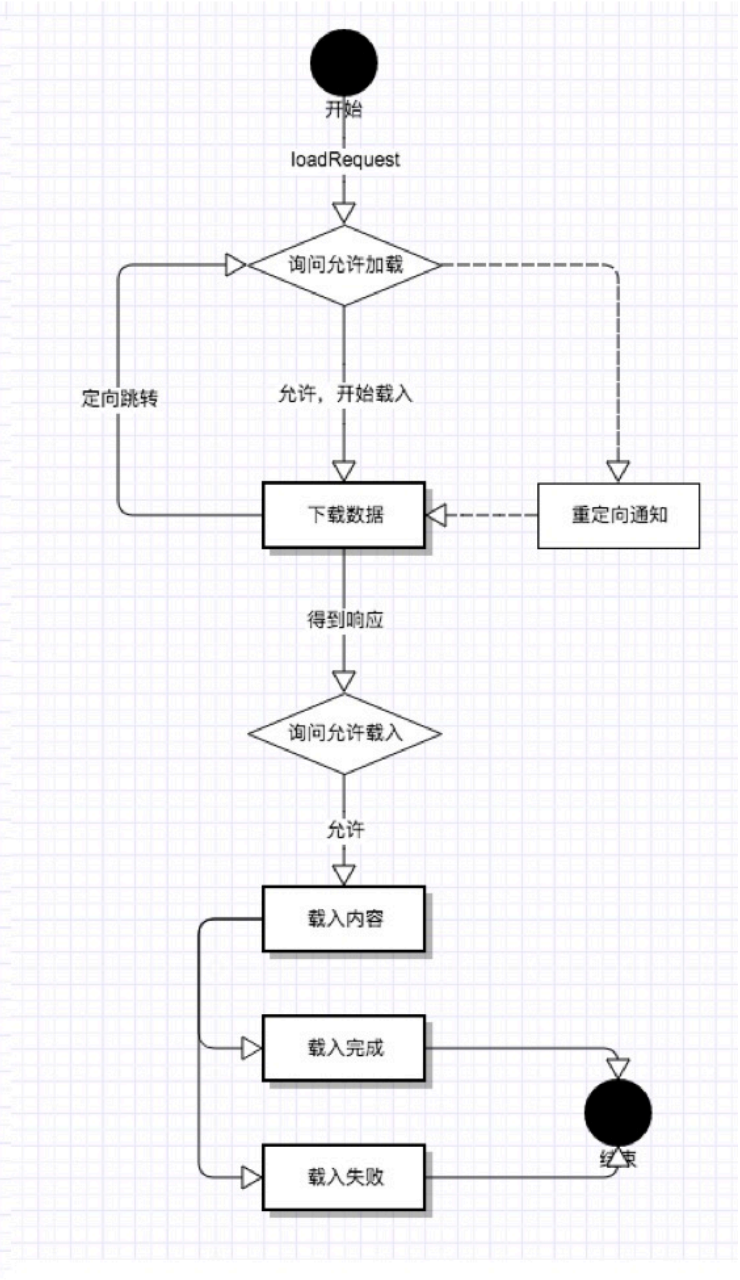
创建WKWebView代码:

```
// 1.创建WKWebView
WKWebView *webView = [[WKWebView alloc] init];
// 2.创建请求
NSURLRequest *url = [NSURLRequest requestWithURL:[NSURL URLWithString:@"https://ww
w.baidu.com"]];
// 3.加载请求
[webView loadRequest:url];
// 4.视图添加
[self.view addSubview:webView];
```

流程上有一定的区别，WKWebView比UIWebView多了一个询问过程，在服务器响应请求之后会询问是否载入内容到当前Frame，在控制上会比UIWebView粒度更细一些。

如下图左边是UIWebView，右边是WKWebView：





以上流程的控制主要是通过Protocol去实现，WKWebView的代理协议为WKNavigationDelegate，对比UIWebViewDelegate 首先跳转询问，就是载入URL之前的一次调用，询问开发者是否加载当前URL，UIWebView只有一次询问，就是请求之前的询问，而WKWebView在URL下载完毕之后还会发一次询问，让开发者根据服务器返回的Web内容再做一次确定。

```
#pragma mark - UIWebViewDelegate
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)re
quest navigationType:(UIWebViewNavigationType)navigationType
{
    // 请求前调用，询问是否加载
    return YES;
}

#pragma mark - WKNavigationDelegate
- (void)webView:(WKWebView *)webView decidePolicyForNavigationAction:(WKNavigation
Action *)navigationAction decisionHandler:(void (^)(WKNavigationActionPolicy))deci
sionHandler
{
    // 请求前调用，询问是否加载
    decisionHandler(WKNavigationActionPolicyAllow);
}

- (void)webView:(WKWebView *)webView decidePolicyForNavigationResponse:(WKNavigati
onResponse *)navigationResponse decisionHandler:(void (^)(WKNavigationResponsePoli
cy))decisionHandler
{
    // 返回响应前调用 并且已经接收到响应 询问是否加载
    decisionHandler(WKNavigationResponsePolicyAllow);
}
```

