

chaocai Lv2

2020年05月21日 阅读 603

[关注](#)

Hybrid 开发之 WebView 交互

WebView 是我们在 app 当中所提供 web 页面的浏览器环境。在 iOS 当中，具体是 WKWebView，或者 UIWebView。由于 UIWebView 即将退出历史舞台，我们这里只讨论 WKWebView，也就是 WebKit 这个系统框架。

在 Hybrid 开发当中，最为关键的是原生与 js 之间的交互。而通过交互过程，最主要是扩展 web 页面所不具别的原生能力。

原生与JS之间的交互

原生调用 JS

我们知道，在 iOS 中，甚至在 safari 中，js 都是通过 JavascriptCore 执行的。我们只要获取到 web 页面的执行上下文 JSContext，就可以操控 JS。

在 UIWebView 时代，我们可以这样获取 JSContext：

```
swift
let jsContext = webView.value(forKeyPath: "documentView.webView.mainFrame.javaScriptContext")
```

而 WKWebView 中，JSContext 运行在不同的进程当中。我们只能通过 WKWebView 的其他方法实现。

直接/立即调用

WKWebView 提供了十分便捷的方法，来执行 JS 的方法。

```
swift
open func evaluateJavaScript(_ javaScriptString: String, completionHandler: ((Any?, Error?)
```

[首页](#) ▼[探索掘金](#)

- 执行 js 代码的运行环境是，web 页面的 **全局执行上下文**。所以不管是否包含 **window.**，它调用/操作的都是 window 的函数/变量。
- 执行的过程是异步的，因为实际 JS 的执行是在不同的进程中执行的。而 completionHandler 总是在主线程的，可以放心执行 UI 操作。

如果需要传递参数，需要先转化为 JSON 字符串

```
let data = JSON(dataToJs).rawString()!  
webView?.evaluateJavaScript("jsFunc(\(data))", completionHandler: { (ret, err) in  
    debugPrint(err ?? ret ?? "")  
})
```

swift

注入脚本

某些情况下，我们需要注入一些 js 代码到 web 页面的运行环境当中。

```
let scrip = WKUserScript(source: sourceCode, injectionTime: .atDocumentStart, forMainFrameOnly: true)  
webView?.configuration.userContentController.addUserScript(scrip)
```

swift

这一过程实际上与直接执行一段 JS 效果是一样的，但时机比较早，而可以在特定时机时执行：

.atDocumentStart: Inject the script after the document element has been created, but before any other content has been loaded.

如果注入的时机是 **.atDocumentStart**，它是除了 DOM 树创建之外，其他资源加载完成之前。也就是说，早于其他任何 js 的执行。

.atDocumentEnd Inject the script after the document has finished loading, but before any subresources may have finished loading.

值得注意的是，它还可以注入到所有 frame 当中

js 的执行时机

为了理解原生在调用 JS 时候，尤其是注入 JS 时候的执行时机。我们在 web 页面入口文件中添加下列代码。

```
        console.log('document.readyState: ' + document.readyState);
    })
    document.addEventListener('DOMContentLoaded',function() {
        console.log('DOMContentLoaded');
    })
    window.addEventListener('load',function() {
        console.log('window.load');
    })
}
```

可以看到控制台的输出是这样的：

```
[Log] webView didStartProvisionalNavigation a=3
[Log] webView decidePolicyFor navigationResponse a=3
[Log] .atDocumentStart
[Log] main.js
[Log] home vue mounted
[Log] document.readyState: interactive
[Log] DOMContentLoaded
[Log] .atDocumentEnd
[Log] document.readyState: complete
[Log] image load finish
[Log] window.load
[Log] webView didCommit a=undefined
[Log] webView didFinish navigation a=undefined b=89
```

可以看到，执行顺序大概是这样的：

- `webView.load` 加载页面
- 触发 `WKNavigationDelegate` 的 `webView:didStartProvisionalNavigation:` 和 `webView:decidePolicyFor:navigationResponse` 方法
- 执行当 `injectionTime == .atDocumentStart` 时，注入的 JS
- 加载并执行其他 js。譬如 vue 中的 main.js 就是在这时候执行的。
- 首页 `mounted` 事件
- `document.readyState` 的状态为 `interactive`
- document `DOMContentLoaded` 事件，DOM 加载完成
- 执行当 `injectionTime == .atDocumentEnd` 时，注入的 JS
- `document.readyState` 的状态为 `complete`
- 图片等其他资源加载完成
- window.load
- 页面加载完成。触发 `WKNavigationDelegate` 的 `webView:didCommit:` 和 `webView:didFinish:navigation` 方法

- 每一次刷新页面，执行的上下文是不一样的。在用 `evaluateJavaScript` 执行 js 方法时，必须保证页面加载完成，否则使用注入 JS 的方式。
- 页面每次的刷新，都会执行一遍所注入的 js。如果多次注入同一 js，会在加载过程中执行多次。所以注入 JS 尽量在初始化 webView，获取调用 `webView.load` 之前。

JS 调用原生的方法

messageHandlers 方式

在 JS 中，我们可以向指定的方法发送消息。

```
window.webkit.messageHandlers.myNativeMethod.postMessage(message)
```

javascript

其中，webkit.messageHandlers 是在 WKWebView 当中才有的，专门用于处理原生与 web 页面之间数据传递。

而 myNativeMethod 这个方法，需要先在原生代码当中注册（注入）。

```
webView?.configuration.userContentController.add(self, name: "myNativeMethod")
```

swift

当 web 页面 postMessage 的时候，WKScriptMessageHandler 的下列方法，会触发。

```
func userContentController(_ userContentController: WKUserContentController, didReceive message: WKScriptMessage) {
    if message.name == "myNativeMethod" {
        let body = message.body
        self.myNativeMethod(body)
    }
}
```

swift

如果方法名匹配的话，我们就可以调用指定的原生方法了。

拦截请求的方式

和 UIWebView 时代的方法一样。我们可以通过拦截 web 页面的 document 请求来达到调用原生方法的目的。当 web 页面中通过改变 document 的 location 来通知原生，传递数据，以此来实现调用原生方法的目的。

javascript



首页 ▾

探索掘金



这个过程，可以理解为一次请求。其中 `myApp` 是我们定义的协议名；`myNativeMethod` 类似于网络请求的子路径，表示要执行的方法名。后面的 `query` 是需要传递的参数。

