一站式解决WKWebView各类问题

卡洛斯 62

为什么要使用 WKWebView

- UIWebView 在 iOS 12 后被标记为过期了。
- H5 在 UIWebView 和 WKWebView 上的行为不一致，特别在滚动监听上了。另外像微信这样的分享渠道都是使用的 WKWebView，如果还在使用 UIWebView 的话，会导致 H5 的显示效果跟微信上的不一致。
- WKWebView 可以带来执行效率和渲染效率的提升。

为什么要重新造轮子

我们知道 WKWebView 有典型的两个问题：

1. 支持离线包时，ajax body 就会丢失
2. cookie 同步问题

关于 body 丢失的问题

我了解到的解决 body 丢失问题的方式如下：

- H5 在 进行 ajax 请求时，把 body 体放入请求头里，在拦截的时候，再从请求头里拿出 body，然后重新构建新的请求，通过 NSURLSession 发送。但是请求头不适合放入太多数据，而且也需要 H5 侧配合才行。
- H5 ajax 请求是通过 JSAPI 转发到 Native 来发送，这个确实可以解决 body 丢失的问题，但是会让 H5 侧有集成的成本。
- 在请求之前把 url 上的 http/https scheme 替换成自定义的 customscheme，然后返回的 Html 中的子资源链接，使用 src='//xxx.com/a.js' 的形式去加载，这样也会让子资源带上 customscheme，通过这种方式 NSURLProtocol 只用注册和拦截 customscheme，针对 customscheme 丢失 body 信息没有任何影响，然后 H5 ajax 请求，还是可以走 http/https 来请求。这种也可以解决 body 丢失的问题，但是必须要求 H5 侧能够按照规范来，如果 H5 没有历史包袱，这种也是可行的，如果有历史包袱的话，也会有改动的成本。
- 注入 ajax hook js 代码，对所有 XMLHTTPRequest 对象进行 hook，在 open，send 等方法处进行拦截，通过 JSAPI 把 url，数据

转发到 Native 去发送请求。这种可以解决 body 丢失的问题，而且对 H5 ajax 请求是无侵入式的，但是这种方式相关开源实践很少。

这里的话，只有 ajax hook 是无侵入式的，H5 侧没有任何改动成本，我自己的原则也是最小化改动 H5，让 H5 无感知的使用 WKWebView 的能力，所以 KKJSBridge 也是基于 ajax hook 来实现的。

关于 cookie 同步的问题

我们都知道 WKWebView 的 cookie 是单独保存在 WKWebView cookie 里的，而不是我们平时用的 NSHTTPCookieStorage，所以这就会导致 cookie 不同步的问题，一些文章也介绍了很多但是都没有说处理的很全面。这里我列下所有情况下 cookie 同步的问题：

- 首次同步请求的 cookie 同步（NSHTTPCookieStorage 同步到 WKWebView cookie）
- 异步 ajax 的 cookie 同步（NSHTTPCookieStorage 同步到 WKWebView cookie）
- 服务器端（302）重定向/浏览器重定向的 cookie 同步（NSHTTPCookieStorage 同步到 WKWebView cookie）
- 服务器端响应头里的 Set-Cookie 同步（WKWebView cookie 同步到 NSHTTPCookieStorage）
- H5 侧的 document.cookie 同步（WKWebView cookie 同步到 NSHTTPCookieStorage）
- 异步 ajax 响应头里的 Set-Cookie 同步（WKWebView cookie 同步到 NSHTTPCookieStorage）
- cookie HTTPOnly 同步 （NSHTTPCookieStorage 同步到 WKWebView cookie 和 WKWebView cookie 同步到 NSHTTPCookieStorage）

基本上开源的处理方式都只处理到了前四步，后面几个是没有涉及到的，针对剩下的 cookie 同步问题，我们也是需要处理。

所以 KKJSBridge 是为了完全解决上面两个问题而出的新轮子，同时 KKJSBridge 也会提供其他的特性。

KKJSBridge 支持的功能

- 基于 MessageHandler 搭建通信层
- 支持模块化的管理 JSAPI
- 支持模块共享上下文信息
- 支持模块消息转发
- 支持离线资源
- 支持 ajax hook 避免 body 丢失
- Native 和 H5 侧都可以控制 ajax hook 开关
- Cookie 统一管理
- WKWebView 复用
- 兼容 WebViewJavascriptBridge