允许加载后，WebView就开始加载指定URL的内容，在加载之前会调用一次 ‘开始下载’ 回调，通知开发者WebView已经开始下载

```
#pragma mark - UIWebViewDelegate
- (void)webViewDidStartLoad:(UIWebView *)webView
{
    // 页面开始加载时调用
}

#pragma mark - WKNavigationDelegate
- (void)webView:(WKWebView *)webView didStartProvisionalNavigation:(null_unspecified WKNavigation *)navigation
{
    // 页面开始加载时调用
}

- (void)webView:(WKWebView *)webView didCommitNavigation:(null_unspecified WKNavigation *)navigation
{
    // 内容开始返回时调用
}
```

页面加载成功之后，UIWebView会直接载入视图并调用载入成功回调，而WKWebView会发询问，确定内容被允许之后再载入视图。

```
#pragma mark - UIWebViewDelegate
- (void)webViewDidFinishLoad:(UIWebView *)webView
{
    // 页面加载成功时调用
}

#pragma mark - WKNavigationDelegate
- (void)webView:(WKWebView *)webView didFinishNavigation:(null_unspecified WKNavigation *)navigation
{
    // 页面加载成功时调用
}
```

页面发生错误则进入错误回调。

```
#pragma mark - UIWebViewDelegate
- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error
{
    // 页面加载失败时调用
}

#pragma mark - WKNavigationDelegate
- (void)webView:(WKWebView *)webView didFailProvisionalNavigation:(null_unspecifie
d WKNavigation *)navigation withError:(NSError *)error
{
    // 启动时加载数据发生错误就会调用
}
- (void)webView:(WKWebView *)webView didFailNavigation:(null_unspecified WKNavigat
ion *)navigation withError:(NSError *)error
{
    // 当一个正在提交的页面在跳转过程中出现错误时调用
}
```

而除此之外，WKWebView 对比 UIWebView 有较大差异的地方有几点。

1、 WKWebView多了一个重定向通知，在收到服务器重定向消息并且跳转询问允许之后，会回调重定向方法，这点是UIWebView没有的，在UIWebView之上需要验证是否重定向，得在询问方法验证header信息。

```
#pragma mark - WKNavigationDelegate
- (void)webView:(WKWebView *)webView didReceiveServerRedirectForProvisionalNavigat
ion:(null_unspecified WKNavigation *)navigation
{
    // 重定向时调用
}
```

2、 因为WKWebView是跨进程的方案，当WKWebView进程退出时，会对主进程做一次方法回调。

PS：该方法是在iOS9之后才出现，而我们最低支持版本是iOS8，所以还得考虑iOS8下WKWebView进程崩溃问题，另外该方法也不是很靠谱，不一定所有崩溃情况都会触发回调。

```
- (void)webViewWebContentProcessDidTerminate:(WKWebView *)webView API_AVAILABLE(ma
cosx(10.11), ios(9.0))
{
    // 当WebView的网页内容被终止时调用
}
```

3、 https证书自定义处理（证书认证代理不保证会回调）

```
- (void)webView:(WKWebView *)webView didReceiveAuthenticationChallenge:(NSURLAuthe
nticationChallenge *)challenge completionHandler:(void (^)(NSURLSessionAuthChallen
geDisposition disposition, NSURLCredential * _Nullable credential))completionHandl
er
{
    if (trusted) {
        [challenge.sender useCredential:[NSURLCredential credentialForTrust:challe
nge.protectionSpace.serverTrust] forAuthenticationChallenge:challenge];
    } else {
        [challenge.sender cancelAuthenticationChallenge:challenge];
    }
}
```

4、 新增WKUIDelegate协议，该协议包含一些UI相关的内容。

在UIWebView中Alert、Confirm、Prompt等视图是直接可执行的，在WKWebView中需要通过WKUIDelegate协议接收通知，然后通过iOS原生执行。

```
#pragma mark - WKUIDelegate
-(WKWebView *)webView:(WKWebView *)webView createWebViewWithConfiguration:(WKWebVi
ewConfiguration *)configuration forNavigationAction:(WKNavigationAction *)navigati
onAction windowFeatures:(WKWindowFeatures *)windowFeatures
{
    // 创建一个新的WebView
}

-(void)webView:(WKWebView *)webView runJavaScriptTextInputPanelWithPrompt:(NSStrin
g *)prompt defaultText:(nullable NSString *)defaultText initiatedByFrame:(WKFrameI
nfo *)frame completionHandler:(void (^)(NSString * __nullable result))completionHa
ndler
{
    // 输入框
}


-(void)webView:(WKWebView *)webView runJavaScriptConfirmPanelWithMessage:(NSString
 *)message initiatedByFrame:(WKFrameInfo *)frame completionHandler:(void (^)(BOOL
result))completionHandler
{
    // 确认框
}

-(void)webView:(WKWebView *)webView runJavaScriptAlertPanelWithMessage:(NSString *
)message initiatedByFrame:(WKFrameInfo *)frame completionHandler:(void (^)(void))c
ompletionHandler
{
    // 警告框
}
```