然后，我们可以在 `WKNavigationDelegate` 的 `webView:decidePolicyFor navigationAction:decisionHandler` 方法中，监听到这个请求

```
swift
func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction, de
    let url = navigationAction.request.url
    if url?.host?.lowercased() == "myapp"{
        if url?.host?.lowercased() == "myNativeMethod" {
            var params = [String:String]()
            url?.query?.components(separatedBy: "&").forEach({
                let arr = $0.components(separatedBy: "&")
                params[arr[0]] = arr.count >= 1 ? arr[1] : ""
            })
            self.myNativeMethod(params)
        }
        decisionHandler(.cancel)
    } else {
        decisionHandler(.allow)
    }
}
```

当我们发现，请求的协议名是我们的，并且匹配到方法名时，就可以调用相应的原生方法了。

值得注意的是，当发现是我们专门用于 web 页面交互的协议时，需要把这个请求取消掉。不然 web 页面会因此抛出加载异常。

当然，我们也可以通过 `iframe` 来实现这个过程。方法是先添加一个不会展示的 `iframe` 元素。

```
javascript
window.messagingIframe = document.createElement('iframe');
messagingIframe.style.display = 'none';
document.documentElement.appendChild(messagingIframe);
```

当需要调用原生方法时

```
javascript
messagingIframe.src = "myApp://myNativeMethod"
```

WebViewJavascriptBridge



首页 ▾

探索掘金



WebViewJavascriptBridge 是我们在做与 web 页面交互时，经常使用的框架。由于它包含了三端的代码，可以大大提交我们的开发效率。

原生调用JS

首先需要在 web 页面的 JS 代码中，注册方法：

```
javascript
setupWebViewJavascriptBridge(bridge => {
  bridge.registerHandler('jsFunc',(data, responseCallback) =>{
    console.log('jsFunc');
    console.log(data);
    responseCallback("suceess");
  })
})
```

其中setupWebViewJavascriptBridge 是 WebViewJavascriptBridge 的初始化过程。所有操作必须在初始化完成之后进行：

```
javascript
function setupWebViewJavascriptBridge(callback) {
  if (window.WebViewJavascriptBridge) { return callback(WebViewJavascriptBridge); }
  if (window.WVJBCallbacks) { return window.WVJBCallbacks.push(callback); }
  window.WVJBCallbacks = [callback];
  var WVJBIframe = document.createElement('iframe');
  WVJBIframe.style.display = 'none';
  WVJBIframe.src = 'https://__bridge_loaded__';
  document.documentElement.appendChild(WVJBIframe);
  setTimeout(function() { document.documentElement.removeChild(WVJBIframe) }, 0);
}
```

然后，是在原生端调用：

```
swift
self.bridge = WKWebViewJavascriptBridge(for: webView)

self.bridge?.callHandler("jsFunc", data: dataToJs, responseCallback: { (result) in
    debugPrint("jsFunc:\(result ?? "")")
})
```

JS调用原生

首先，还是先注册



首页 ▾

探索掘金



swift

```
self.bridge?.registerHandler("nativeFunc", handler: { (data, callBack) in
    debugPrint("nativeFunc:\(data ?? "")")
    callBack?(dataToJs)
})
```

然后，在 web 页面中调用

javascript

```
window.WebViewJavascriptBridge.callHandler('nativeFunc',data)
```

基本原理

在 WebViewJavascriptBridge 中，web 与原生交互都被抽象为发送/接受消息。Web 页面向原生发送消息，都是通过改变 DOM 的 location 实现的。唯一不同的是，它使用的是 iframe。

消息类型	协议名	旧版协议名	子路径
初始化消息	https	wvjbscheme	bridge_loaded
普通消息	https	wvjbscheme	wvjb_queue_message

当收到初始化消息时，原生端会注入 JS，创建用于桥架的 WebViewJavascriptBridge JS 对象。

当收到普通消息时，并不是通过 URL query 来获取请求参数。而是通过一个专门的 JS 方法。

javascript

```
WebViewJavascriptBridge._fetchQueue();
```

这样做的好处显而易见。比 URL query 方式可以传递更为复杂的数据结构。

值得注意的是，由于 js 向原生发送消息过程是异步的。有可能从发送到接受过程中，发送了多个请求。所以必须要有个消息队列，也就是一个数组，用于缓存消息。在接受数据的时候再进行拆包和情况消息队列。

原生向 JS 发送消息，也是通过特定的 JS 方法：


javascript

```
WebViewJavascriptBridge._handleMessageFromObjC();
```

判断document加载过程的几个不同方法

关注下面的标签，发现更多相似文章

Webkit



chaocai Lv2

iOS开发者 @ 某公司
获得点赞 99 · 获得阅读 17,768

关注

安装掘金浏览器插件

打开新标签页发现好内容，掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧！

输入评论...

相关推荐

伤心的Easyman · 3天前 · Webkit

WKWebview秒开的实践及踩坑之路

👍 27

💬 12

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

 53

 6

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

 544

 38

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

一只只有交流障碍的丑程 · 5月前 · Webkit

Android 中的Cookie了解一下

 9

 1

