伤心的Easyman Lv2
2020年08月17日 阅读 1147

关注

WKWebview秒开的实践及踩坑之路

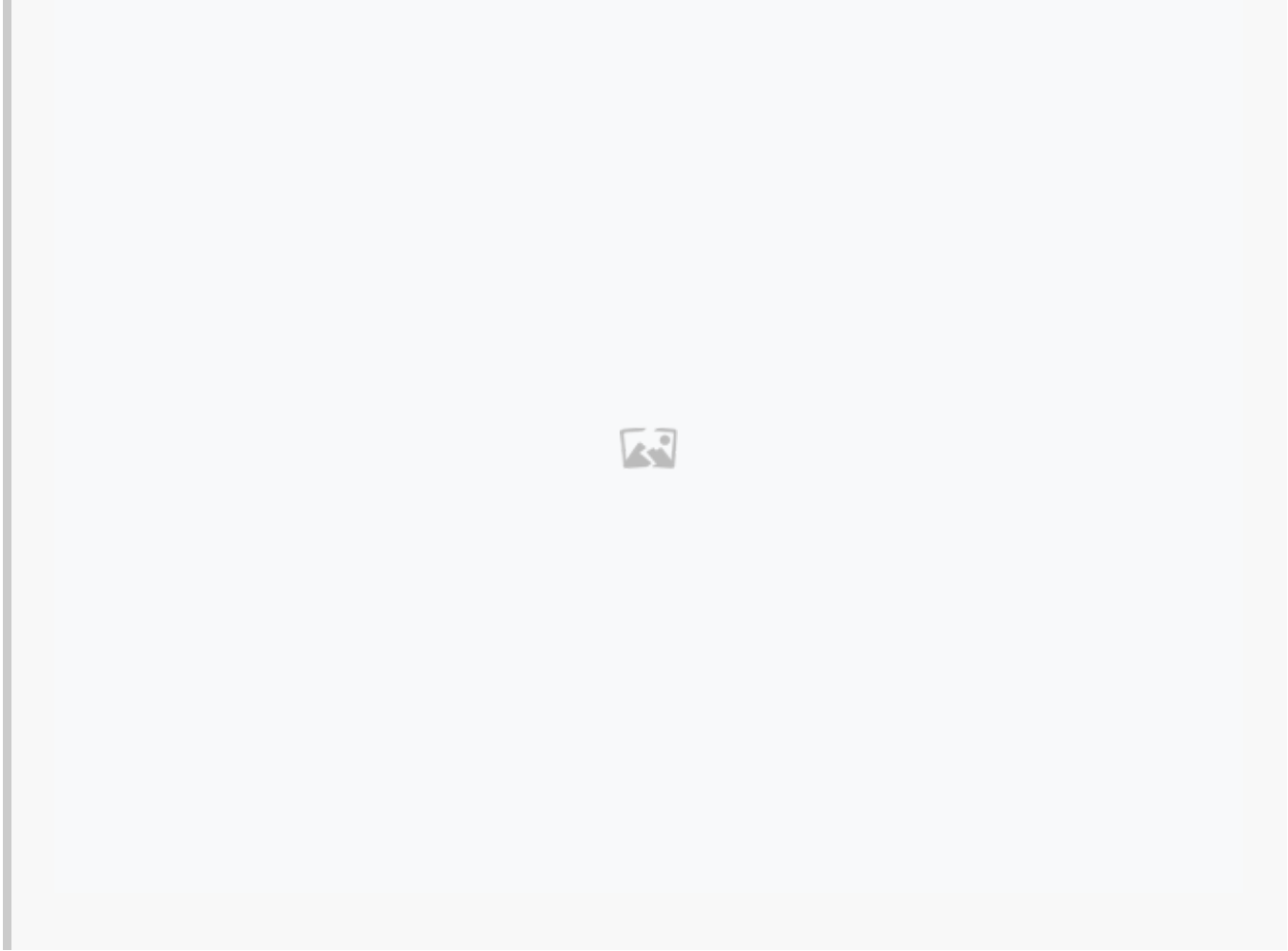
优化背景

- 众所周知，H5的部分优势（开发快，迭代快，热更新）是很明显的，公司客户端的部分业务都是由H5来实现的，网络好的情况下体验也是很不错的
- 但是其实H5的体验是比原生差的，这就需要想办法如何提高H5加载速度，优化体验，首屏的加载速度还是很影响体验的