其次，WKWebView关闭时的回调通知也在WKUIDelegate协议中。注：该方法也是iOS9才有。

另外，对于类似 'A' 标签 'target=*blank*' *这种情况，会要求创建一个新的WKWebView视图，这个消息的通知回调也在该协议中，不过针对iOS设备在当前一个视图中显示，该标签点击会没反应，所以在视图载入之后会清除掉所有的*blank标记

5、 WKWebView相关类介绍

```
WKBackForwardList：之前访问过的 web 页面的列表，可以通过后退和前进动作来访问到。

WKBackForwardListItem：webview 中后退列表里的某一个网页。

WKFrameInfo：包含一个网页的布局信息。

WKNavigation：包含一个网页的加载进度信息。

WKNavigationAction：包含可能让网页导航变化的信息，用于判断是否做出导航变化。

WKNavigationResponse：包含可能让网页导航变化的返回内容信息，用于判断是否做出导航变化。

WKPreferences：概括一个 webview 的偏好设置。

WKProcessPool：表示一个 web 内容加载池。

WKUserContentController：提供使用 JavaScript post 信息和注入 script 的方法。

WKScriptMessage：包含网页发出的信息。

WKUserScript：表示可以被网页接受的用户脚本。

WKWebViewConfiguration：初始化 webview 的设置。

WKWindowFeatures：指定加载新网页时的窗口属性。

WKWebsiteDataStore：website站点使用各种数据类型，比如：cookies, disk and memory caches,
 and persistent data such as WebSQL,IndexedDB databases, and local storage

WKWebViewConfiguration：webview初始化配置
```

四.JavaScript与Native交互

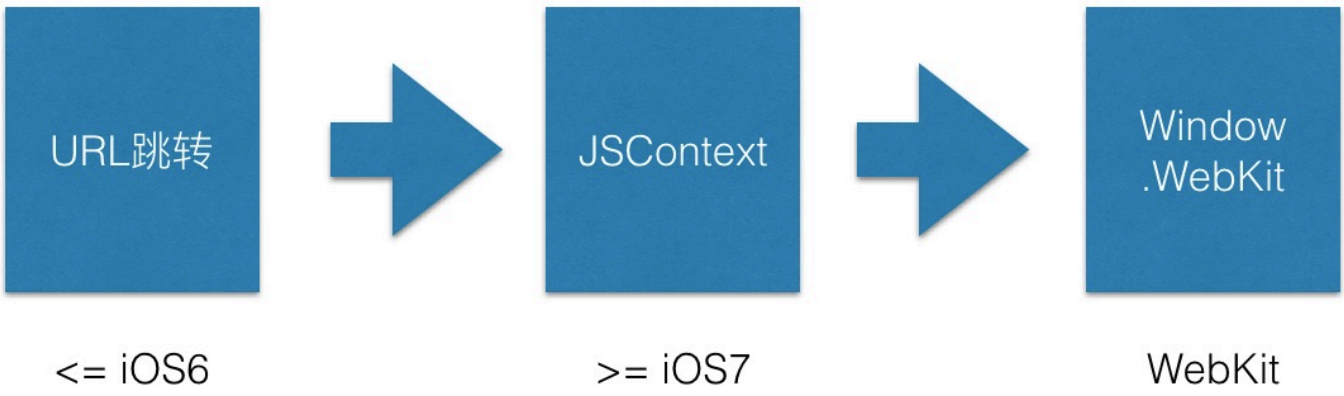
1.Native调用JavaScript

依靠WebView提供的接口实现，WKWebView提供的接口和UIWebView命名上较为类似，区别是WKWebView的这个接口是异步的，而UIWebView是同步接口

```
#pragma mark - UIWebView
NSString *title = [webView stringByEvaluatingJavaScriptFromString:@"document.title
"];

#pragma mark - WKWebView
[wkWebView evaluateJavaScript:@"document.title"
            completionHandler:^(id _Nullable ret, NSError * _Nullable error) {
    NSString *title = ret;
}];
```