基于 MessageHandler 搭建通信层

WKWebView 支持新的 MessageHandler 通信方式，那我们就它来替换老的 iframe 通信方式了，原理是类似的，我们需要在 JS 侧创建一个统一的 callNative 函数，可以接受模块，方法，数据和回调，针对回调产生一个唯一的回调 id，并在 JS 侧把 id 和回调函数进行缓存，当在 Native 处理完逻辑并回调给 H5 时，可以通过保存的回调 id，找到缓存中的回调函数来完成个 JS 侧的数据获取。

```
var KKJSBridge = /** @class */ (function () {
        function KKJSBridge() {
            this.uniqueId = 1;
            this.callbackCache = {};
            this.eventCallbackCache = {};
        }
        /**
         * 调用 Natvie 方法git
         * @param module 模块
         * @param method 方法
         * @param data 数据
         * @param callback 调用回调
         */
        KKJSBridge.prototype.callNative = function (module, method, data, callback) {
            var message = {
                module: module || 'default',
                method: method,
                data: data,
                callbackId: null
            };
            if (callback) {
                // 拼装 callbackId
                var callbackId = 'cb_' + message.module + '_' + method + '_' + (this.uniqueId++) + '_' + new
Date().getTime();
                // 缓存 callback，用于在 Native 处理完消息后，通知 H5
                this.callbackCache[callbackId] = callback;
```

支持模块化的管理 JSAPI

有时我们需要把 JSAPI 进行分类，来更好的管理，比如我们会把 JSAPI 分成基础，数据，设备，界面，媒体，传感器，业务等模块，每个模块管理好自己分类里的 JSAPI，在多人开发的时候，也能更好的进行维护。

在 KKJSBridge 创建一个模块很简单，可以在代码中创建一个模块a，然后通过 KKJSBridgeEngine 进行注册。

```
@implementation ModuleA

+ (nonnull NSString *)moduleName {
    return @"a";
}

- (void)callToAddOneForA:(KKJSBridgeEngine *)engine params:(NSDictionary *)params responseCallback:(void (^)
(NSDictionary *responseData))responseCallback {
    NSInteger a = [params[@"a"] integerValue];
    a++;
    responseCallback ? responseCallback(@{@"a": @(a)}) : nil;
}

@end
```

```
// 给 webView 绑定一个 Bridge Engine
_jsBridgeEngine = [KKJSBridgeEngine bridgeForWebView:self.webView];
// 注册 模块A
[self.jsBridgeEngine.moduleRegister registerModuleClass:ModuleA.class];
```

在 JS 代码里可以用如下形式来调用。

```
window.KKJSBridge.call('a', 'callToAddOneForA', {a:4}, function(res) {
    console.log('receive callToAddOneForA res 4+1=: ', res.a);
});
```

支持模块共享上下文信息

有的时候，我们需要通过 JSAPI 去控制 Native 的外观或者行为，这时是需要借助一些上下文辅助信息来帮我我们完成业务上需求。比如：常见改变导航栏的背景色

```
@interface ModuleB()

@property (nonatomic, weak) ModuleContext *context;

@end

@implementation ModuleB

+ (nonnull NSString *)moduleName {
    return @"b";
}

- (instancetype)initWithEngine:(KKJSBridgeEngine *)engine context:(id)context {
    if (self = [super init]) {
        _context = context;
        NSLog(@"ModuleB 初始化并带上 %@", self.context.name);
    }

    return self;
}

- (void)callToGetVCTitle:(KKJSBridgeEngine *)engine params:(NSDictionary *)params responseCallback:(void (^)
(NSDictionary *responseData))responseCallback {
    responseCallback ? responseCallback(@{@"title": self.context.vc.navigationItem.title ?
self.context.vc.navigationItem.title : @""}) : nil;
}
```

```
// 给 webView 绑定一个 Bridge Engine
_jsBridgeEngine = [KKJSBridgeEngine bridgeForWebView:self.webView];

// 创建一个上下文对象
ModuleContext *context = [ModuleContext new];
context.vc = self;
context.scrollView = self.webView.scrollView;
context.name = @"上下文";

// 注册 模块B 并带入上下文
[self.jsBridgeEngine.moduleRegister registerModuleClass:ModuleB.class withContext:context];
```