加载速度

关于加载速度慢有很多文章都已经详细解释了，h5在加载工作中做了很多事

> 初始化 webview -> 请求页面 -> 下载数据 -> 解析HTML -> 请求 js/css 资源 -> dom 渲染 ->



一般页面在 dom 渲染后才能展示，可以发现，H5 首屏渲染白屏问题的原因关键在于，如何优化减少从请求下载页面到渲染之间这段时间的耗时。

前后端优化

这其中可做的优化特别多，前后端能够做的是

- 降低请求量：减少 HTTP 请求数, 合并资源，minify / gzip 压缩，webP，lazyLoad。

因为手机浏览器同时响应请求是4个，4个的请求数也许不是特别靠谱，没有查到出处，但是肯定是越少越好

- HTTP 协议缓存请求，离线缓存 manifest，离线数据缓存localStorage。
- 加快请求速度：预解析DNS，减少域名数，并行加载，CDN 分发。
- 渲染：JS/CSS优化，加载顺序，服务端渲染模板直出。

和展示数据的请求。

所以客户端内，优化最关键的其实就是如何缓存这些网络资源，也就是离线包缓存方案。

离线包方案的实践

方案选型是两种

- 基于 LocalWebServer 实现 WKWebView 离线资源加载
- 使用WKURLSchemeHandler实现 WKWebView 离线资源加载

LocalWebServer

基于iOS的local web server，目前大致有以下几种较为完善的框架：

- CocoaHttpServer (支持iOS、macOS及多种网络场景)
- GCDWebServer （基于iOS，不支持 https 及 webSocket）
- Telegraph （Swift实现，功能较上面两类更完善）

当时采用的是GCDWebServer，在打开APP后直接启动Webserver，H5的链接直接替换成本地localhost+端口号链接的地址。

本来的方案是本地服务器和远端h5服务器同步下载资源，下载后客户端请求本地服务器的路径，如未找到相应的资源再请求远端服务器的资源文件。

测试过程中碰到很多奇怪的问题（暂不一一举例），也有提到以下问题并且时间紧急所以并未做进一步的深入：

> - 资源访问权限安全问题
> - APP前后台切换时，服务重启性能耗时问题
> - 服务运行时，电量及CPU占有率问题
> - 多线程及磁盘IO问题

WKURLSchemeHandler

关于离线包

关于离线包的分发，就是普通的zip离线包和一个版本控制的json文件，每次打离线包会修改json文件里的版本号，并附有离线包下载地址。此处可以优化的更好，但暂时并不需要太复杂。