2.JavaScript调Native

- iOS6之前，UIWebView是不支持共享对象的，Web端需要通知Native，需要通过修改location.url，利用跳转询问协议来间接实现
- iOS7开始，新增了JavaScriptCore库，内部有JSContext对象可实现共享
- iOS8开始，提供了WKWebView，Web通过window.webkit对象实现共享



1.URL捕获

```
// Javascript
<a href="iosapp://loginPage">跳到登录页面</a>
```

```
// 在URL加载前 捕获需要特殊处理的URL
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)re
quest navigationType:(UIWebViewNavigationType)navigationType{
    NSURL *url = [request URL];
    if ([[url absoluteString] isEqualToString:@"iosapp://loginPage"]) {
        NSString *slug = [url path];
        [self performSegueWithIdentifier:@"loginPage" sender:slug];
        return NO;
    }
    return YES;
}
```

通过NSURLProtocol（通常用来做UIWebView缓存）

```
- (void)startLoading
{
    NSString * url = [[[self request] URL] absoluteString];

    if([url isEqualToString:@"iosapp://loginPage"]) {
          NSString *slug = [url path];
        [self performSegueWithIdentifier:@"loginPage" sender:slug];
    }
}
```

2.通过JavaScriptCore实现

- JSContext 代表JS的执行环境，通过-evaluateScript:方法就可以执行JS代码
- JSValue 封装了JS与ObjC中的对应的类型，以及调用JS的API等
- JSExport 是一个协议，遵守此协议，就可以定义我们自己的协议，在协议中声明的API都会在JS中暴露出来，才能调用

```
// Javascript
function callNative() {
    window.APP.showMessage();
    window.showName('XiaoMing');
}
```

```
@protocol ZAJSExport <JSExport>

// Q1: 此处 @optional还是@required
- (void)showMessage:(NSString *)message;

@end


...
```

```
- (void)webViewDidStartLoad:(JSBridgeWebView *)webView
{

}

- (void)webViewDidFinishLoad:(JSBridgeWebView *)webView
{
    __block typeof(self) weakSelf = self;

     // Q2: 此时获取的JSContext 是否一定是准确的?
     // 获取当前JS运行环境
    self.context = [self.webView valueForKeyPath:@"documentView.webView.mainFrame.
javaScriptContext"];
    NSLog(@"%@", self.jsContext);

    // 注入JS需要的“APP”对象
    self.context[@"APP"] = self;
    self.context.exceptionHandler = ^(JSContext *context, JSValue *exception) {
        NSLog(@"exception:%@", exception);
    };

    self.context[@"showName"] = ^ (NSString *name) {
        dispatch_async(dispatch_get_main_queue(), ^{
          NSString *info = [NSString stringWithFormat:@"Hello, %@",name];
         [weakSelf showMsg:info];
        });
    };

    // context可以直接执行JS代码
   [self.context evaluateScript:@"var squareFunc = function(value) { return value
* 2 }"];
    // 计算正方形的面积方法
    JSValue *square = [self.context evaluateScript:@"squareFunc(num)"];
    // 输出JS调用结果
     NSLog(@"%@", square.toNumber);
}

#pragma mark - ZAJSExport
- (void)showMessage:(NSString *)message
{
    // 子线程
    NSLog(@"current:%@",[NSThread currentThread]);
    // 切换到UI主线程
    dispatch_async(dispatch_get_main_queue(), ^{
        UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"J
S调用了OC" message:message preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction *cancel = [UIAlertAction actionWithTitle:@"确定" style:UIAler
```

```
tActionStyleCancel handler:nil];
        [alert addAction:cancel];
        [self presentViewController:alert animated:YES completion:nil];
    });
}
```

3.WKWebView scriptMessageHandler

绑定共享对象，是通过特定的构造方法实现，参考代码，通过指定 ‘UserContentController’ 对象的 ‘ScriptMessageHandler’ 经过 ‘Configuration’ 参数构造时传入