在 JS 代码里可以用如下形式来调用。

```
window.KKJSBridge.call('b', 'callToGetVCTitle', {}, function(res) {
    console.log('receive vc title: ', res.title);
});
```

支持模块消息转发

有时候前期我们只定义了一个模块，所有 JSAPI 都在里面，后期想清楚了，又想要把 JSAPI 分模块来管理，那这里提供了一个转发机制，把一个模块的消息转发给另一个模块。

```
@implementation ModuleDefault

+ (nonnull NSString *)moduleName {
    return @"default";
}

+ (nonnull NSDictionary<NSString *, NSString *> *)methodInvokeMapper {
    // 消息转发，可以把 本模块的消息转发到 c 模块里
    return @{@"method": @"c.method"};
}

- (void)method:(KKJSBridgeEngine *)engine params:(NSDictionary *)params responseCallback:(void (^)(NSDictionary *responseData))responseCallback {
    responseCallback ? responseCallback(@{@"desc": @"我是默认模块"}) : nil;
}
@end

@implementation ModuleC

+ (nonnull NSString *)moduleName {
    return @"c";
}

- (void)method:(KKJSBridgeEngine *)engine params:(NSDictionary *)params responseCallback:(void (^)(NSDictionary *responseData))responseCallback {
    responseCallback ? responseCallback(@{@"desc": @"我是c模块"}) : nil;

// 给 webView 绑定一个 Bridge Engine
_jsBridgeEngine = [KKJSBridgeEngine bridgeForWebView:self.webView];
// 注册 默认模块
[self.jsBridgeEngine.moduleRegister registerModuleClass:ModuleDefault.class];
// 注册 模块C
[self.jsBridgeEngine.moduleRegister registerModuleClass:ModuleC.class];
```

在 JS 代码里可以用如下形式来调用。

```
window.KKJSBridge.call('default', 'method', {a:4}, function(res) {
    // 这里输出的是'receive method desc of default module：我是c模块'
    console.log('receive method desc of default module： ', res.desc);
});
```

支持离线资源&支持 ajax hook

我们都知道 WKWebView 支持离线资源，需要使用 NSURLProtocol 注册 http/https scheme 进行请求拦截，才能实现加载本地资源。而一旦进行注册， Network Process 进程把 ajax 请求 encode 后，通过 IPC 发送回 App Process，由于出于性能考虑，而不会把 HTTPBody 和 HTTPBodyStream 发送回来，所以就会导致 ajax 请求 body 体丢失。

为了既不影响 WKWebView 原有的离线包流程，又不影响 H5 侧发送 ajax 请求的体验，ajax hook 方式是目前合适的方式。好在有一个开源库 Ajax-hook 实现了在 H5 侧进行 ajax hook，那我们可以使用这个能力，在关键属性，方法，回调里通过 JSAPI 来调用 Native 的能力。

这里的样例代码展现的是怎么 hook readyState 属性和 open 方法，拦截 open 方法，可以使用 JSAPI 去调用 Native 侧发送请求能力，然后把 readyState 的所有状态按照 XMLHttpRequest 标准返回。具体的逻辑可以查看 GitHub。

```
window._hookAjaxProxy.hookAjax({
            // 拦截属性
            readyState: {
                getter: function (v, xhr) {
                    return xhr.callbackProperties.readyState;
                }
            }
            //拦截方法
            open: function (arg, xhr) {
                console.log("open called: method:%s,url:%s,async:%s", arg[0], arg[1], arg[2]);
                var method = arg[0];
                var url = arg[1];
                var async = arg[2];
                _XHR.cacheXHRIfNeed(this);
                window.KKJSBridge.call(_XHR.moduleName, 'open', {
                    "id": this.id,
                    "method": method,
                    "url": url,
                    "scheme": window.location.protocol,
                    "host": window.location.hostname,
                    "port": window.location.port,
                    "href": window.location.href,
                    "referer": document.referrer != "" ? document.referrer : null,
                    "useragent": navigator.userAgent,
                    "async": async
            });
```