离线包的下载和解压

只是简单的下载并解压到本地资源路径，关于版本比对的代码这里没有展示出来，自行注意，避免每次都全量更新。

```OC
/* 创建网络下载对象 */
        AFURLSessionManager *manager = [[AFURLSessionManager alloc] initWithSessionConfigur
        /* 下载地址 */
        NSURL *url = [NSURL URLWithString:request.urlParameters.path];
        NSURLRequest *request = [NSURLRequest requestWithURL:url];
        /* 下载路径 */
        //获取Document文件
        NSString * docsdir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUs
        NSString * zipFilePath = [docsdir stringByAppendingPathComponent:@"zip"];//将需要创建
        NSString * H5FilePath = [docsdir stringByAppendingPathComponent:@"H5"];
        NSFileManager *fileManager = [NSFileManager defaultManager];
        BOOL zipIsDir = NO;
        BOOL H5IsDir = NO;
        // fileExistsAtPath 判断一个文件或目录是否有效，isDirectory判断是否一个目录
        BOOL zipexisted = [fileManager fileExistsAtPath:zipFilePath isDirectory:&zipIsDir];
        BOOL H5Existed = [fileManager fileExistsAtPath:H5FilePath isDirectory:&H5IsDir];
        if ( !(zipIsDir == YES && zipexisted == YES) ) {//如果文件夹不存在
            [fileManager createDirectoryAtPath:zipFilePath withIntermediateDirectories:YES
        }
        if (!(H5IsDir == YES && H5Existed == YES) ) {
            [fileManager createDirectoryAtPath:H5FilePath withIntermediateDirectories:YES a
        }
        //删除
//        [[NSFileManager defaultManager] removeItemAtPath:zipFilePath error:nil];
        NSString *filePath = [zipFilePath stringByAppendingPathComponent:url.lastPathCompon
        /* 开始请求下载 */
        NSURLSessionDownloadTask *downloadTask = [manager downloadTaskWithRequest:request p
            NSLog(@"下载进度：%.0f%", downloadProgress.fractionCompleted * 100);
        } destination:^NSURL * _Nonnull(NSURL * _Nonnull targetPath, NSURLResponse * _Nonnu
            /* 设定下载到的位置 */
            return [NSURL fileURLWithPath:filePath];
        } completionHandler:^(NSURLResponse * _Nonnull response, NSURL * _Nullable filePath
            NSTimeInterval delta = CACurrentMediaTime() - self->start;
             NSLog(@"下载完成，耗时：%f",delta);
            // filePath就是你下载文件的位置，你可以解压，也可以直接拿来使用
            NSString *imgFilePath = [filePath path];// 将NSURL转成NSString
```

```
//            [[NSFileManager defaultManager] removeItemAtPath:H5FilePath error:nil];
            [fileManager createDirectoryAtPath:H5FilePath withIntermediateDirectories:YES a
            //解压
            [SSZipArchive unzipFileAtPath:zipPath toDestination:H5FilePath];
            //清理缓存
            [DLCommenHelper clearWebCache];
        }];
        [downloadTask resume];
```

WKWebview缓存池

美团有篇文章提到，在使用iOS 10的模拟器测试 WKWebView的加载速度，首次初始化的时间耗时有700多毫秒。其实本人用iOS 13的真机，发现初始化的时间约在200毫秒左右甚至更短。虽然只占整个加载时间的特别小的一部分，但是本着能优则优的原则还是做了处理，也就是预加载Webview

- 新建了一个单例类`SDIWKWebViewPool`，默认缓存池里的数量是10个

OC

```
+ (instancetype)sharedInstance {
    static dispatch_once_t onceToken;
    static SDIWKWebViewPool *instance = nil;
    dispatch_once(&onceToken,^{
        instance = [[super allocWithZone:NULL] init];
    });
    return instance;
}

+ (id)allocWithZone:(struct _NSZone *)zone{
    return [self sharedInstance];
}

- (instancetype)init
{
    self = [super init];
    if (self) {
        self.initialViewsMaxCount = 10;
        self.preloadedViews = [NSMutableArray arrayWithCapacity:self.initialViewsMaxCount];
    }
    return self;
}
```

- 在合适的地方提前调用`//预加载wkwebview [[SDIWKWebViewPool sharedInstance]`

```OC
/**
 预初始化若干WKWebView

 @param count 个数
 */
- (void)prepareWithCount:(NSUInteger)count {
    NSTimeInterval start = CACurrentMediaTime();
    // Actually does nothing, only initialization must be called.
    while (self.preloadedViews.count < MIN(count,self.initialViewsMaxCount)) {
        id preloadedView = [self createPreloadedView];
        if (preloadedView) {
            [self.preloadedViews addObject:preloadedView];
        } else {
            break;
        }
    }
    NSTimeInterval delta = CACurrentMediaTime() - start;
    NSLog(@"=======初始化耗时：%f",  delta);
}

/**
 从池中获取一个WKWebView
 @return WKWebView
 */
- (WKWebView *)getWKWebViewFromPool {
    if (!self.preloadedViews.count) {
        NSLog(@"不够啦！");
        return [self createPreloadedView];
    } else {
        id preloadedView = self.preloadedViews.firstObject;
        [self.preloadedViews removeObject:preloadedView];
        return preloadedView;
    }
}
```