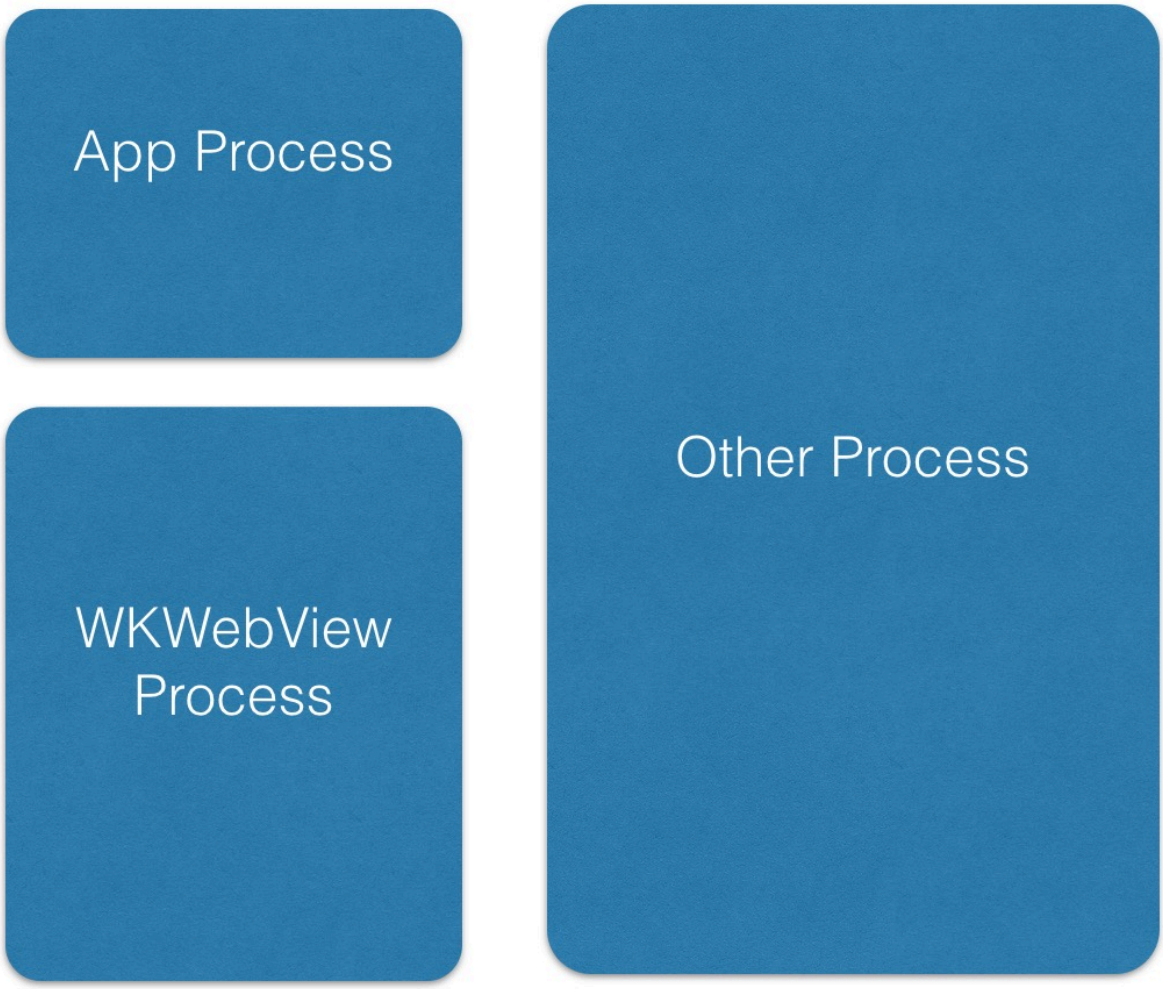
```
// Javascript
function callNative() {
    const params = {
        name: 'XiaoMing',
        gender: 'male',
    };
    window.webkit.messageHandlers.setJsContent.postMessage(params);
}
```

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

    WKUserContentController *userContent = [[WKUserContentController alloc] init];
    [userContent addScriptMessageHandler:id<WKScriptMessageHandler> name:@"setJsCo
ntent"];

    WKWebViewConfiguration *config = [[WKWebViewConfiguration alloc] init];
    config.userContentController = userContent;

    WKWebView *webview = [[WKWebView alloc] initWithFrame:self.view.bounds configu
ration:config];
    NSURLRequest *url = [NSURLRequest requestWithURL:[NSURL URLWithString:@"https:
//www.baidu.com"]];
    [webView loadRequest:url];
    [self.view addSubview:webview];
}


#pragma mark - WKScriptMessageHandler
-(void)userContentController:(WKUserContentController *)userContentController didR
eceiveScriptMessage:(WKScriptMessage *)message {
    if ([message.name isEqualToString:@"setJsContent"]) {
    NSDictionary *body = message.body;
    NSLog(@"name: %@ body: %@", message.name, message.body);
  }
}
```

五.常见坑&填坑

WKWebView是一个新组件，并且是采用跨进程方案，实现了比较好的性能和体验，那同样的，肯定会带来一些问题

WKWebView内存结构



WKWebView 是一个多进程组件，Network Loading 以及 UI Rendering 在其它进程中执行。初次适配 WKWebView 的时候，App 进程内存消耗大幅下降，但是仔细观察会发现Other Process 的内存占用会增加。在一些用 webGL 渲染的复杂页面，使用 WKWebView 总体的内存占用也就是 App Process Memory + Other Process Memory。

1.白屏

在 UIWebView 上当内存占用太大的时候，App Process 会 crash；而在 WKWebView 上当总体的内存占用比较大的时候，WebContent Process 会 crash，从而出现白屏现象。

A、借助 WKNavigtionDelegate