Native 和 H5 侧都可以控制 ajax hook 开关

考虑在 H5 侧增加开关，目的是应对突发情况，当 H5 侧在发送 ajax 请求有问题时，我们可以通过 H5 侧的开关，来动态关闭 hook 功能，H5 侧控制了 ajax hook 开关时，都会通过 JSAPI 来通知 Native，可以让 H5 的开关同步到 Native 侧，这里我们可以通过 KVO/RAC 形式可以来监听 Native 侧的开关，来动态的注册和取消注册 http/https，来保证 H5 的变化能动态的同步到 Native 而不影响正常功能。

```
var KKJSBridgeConfig = /** @class */ (function () {
    function KKJSBridgeConfig() {
    }

    /**
     * 开启 ajax hook，方便 H5 自己控制是否开启 ajax hook
     */
    KKJSBridgeConfig.enableAjaxHook = function (enable) {
        KKJSBridgeConfig.enableAjaxHookWithNotify(enable, true);
    };
    /**
     * 开启 ajax hook 并 通知 native，方便 H5 自己控制是否开启 ajax hook
     */
    KKJSBridgeConfig.enableAjaxHookWithNotify = function (enable, notifyNative) {
        function _innerEnableAjaxHook(enable) {
            if (enable) {
                hookAjax();
            }
            else {
                unHookAjax();
            }
        }
        if (notifyNative) { // 是否需要通知到 native 侧，当需要通知 native 侧时，需要在通知后再来执行 H5 侧的开关修改
            window.KKJSBridge.call(KKJSBridgeConfig.moduleName, 'receiveConfig', { isEnableAjaxHook: enable },
function (data) {
                console.log("h5 control ajaxHook ", enable);
```

Cookie 统一管理

为了让 Cookie 处理更加统一和高内聚，这里定义了 KKWebView 并继承 WKWebView。同时也定义 KKWebViewCookieManager 来帮助处理 cookie 同步的问题。

还是先列下所有情况下 cookie 同步的问题，然后一一看下是怎么解决的：

首次同步请求的 cookie 同步（NSHTTPCookieStorage 同步到 WKWebView cookie）

在首次请求发送之前，先进行 cookie 同步处理。

```
@implementation KKWebViewCookieManager
+ (void)syncRequestCookie:(NSMutableURLRequest *)request {
    if (!request.URL) {
        return;
    }

    NSArray *availableCookie = [[NSHTTPCookieStorage sharedHTTPCookieStorage] cookiesForURL:request.URL];
    if (availableCookie.count > 0) {
        NSDictionary *reqHeader = [NSHTTPCookie requestHeaderFieldsWithCookies:availableCookie];
        NSString *cookieStr = [reqHeader objectForKey:@"Cookie"];
        [request setValue:cookieStr forHTTPHeaderField:@"Cookie"];
    }
}
@end
```

```
- (nullable WKNavigation *)loadRequest:(NSURLRequest *)request {
    NSMutableURLRequest *requestWithCookie = request.mutableCopy;
    [KKWebViewCookieManager syncRequestCookie:requestWithCookie];
    return [super loadRequest:requestWithCookie];
}
```

异步 ajax 的 cookie 同步（NSHTTPCookieStorage 同步到 WKWebView cookie）

通过 JS 注入的形式，来处理异步 ajax cookie 同步的问题。

```
@implementation KKWebViewCookieManager
+ (NSString *)ajaxCookieScripts {
    NSMutableString *cookieScript = [[NSMutableString alloc] init];
    for (NSHTTPCookie *cookie in [[NSHTTPCookieStorage sharedHTTPCookieStorage] cookies]) {
        // Skip cookies that will break our script
        if ([cookie.value rangeOfString:@"'"].location != NSNotFound) {
            continue;
        }
        // Create a line that appends this cookie to the web view's document's cookies
        [cookieScript appendFormat:@"document.cookie='%@=%@;", cookie.name, cookie.value];
        if (cookie.domain || cookie.domain.length > 0) {
            [cookieScript appendFormat:@"domain=%@;", cookie.domain];
        }
        if (cookie.path || cookie.path.length > 0) {
            [cookieScript appendFormat:@"path=%@;", cookie.path];
        }
        if (cookie.expiresDate) {
            [cookieScript appendFormat:@"expires=%@;", [[self cookieDateFormatter]
stringFromDate:cookie.expiresDate]];
        }
        if (cookie.secure) {
            // 只有 https 请求才能携带该 cookie
            [cookieScript appendString:@"Secure;"];
        }
        if (cookie.HTTPOnly) {
            // 保持 native 的 cookie 完整性，当 HTTPOnly 时，不能通过 document.cookie 来读取该 cookie。
```

```
WKUserScript *cookieScript = [[WKUserScript alloc] initWithSource:[KKWebViewCookieManager ajaxCookieScripts]
injectionTime:WKUserScriptInjectionTimeAtDocumentStart forMainFrameOnly:NO];
    [self.configuration.userContentController addUserScript:cookieScript];
```