- 创建webview的方法如下,需要注意的是`kWKWebViewReuseScheme`，WKWebView需要注册这个scheme才能实现拦截，这个是WKWebview拦截需要的准备工作。

- `SDICustomURLSchemeHandler`是我的自定义拦截类

> 关于这里的版本为什么设置成iOS 12以上，`WKURLSchemeHandler`是苹果iOS 11就已推出，但是有发现某款机型在iOS 11.2上拦截失效，导致产生Webview白屏。所以这里一刀切，直接12以上才处理，其实iOS 12一下的用户量特别少，所以不需要太担心。

```OC
//scheme定义
#define kWKWebViewReuseScheme    @"kwebview"

/**
 创建一个WKWebView
 @return WKWebView
 */
- (WKWebView *)createPreloadedView {
      WKUserContentController *userContentController = WKUserContentController.new;
     WKWebViewConfiguration *configuration = [[WKWebViewConfiguration alloc] init];
      NSString *cookieSource = [NSString stringWithFormat:@"document.cookie = 'API_SESSION=
      WKUserScript *cookieScript = [[WKUserScript alloc] initWithSource:cookieSource inject
      [userContentController addUserScript:cookieScript];
     // 赋值userContentController
     configuration.userContentController = userContentController;
     configuration.preferences.javaScriptEnabled = YES;
     configuration.suppressesIncrementalRendering = YES; // 是否支持记忆读取
     [configuration.preferences setValue:@YES forKey:@"allowFileAccessFromFileURLs"];//支持跨
//    WKWebViewConfiguration *wkWebConfig = [[WKWebViewConfiguration alloc] init];
//    WKUserContentController *wkUController = [[WKUserContentController alloc] init];
//    wkWebConfig.userContentController = wkUController;

    if (@available(iOS 12.0, *)) {
        [configuration setURLSchemeHandler:[[SDICustomURLSchemeHandler alloc] init] forURLS
    } else {
        // Fallback on earlier versions
    }
    WKWebView *wkWebView = [[WKWebView alloc]initWithFrame:CGRectZero configuration:configu
    //根据自己的业务
    wkWebView.allowsBackForwardNavigationGestures = YES;
    return wkWebView;
}
```

替换url scheme

```
if (@available(iOS 12.0, *)) {
                if([urlString hasPrefix:@"http"] && [urlString containsString:@"ui-h5"]){
                    urlString = [urlString stringByReplacingOccurrencesOfString:@"https" wi
                }
            }
```

这里是通过规则直接把 `https` 替换为 `kWKWebViewReuseScheme` ,也就是替换url scheme http(s)为自定

> 需要注意的有两点: 前端这边加载js等资源都是用相对路径，前端的ajax请求，像post请求，scheme使用http(s)不使用自定义协议，这样native不会拦截，完全交给H5与服务器交互，就不会发生发送post请求，body丢失的情况。
>
> 在我的项目里，H5对服务器的请求都是通过native端来转发的，所以也不存在拦截post请求，body丢失的情况。所以上面这样的改动对H5端是无侵入式的，不需要修改业务代码。