```
// iOS 9以后 WKNavigtionDelegate
- (void)webViewWebContentProcessDidTerminate:(WKWebView *)webView API_AVAILABLE(ma
cosx(10.11), ios(9.0));
```

当 WKWebView 总体内存占用过大，页面即将白屏的时候，系统会调用上面的回调函数，我们在该函数里执行[webView reload](这个时候 webView.URL 取值尚不为 nil）解决白屏问题。在一些高内存消耗的页面可能会频繁刷新当前页面，H5侧也要做相应的适配操作。

B、检测 webView.title 是否为空

在一个高内存消耗的H5页面上 present 系统相机，拍照完毕后返回原来页面的时候出现白屏现象（拍照过程消耗了大量内存，导致内存紧张，WebContent Process 被系统挂起） 另一种现象是 webView.titile 会被置空, 因此，可以在 viewWillAppear 的时候检测 webView.title 是否为空来 reload 页面。

综合以上两种方法可以解决绝大多数的白屏问题。

2.WKWebView Cookie 问题

WKWebView请求头不会自动带上Cookie

```
A、WKWebView loadRequest 前，在 request header 中设置 Cookie，解决首个请求 Cookie 带不上的问题；
WKWebView * webView = [WKWebView new];
NSMutableURLRequest * request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:@"http://h5.qzone.qq.com/mqzone/index"]];

[request addValue:@"skey=skeyValue" forHTTPHeaderField:@"Cookie"];
[webView loadRequest:request];
```

B、通过 document.cookie 设置 Cookie 解决后续页面(同域)Ajax、iframe 请求的 Cookie 问题； 注意：document.cookie()无法跨域设置 cookie

```
WKUserContentController* userContentController = [WKUserContentController new];
WKUserScript * cookieScript = [[WKUserScript alloc] initWithSource: @"document.cookie = 'skey=skeyValue';" injectionTime:WKUserScriptInjectionTimeAtDocumentStart forMainFrameOnly:NO];

[userContentController addUserScript:cookieScript];
```

这种方案无法解决302请求的 Cookie 问题，比如，第一个请求是 www.a.com，我们通过在 request header 里带上 Cookie 解决该请求的 Cookie 问题，接着页面302跳转到 www.b.com，这个时候 www.b.com 这个请求就可能因为没有携带 cookie 而无法访问。当然，由于每一次页面跳转前都会调用回调函数：

```
- (void)webView:(WKWebView *)webView decidePolicyForNavigationAction:(WKNavigationAction *)navigationAction decisionHandler:(void (^)(WKNavigationActionPolicy))decisionHandler;
```

可以在该回调函数里拦截302请求，copy request，在 request header 中带上 cookie 并重新 loadRequest。不过这种方法依然解决不了页面 iframe 跨域请求的 Cookie 问题，毕竟-[WKWebView loadRequest:]只适合

加载 mainFrame 请求。

3.WKWebView不支持NSURLProtocol

WKWebView在独立于App进程之外的进程中执行网络请求，请求数据不经过主进程，因此，在WKWebView上直接使用NSURLProtocol无法拦截请求。

4.修改UserAgent

修改UserAgent要通过写入NSUserDefaults来统一修改

```
  [_webView evaluateJavaScript:@"navigator.userAgent" completionHandler:^(id resul
t, NSError *error) {
    NSString *userAgent = result;
    NSString *newUserAgent = [userAgent stringByAppendingString:@" AnAnApp"];
    NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys:newUserA
gent, @"UserAgent", nil];
    [[NSUserDefaults standardUserDefaults] registerDefaults:dictionary];
    [[NSUserDefaults standardUserDefaults] synchronize];
    if ([_webView respondsToSelector:@selector(setCustomUserAgent:)]) {
      [_webView setCustomUserAgent:newUserAgent];
    }
  }];
```

5. UIWebView中JSContext创建时机

A.在渲染网页时遇到script标签时，就会创建JSContext环境去运行JavaScript代码

B.使用如下方法获取JSContext环境时，这时无论是否遇到script标签，都会去创造出来一个JSContext环境，而且和遇到script标签再创造环境是同一个