服务器端（302）重定向/浏览器重定向的 cookie 同步（NSHTTPCookieStorage 同步到 WKWebView cookie）

在每次跳转之前，通过再次同步请求的 cookie，来解决重定向 cookie 不同步的问题。

```
// 1、在发送请求之前，决定是否跳转
- (void)webView:(WKWebView *)webView decidePolicyForNavigationAction:(WKNavigationAction *)navigationAction
decisionHandler:(void (^)(WKNavigationActionPolicy))decisionHandler {

    /**
     【COOKIE 3】对服务器端重定向(302)/浏览器重定向(a标签[包括 target="_blank"]) 进行同步 cookie 处理。
     由于所有的跳转都会是 NSMutableURLRequest 类型，同时也无法单独区分出 302 服务器端重定向跳转，所以这里统一对服务器端重定向
(302)/浏览器重定向(a标签[包括 target="_blank"])进行同步 cookie 处理。
     */
    if ([navigationAction.request isKindOfClass:NSMutableURLRequest.class]) {
        [KKWebViewCookieManager syncRequestCookie:(NSMutableURLRequest *)navigationAction.request];
    }

    ...
}
```

服务器端响应头里的 Set-Cookie 同步（WKWebView cookie 同步到 NSHTTPCookieStorage）

通过 WKWebView 的代理方法，在收到响应头后，可以从响应头 Set-Cookie 取到 cookie 信息，然后同步给 NSHTTPCookieStorage。

```
// 3、在收到响应后，决定是否跳转
- (void)webView:(WKWebView *)webView decidePolicyForNavigationResponse:(WKNavigationResponse *)navigationResponse
decisionHandler:(void (^)(WKNavigationResponsePolicy))decisionHandler {
    // iOS 12 之后，响应头里 Set-Cookie 不再返回。 所以这里针对系统版本做区分处理。
    if (@available(iOS 11.0, *)) {
        //  【COOKIE 4】同步 WKWebView cookie 到 NSHTTPCookieStorage。
        [KKWebViewCookieManager copyWKHTTPCookieStoreToNSHTTPCookieStorageForWebViewOniOS11:webView
withCompletion:nil];
    } else {
        //  【COOKIE 4】同步服务器端响应头里的 Set-Cookie，既把 WKWebView cookie 同步到 NSHTTPCookieStorage。
        NSHTTPURLResponse *response = (NSHTTPURLResponse *)navigationResponse.response;
        NSArray *cookies = [NSHTTPCookie cookiesWithResponseHeaderFields:[response allHeaderFields]
forURL:response.URL];
        for (NSHTTPCookie *cookie in cookies) {
            [[NSHTTPCookieStorage sharedHTTPCookieStorage] setCookie:cookie];
        }
    }

    ...
}
```

H5 侧的 document.cookie 同步（WKWebView cookie 同步到 NSHTTPCookieStorage）

可以使用 cookie hook 来把 document.cookie 的修改，通过 JSAPI 同步给 NSHTTPCookieStorage。

```
var cookieDesc = Object.getOwnPropertyDescriptor(Document.prototype, 'cookie') ||
Object.getOwnPropertyDescriptor(HTMLDocument.prototype, 'cookie');
if (cookieDesc && cookieDesc.configurable) {
    Object.defineProperty(document, 'cookie', {
        configurable: true,
        enumerable: true,
        get: function () {
            console.log('getCookie');
            return cookieDesc.get.call(document);
        },
        set: function (val) {
            console.log('setCookie');
            cookieDesc.set.call(document, val);
            window.KKJSBridge.call(_COOKIE.moduleName, 'setCookie', {
                "cookie": val
            });
        }
    });
}
```

异步 ajax 响应头里的 Set-Cookie 同步（WKWebView cookie 同步到 NSHTTPCookieStorage）

处理 ajax response Set-Cookie 同步问题，此时 Set-Cookie 并不会触发 document.cookie 设置 cookie。一般只有登录相关的 ajax 请求才会在 response 里返回 Set-Cookie。好在 Hybird WebView 都是以 Native 的登录 cookie 为准，这种情况影响不大，主要是需要跟前端约定好。