最最重要的自定义SDICustomURLSchemeHandler类

OC

```
- (void)webView:(WKWebView *)webView startURLSchemeTask:(id <WKURLSchemeTask>)urlSchemeTask
API_AVAILABLE(ios(12.0)){
    dispatch_sync(self.serialQueue, ^{
        [_taskVaildDic setValue:@(YES) forKey:urlSchemeTask.description];
    });

    NSDictionary *headers = urlSchemeTask.request.allHTTPHeaderFields;
    NSString *accept = headers[@"Accept"];

    //当前的requestUrl的scheme都是customScheme
    NSString *requestUrl = urlSchemeTask.request.URL.absoluteString;
    NSString *fileName = [[requestUrl componentsSeparatedByString:@"?"].firstObject compone
    NSString *replacedStr = [requestUrl stringByReplacingOccurrencesOfString:kWKWebViewReus
    self.replacedStr = replacedStr;
    //Intercept and load local resources.
    if ((accept.length >= @"text".length && [accept rangeOfString:@"text/html"].location !=
        [self loadLocalFile:fileName urlSchemeTask:urlSchemeTask];
    } else if ([self isMatchingRegularExpressionPattern:@"\\.(js|css)" text:requestUrl]) {/
        [self loadLocalFile:fileName urlSchemeTask:urlSchemeTask];
    } else if (accept.length >= @"image".length && [accept rangeOfString:@"image"].location
      NSString *key = [[SDWebImageManager sharedManager] cacheKeyForURL:[NSURL URLWithStrin
        [[SDWebImageManager sharedManager].imageCache queryImageForKey:key options:SDWebIma
            if (image) {
                NSData *imgData = UIImageJPEGRepresentation(image, 1);
                NSString *mimeType = [self getMIMETypeWithCAPIAtFilePath:fileName] ?: @"ima
                [self resendRequestWithUrlSchemeTask:urlSchemeTask mimeType:mimeType reques
            } else {
                [self loadLocalFile:fileName urlSchemeTask:urlSchemeTask];
            }
        }];
    } else {
        //return an empty json.
        NSData *data = [NSJSONSerialization dataWithJSONObject:@{ } options:NSJSONWritingPr
        [self resendRequestWithUrlSchemeTask:urlSchemeTask mimeType:@"text/html" reque
    }
```

```
    NSError *error = NULL;
    NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:pattern
    NSTextCheckingResult *result = [regex firstMatchInString:text options:0 range:NSMakeRan
    return MHObjectIsNil(result)?NO:YES;
}
```

- 上面的代码是拦截资源请求后的处理代码。收到拦截请求后，先获取本地资源包对应的资源，转换成data回传给webView进行渲染处理；若本地没有，则customScheme替换成https的url重发请求通知webview，这就是基本流程。

- 以下就是加载本地资源和重发请求的代码

```OC
//Load local resources, eg: html、js、css...
- (void)loadLocalFile:(NSString *)fileName urlSchemeTask:(id <WKURLSchemeTask>)urlSchemeTas
    if(![self->_taskVaildDic boolValueForKey:urlSchemeTask.description default:NO] || !urlS
                return;
    }

    NSString * docsdir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDo
    NSString * H5FilePath = [[docsdir stringByAppendingPathComponent:@"H5"] stringByAppendi
    //If the resource do not exist, re-send request by replacing to http(s).
    NSString *filePath = [H5FilePath stringByAppendingPathComponent:fileName];

    if (![[NSFileManager defaultManager] fileExistsAtPath:filePath]) {
        NSLog(@"开始重新发送网络请求");
        if ([self.replacedStr hasPrefix:kWKWebViewReuseScheme]) {

            self.replacedStr =[self.replacedStr stringByReplacingOccurrencesOfString:kWKWeb

            NSLog(@"请求地址:%@",self.replacedStr);

        }

        self.replacedStr = [NSString stringWithFormat:@"%@?%@",self.replacedStr,[SAMKeychai
        start = CACurrentMediaTime();//开始加载时间
        NSLog(@"web请求开始地址:%@",self.replacedStr);

        @weakify(self)
        NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL URLWithSt
        NSURLSession *session = [NSURLSession sessionWithConfiguration:[NSURLSessionConfigu
        NSURLSessionDataTask *dataTask = [session dataTaskWithRequest:request completionHan
            @strongify(self)
            if([self->_taskVaildDic boolValueForKey:urlSchemeTask.description default:
                return;
```

```
            [urlSchemeTask didReceiveData:data];
            if (error) {
                [urlSchemeTask didFailWithError:error];
            } else {
                NSTimeInterval delta = CACurrentMediaTime() - self->start;
                NSLog(@"=======web请求结束地址%@: : : %f", self.replacedStr, delta);
                [urlSchemeTask didFinish];
            }
        }];
        [dataTask resume];
        [session finishTasksAndInvalidate];
    } else {
        NSLog(@"filePath:%@",filePath);
        if(![self->_taskVaildDic boolValueForKey:urlSchemeTask.description default:NO] || !
            NSLog(@"return");
            return;
        }

        NSData *data = [NSData dataWithContentsOfFile:filePath options:NSDataReadingMappedI
        [self resendRequestWithUrlSchemeTask:urlSchemeTask mimeType:[self getMIMETypeWithCA
    }
}

- (void)resendRequestWithUrlSchemeTask:(id <WKURLSchemeTask>)urlSchemeTask
                              mimeType:(NSString *)mimeType
                           requestData:(NSData *)requestData  API_AVAILABLE(ios(11.0)) {
    if(![self->_taskVaildDic boolValueForKey:urlSchemeTask.description default:NO] || !urlS
        return;
    }

    NSString *mimeType_local = mimeType ? mimeType : @"text/html";
    NSData *data = requestData ? requestData : [NSData data];
    NSURLResponse *response = [[NSURLResponse alloc] initWithURL:urlSchemeTask.request.URL
                                                        MIMEType:mimeType_local
                                           expectedContentLength:data.length
                                                textEncodingName:nil];
    [urlSchemeTask didReceiveResponse:response];
    [urlSchemeTask didReceiveData:data];
    [urlSchemeTask didFinish];
}
```