```
JSContext *context = [webView valueForKeyPath:@"documentView.webView.mainFrame.jav
aScriptContext"]
```

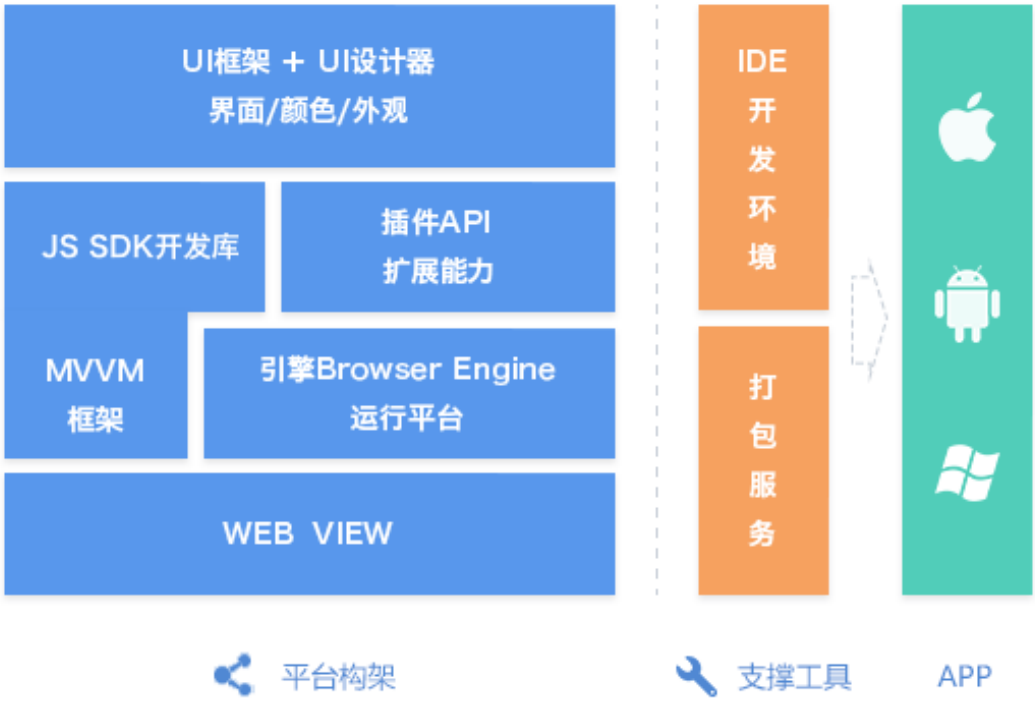
同一个webview的JSContext，在不同时机，根本就不是一个JS上下文对象，地址都不一样。相同的JS，运行在不同的JS环境里，自然效果是完全不一样的。 每次WebView加载一个新Url的时候，都会丢掉旧的JS上下文，重新启用一个新的JS上下文，因此你在webViewDidStartLoad的时候即便使用stringByEvaluatingJavaScriptFromString去注入js，也是把js代码在旧的上下文中执行，当新的js上下文完全不受任何影响，没任何效果。

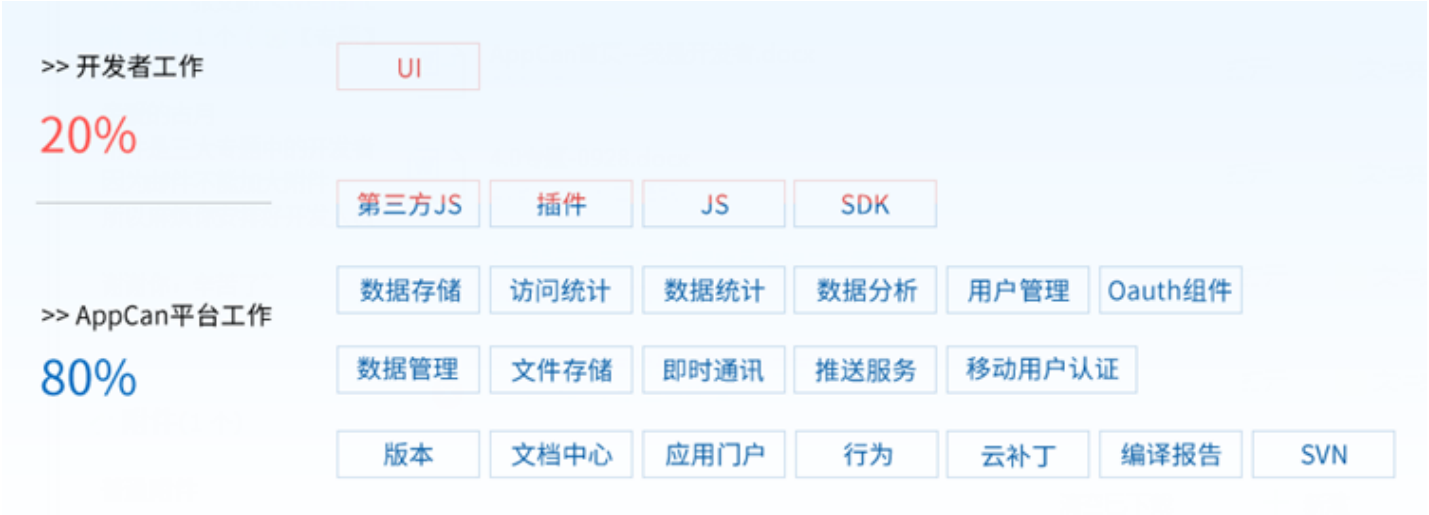
六.Hybird简介

混合开发，也就是 Native+Web 的开发模式，由原生提供统一的API给JS调用，实际的主要逻辑由HTML和JS来完成，最终在webview中显示的，只需要写一套代码即可达到跨平台效果，可直接在浏览器中调试。

Hybird App 的较早实践者是PhoneGap(Cordova)，还有如Titanium、Salama、WeX5、Kerkee和国内的AppCan，项目各有各的实现方式，大致的原理基本相同。