如果再和 ajax hook 结合后，所有的 ajax 都是走的 native 发送请求，request 自动从 NSHTTPCookieStorage 获取 cookie，并且 response Set-Cookie 也都会存在 NSHTTPCookieStorage 里，所以这个问题是可以通过 ajax hook 来完美解决的。

cookie HTTPOnly 同步 （NSHTTPCookieStorage 同步到 WKWebView cookie 和 WKWebView cookie 同步到 NSHTTPCookieStorage）

我们都知道一旦设置了 HTTPOnly，则意味着 通过 document.cookie 是获取不到该 cookie，而实际发送请求时，还是会发送出去的。

针对这类的 cookie，需要分为两类来看：

如果 HTTPOnly 类 cookie 也是在 Native 上的登录接口返回的，而通过 ajax cookie js 注入去同步 cookie 时，HTTPOnly cookie 也是可以发送让 ajax 携带并发送的，这种是正常的操作，也没有问题。

而如果 HTTPOnly 类 cookie 是在 H5 侧通过 ajax reposne Set-Cookie HttpOnly 设置的，这种情况处理不了，因为从 document.cookie 本身是读取不到 HTTPOnly 类 cookie 的。所以还是建议针对这类 cookie 最好是通过 native 来管理。

针对第二类的 HTTPOnly 类 cookie 问题，如果结合 ajax hook 来看，Natvie 发送的请求是不会依赖 document.cookie，也是会通过 NSHTTPCookieStorage 获取 cookie，来发送请求的，所以针对这点，也是可以通过 ajax hook 来完美解决。

WKWebView 复用

为了加载 H5 的首屏速度，可以在启动时，加载一个 WKWebView 到复用池子里，等第一次加载 H5 时，可以从复用池取出缓存的 WKWebView，就可以直接使用，省去初始化的过程。

```
@implementation WebViewController

+ (void)load {
    __block id observer = [[NSNotificationCenter defaultCenter]
addObserverForName:UIApplicationDidFinishLaunchingNotification object:nil queue:nil usingBlock:^(NSNotification *
_Nonnull note) {
        [self prepareWebView];
        [[NSNotificationCenter defaultCenter] removeObserver:observer];
    }];
}

+ (void)prepareWebView {
    // 预先缓存一个 webView
    [KKWebView configCustomUAWithType:KKWebViewConfigUATypeAppend UAString:@"KKJSBridge/1.0.0"];
    [[KKWebViewPool sharedInstance] enqueueWebViewWithClass:KKWebView.class];
}

- (void)dealloc {
    [[KKWebViewPool sharedInstance] enqueueWebView:self.webView];
    NSLog(@"WebViewController dealloc");
}

- (instancetype)initWithUrl:(NSString *)url {
    if (self = [super initWithNibName:nil bundle:nil]) {
        _url = [url copy];
        [self commonInit];
```

兼容 WebViewJavascriptBridge

兼容 WebViewJavascriptBridge 本身不是目的，这里是提供了一个小思路，可以方便让自己的通信层更新换代。

```
/**
 * 当 H5 还在使用 WebViewJavascriptBridge（基于 iframe 通信） 框架时，可以通过下方代码来兼容 WebViewJavascriptBridge。这样可以在不用改动 H5 任何代码，就可以无缝支持新的 JSBridge。
 * 如果使用不是 WebViewJavascriptBridge 这样的框架，兼容的原理也是类似的。
 */
; (function(window) {
    // 声明 WebViewJavascriptBridge 在函数体作用域里，这样就不会污染全局作用域
    var WebViewJavascriptBridge = {
      init: function (func) {
      },
      registerHandler: function (handlerName, handler) {
        window.KKJSBridgeInstance.on(handlerName, handler);
      },
      callHandler: function (handlerName, data, responseCallback) {
        window.KKJSBridgeInstance.call(null, handlerName, data, responseCallback);
      }
    };
    window.WebViewJavascriptBridge = WebViewJavascriptBridge;

    // 告诉 H5， WebViewJavascriptBridge 已经 ready
    let WebViewJavascriptBridgeReadyEvent: Event = document.createEvent("Events");
    WebViewJavascriptBridge.initEvent("WebViewJavascriptBridgeReady");
    document.dispatchEvent(WebViewJavascriptBridge);
})(window);
```

```
NSString *jsString = [[NSString alloc] initWithContentsOfFile:[[NSBundle bundleForClass:self.class]
pathForResource:@"WebViewJavascriptBridge" ofType:@"js"] encoding:NSUTF8StringEncoding error:NULL];
    WKUserScript *userScript = [[WKUserScript alloc] initWithSource:jsString
injectionTime:WKUserScriptInjectionTimeAtDocumentStart forMainFrameOnly:NO];
    [self.webView.configuration.userContentController addUserScript:userScript];
```