整个过程中遇到的一些踩坑点

> `_taskVaildDic` 是一个NSMutableDictionary，它里面存的是以当前的urlSchemeTask做key，拦截开始时设置YES，收到停止通知时设置NO。这是由于在快速切换webview时，之前的urlSchemeTask已经停止但是后面再次调用了它的方法就会产生该崩溃。
>
> 在实际使用过程中，用bugly监控到还是会有该崩溃发生，只不过次数特别少，一天约四五条左右。还在寻找问题的原因中。

```
- (void)webView:(WKWebView *)webView startURLSchemeTask:(id <WKURLSchemeTask>)urlSchemeTask
API_AVAILABLE(ios(12.0)){
    dispatch_sync(self.serialQueue, ^{
        [_taskVaildDic setValue:@(YES) forKey:urlSchemeTask.description];
    });
}

- (void)webView:(nonnull WKWebView *)webView stopURLSchemeTask:(nonnull id<WKURLSchemeTask>)
    NSError *error = [NSError errorWithDomain:urlSchemeTask.request.URL.absoluteString code
    NSLog(@"weberror:%@",error);
    dispatch_sync(self.serialQueue, ^{
         [self->_taskVaildDic setValue:@(NO) forKey:urlSchemeTask.description];
    });
}
```

2. WKWebview的默认缓存策略问题

之前未考虑到WKWebview的默认缓存策略(WKWebView默认缓存策略完全遵循HTTP缓存协议)，关于HTTP缓存协议可看此文了解：WKWebView默认缓存策略与HTTP缓存协议

在h5打包上线并更新离线包后，H5的资源文件修改是变更md5文件名的。由于缓存策略默认时间是一个小时，会导致缓存的url加载不到修改后的js，css等文件（无论是本地离线包和远端服务器都已经没有这个md5文件）。

简单的解决方案是通过资源链接加版本号后缀，每次更新资源的时候变更版本号，在上面的代码中有做这部分处理。既保证了实时的更新，又保证了加载速度。

3. uni-app 图片CDN问题

做完上述的离线包优化后，发现新下载APP的情况，会偶发加载很慢问题。iOS出现，但是android并未出现。

uni-app 问题链接

H5部分是用`uni-app`开发的，所以发现这个问题后由前端同事修复后恢复正常。

4. chunk-vendors.js文件过大

这个问题也是抓包发现的，在未打开离线包缓存开关时，发现h5的加载速度过慢，发现加载的`chunk-vendors.js`文件过大约1.7M。`stopURLSchemeTask`方法里会报error错误信息`Error Domain=`的错误信息。也由前端同事处理了这个问题。