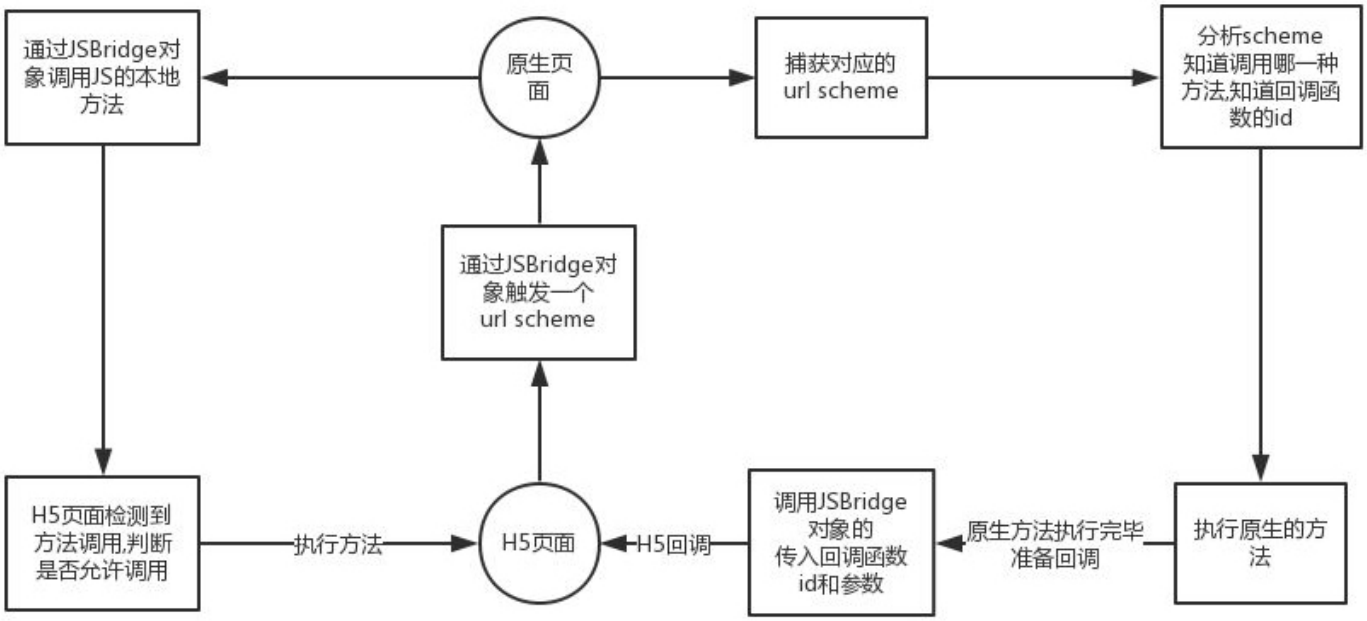
AppCan是基于HTML5技术的Hybird跨平台移动应用开发工具。开发者利用HTML5+CSS3+JavaScript技术，通过AppCan IDE集成开发系统、云端打包器等，快速开发出Android、iOS、WP平台上的移动应用。





AppCan将App底层复杂的原生功能封装在引擎、插件中，开发者仅需调用接口、打包编译，就可以获得原生功能。

开发者可以像开发WebApp一样开发App的视觉UI，以及绝大部分的交互，当需要使用原生功能（如摄像头，计步器等功能）时，需调用官方的API以实现Native的效果。JS和Native的通信方式，已在前文介绍过，使用较多的JSBridge也是基于url scheme去做的封装。



七.参考资料

wkwebkit

UIWebView代码注入时机与姿势

JavaScriptCore与iOS和Android交互

why-use-javascriptcore-in-ios7-if-it-cant-access-a-uiwebviews-runtime

WebFrameLoadDelegate

UIWebView-TS_JavaScriptContext

JSBridge深度剖析