GitHub地址

https://github.com/karosLi/KKJSBridge

TODO

- [ ] Fetch hook。 虽然现在大多数 H5 页面的异步请求都是基于 ajax 实现的，随着 Fetch 的慢慢普及，后面也会多起来。

参考

- Ajax-hook
- HybridPageKit
- kerkee_ios

卡洛斯

62

关注作者

卡洛斯的博客

用户专栏

正在学习的路上

2 人关注　**10** 篇文章

关注专栏　专栏主页

3 条评论

得票 • 时间

撰写评论 ...

提交评论

muyear： 在请求之前把 url 上的 http/https scheme 替换成自定义的 customscheme，然后返回的 Html 中的子资源链接，使用 src='//xxx.com/a.js' 的形式去加载，这样也会让子资源带上 customscheme，通过这种方式 NSURLProtocol 只用注册和拦截 customscheme，针对 customscheme 丢失 body 信息没有任何影响，然后 H5 ajax 请求，还是可以走 http/https 来请求。这种也可以解决 body 丢失的问题，但是必须要求 H5 侧能够按照规范来，如果 H5 没有历史包袱，这种也是可行的，如果有历史包袱的话，也会有改动的成本

• 回复 • 3月19日

muyear： 这种不行吧？在请求之前把 url 上的 http/https scheme 替换成自定义的 customscheme，请求还能发出去?

• 回复 • 3月19日

卡洛斯： 这种如果有H5配合，也是可以的，主要是看H5能配合到什么程度，H5团队和你们团队是不是在一个大团队下，你们的团队目标是否一致

• 回复 • 5月14日

推荐阅读

AJAX请求真的不安全么？谈谈Web安全与AJAX的关系。

开篇三问 AJAX请求真的不安全么？ AJAX请求哪里不安全？ 怎么样让AJAX请求更安全？ 前言 本文包含的内容较多，包括AJAX，C...

撒网要见鱼 • 阅读 10.8k • 73 赞 • 15 评论

再也不学AJAX了！（三）跨域获取资源① - 同源策略

我们之前提到过，AJAX技术使开发者能够专注于互联网中数据的传输，而不再拘泥于数据传输的载体。通过AJAX技术，我们获取数...

libinfs • 阅读 2.7k • 19 赞 • 5 评论

Cookies with CORS

在本地开发环境中，网站是在本地的Express服务器上跑的，地址是localhost:8000（127.0.0.1：8000），而网站里的所有AJAX请求的...

zach5078 • 阅读 2.9k • 4 赞 • 1 评论

前端请求的第N种方式——玩转React Hook

我曾在几年前写过一篇文章——《Jquery ajax, Axios, Fetch区别之我见》——从原理和使用层面分析了ajax,axios和fetch的区别。现...

这是你的玩具车吗 • 阅读 678 • 4 赞

ZooTeam 前端周刊 | 第 84 期

伪类是目前唯一一个可以大规模放心使用的逻辑伪类，非常有用，优点也很突出，但是，其中也不乏一些会让人踩坑的地方，本文主...

政采云前端团队 • 阅读 511 • 3 赞

AJAX 请求真的不安全么？

作者：撒网要见鱼[链接]开篇三问 AJAX请求真的不安全么？ AJAX请求哪里不安全？ 怎么样让AJAX请求更安全？ 前言 本文包含的...

Java技术栈 • 阅读 511 • 2 赞

SameSite小识

A cookie associated with a cross-site resource at [链接] was set without the SameSite attribute. A future release of Chrome will onl...

深红 • 阅读 1.2k • 1 赞

node+ajax实战案例（6）

首先，我们要搞明白cookie是什么？Cookie 是在 HTTP 协议下，服务器或脚本可以维护客户工作站上信息的一种方式。Cookie 是由 ...

镙钉课堂 • 阅读 9 • 1 赞