最终效果

统计了APP在不开离线包方案时，webview平均加载时长在1.5-2秒的范围内（这里是计算的webview开始加载到导航完成的时间），在上述优化完成后，打开的时长在0.25-0.3秒之间。

所以效果还是很显著的，用户的直观感受就是接近于秒开的体验。

总结

上面的优化过程中踩了很多坑，但是也重新梳理了Webview的加载过程，默认缓存策略机制等内容。上面的方案肯定不是最优的，只是一个快速达到WKWebview接近秒开效果的一个方案。

有什么更好的解决方案或者上述文中有不对的地方，希望大家指出，欢迎共同讨论~

> **关注下面的标签，发现更多相似文章**

Webkit

输入评论...

enni_sk

"在我的项目里，H5对服务器的请求都是通过native端来转发的"
是通过JSBridge调用原生接口吗?

2天前 回复

伤心的Easyman Lv2 (作者) iOS工程师 @ 粤海...

是的，统一在客户端对接口数据做加解密密等相关操作

2天前

VitoLI16181 Lv1 大前端、兼职iOS @ 深...

所以你们用的最多的 最成熟的是哪一种方案 👍

2天前 回复

伤心的Easyman Lv2 (作者) iOS工程师 @ 粤海...

我们只是在这个基础上优化扩展，成本比较小

2天前

静坐常思过君 Lv1 全栈工程师 @ 开发者

是基于uni做的吗

2天前 回复

伤心的Easyman Lv2 (作者) iOS工程师 @ 粤海...

几乎一半的业务是用的uni app，考虑到审核的原因，所以没有全用

2天前

静坐常思过君 Lv1 全栈工程师 @ 开...

审核是指哪个平台？我看了uni的，他大补丁就是下载替换文件，你会ios完全可以把app做成动态热更新，不用整包下载，或是可以做成app中嵌入h5页面

1天前

伤心的Easyman Lv2 (作者) iOS工程师 @ 粤海...

是的，我们现在一半的业务就是uni-app，支持热更，可以快速迭代。但是整个APP用的话确实是用审核风险的，毕竟苹果今年三月以来对纯web app管的更严。话说回来，uni-app同样用来开发小程序真的很不错

1天前

静坐常思过君 Lv1 全栈工程师 @ 开...

伤心的Easyman Lv2 (作者) iOS工程师 @ 粤海...

回复 静坐常思过君: 可以的，我们一直就是这样的，H5部分的业务是可以直接热更新的，主要做好web版本管理，还要注意APP内的缓存问题

1天前

静坐常思过君 Lv1 全栈工程师 @ 开...

留个QQ，明天聊聊啊 😃

1天前

伤心的Easyman Lv2 (作者) iOS工程师 @ 粤海...

回复 静坐常思过君: 白天太忙

16小时前

相关推荐

且行且珍惜_iOS · 2月前 · Webkit

iOS WKWebView+UITableView混排

39 4

分享 · pingan8787 · 2年前 · 面试 / 前端 / Webkit / Vue.js

面试的信心来源于过硬的基础

2025 40

RobinsonZhang · 2年前 · Webkit / JavaScript / iOS / 前端

移动端常见bug汇总001

1412 26

练识 · 8月前 · Webkit

2020年史上最全移动端Web整理从开发基础到实战(一)

53 6

chaocai · 3月前 · Webkit

Hybrid 开发之 WebView 交互

8

卞卞村长L · 2年前 · iOS / Android / Webkit / 前端

手机/移动前端开发需要注意的20个要点

首页 探索掘金

RobinsonZhang · 2年前 · Webkit / JavaScript / iOS / 前端 / Apple / WWDC

移动端常见bug汇总002

384 18

郭某某 · 2年前 · 前端 / Webkit

Web全屏模式

290 8

一只有交流障碍的丑程 · 5月前 · Webkit

Android 中的Cookie了解一下

9 1