iOS におけるハイブリッドアプリケーションの開発支援

齋藤　暢郎 †　小出　洋 ††

モバイル端末及び，タブレット端末用の OS である iOS は，ユーザーが任意にアプリケーションを追加することで，PC と同様に，端末の機能を拡張できるという特徴がある．このアプリケーションは，AppStore からインストールするネイティブアプリケーションと，Safari 等のブラウザ上で動作する Web アプリケーションの二種類に分類することができる．Web アプリケーションは Apple 社による審査が必要ないという利点がある反面，利用できる API の種類や動作の自由度が限定されるため，Web アプリケーションの品質やユーザーエクスペリエンス (UX) はネイティブアプリケーションと比較して一般的に大きく劣る．ネイティブアプリケーションの一部に Web アプリケーションを組み込む事で，アプリケーションの UX を維持しつつ，審査にかかるコストを下げるハイブリッドアプリケーションと呼ばれる開発手法も存在する．現在，ハイブリッドアプリケーションの多くは Web-ネイティブ間の複雑な連携を確立しているものは少ない．これは Web-ネイティブ間の連携を実現するとプログラムソースが煩雑になる，iOS の WebKit に対しての理解がないと実装ができない等の原因が挙げられる．Web-ネイティブ間の複雑な連携が簡単に実装できれば，ハイブリッドアプリケーションの開発効率が向上するものと考え，JavaScript に Objective-C のコードを呼び出す関数を実装する事のできるフレームワークを実現した．また，そのフレームワークを利用した拡張ブラウザアプリケーションを実装して，Web アプリケーションを実行することにより，Objective-C で記述された API の動作を確認した．

Nobuo Saito † and Hiroshi Koide††

iOS, the OS for mobile or tablet device has feature as with PC that there users can add any application, and extend the functionality of the terminal. This application is divided to native application that installed from AppStore, and Web application that run on the browser such as Safari. Web application has advantage that do not need to be reviewed by Apple.inc, but the degree of freedom of movement and type of API that can be used is limited, the user experience (UX) and quality are generally compared to the native application of Web applications it is inferior to. Development methodology native application incorporating Web application called hybrid application. The hybrid application can maintain UX of the application while lowering the cost of update. Currently, there are only few hybrid application that establish a complex cooperation between Web and native. The cause of this is something like the following: if you implement cooperation between web and native, the program source becomes complicated, or If you have no understanding of WebKit for iOS, you can not implement it. In this study, I made the framework that can implement a function that calls the code in Objective-C to JavaScript, because if we can be coordination complex between Web and native easily, efficient development of hybrid application is improved. In addition, I implemented extended browser application that using the framework, and ran the Web application on the application, verified the operation of the API, which is written in Objective-C.

1. はじめに

iOS 向けにアプリケーションを開発する場合，ハイブリッドアプリケーションを開発するという選択肢は幾つかの大きな利点がある．例えば，HTML のプログラム資産をそのままネイティブアプリケーションで活かせるといった事や，サーバー・クライアントモデルが容易に構築できる，或いはアップデートにかかる審査のコストが比較的軽く済むといった事が挙げられる．しかし，利点の中で最大といっても差し支えない，Web 環境とネイティブ環境の連携は，現在多くのハイブリッドアプリケーションで実現されておらず，一部のアプ

† 九州工業大学 情報工学部
Kyushu Institute of Technology
tarunon@klab.ai.kyutech.ac.jp
†† 九州工業大学 大学院情報工学研究院
Kyushu Institute of Technology
hiroshi@klab.ai.kyutech.ac.jp

リケーションしか実現できていない．PhoneGap[1] 等の既存のフレームワークを用いる事で，一部の Cocoa Touch[2] の API を JavaScript から呼び出すことは可能であるが，任意の処理を Objective-C で処理したい等のニーズには応えにくい．既存のフレームワークを用いずに独自に Web-ネイティブ間の連携を実現しようとなると，JavaScript・Objective-C 双方のプログラムソースが煩雑になったり，或いは WebKit の知識が無ければ実装すらできないといった問題が発生していた．本研究では，JavaScript から Objective-C のコードを呼び出すためのフレームワークとして，JS2ObjC を開発した．JS2ObjC は，JavaScript に任意の関数を追加し，その関数を呼び出すことで Objective-C 側で登録した操作を行い，結果を JavaScript で受け取るといった相互的な連携を実現する．また，JS2ObjC を利用したアプリケーションを開発し，JavaScript から Objective-C のコードを動作させることが可能であることを確認した．本論文の構成は以下のとおりである．第 2 章では，JavaScript と Objective-C における言語間の連携手法について説明し，第 3 章では，本研究で開発したフレームワークである JS2ObjC の機能及び実装方法を紹介する．第 4 章では，JS2ObjC を搭載した拡張ブラウザアプリケーションの動作を検証し，第 5 章では，類似のフレームワークである PhoneGap との比較を行い，そして第 6 章で本研究についてのまとめと課題を述べる．

2. JavaScript と Objective-C における言語間の連携手法

本研究の根幹は，JavaScript のコードから Objective-C のコードを呼び出すという点である．これにより，Web-ネイティブ間の連携が可能となり，ハイブリッドアプリケーションの利点をより活かせるものと考える．これを実現するには，JavaScript と Objective-C のコードの間で，相互通信を行う必要がある．本章では，Objective-C から JavaScript の関数を，また，JavaScript から Objective-C のメソッドを呼び出す手法を紹介し，本研究で考案した言語間の相互的な連携手法，またそれを可能とする iOS における WebKit の特徴について述べる．以後，JavaScript の関数/メソッドを「関数」，Objective-C の関数/メソッドを「メソッド」と分けて表記する．

2.1 Objective-C から JavaScript への接続

Objective-C から JavaScript のコードを呼び出すためのプリミティブは，Cocoa Touch でメソッドとして提供されている．Cocoa Touch では，Web ページを表示する際に `UIWebView`[3] というクラスを利用する．このクラスは，WebKit ベースで Web ページを表示するクラスであり，iOS で Web ページを表示する機能をもたせる場合はこのクラスを利用する．この `UIWebView` には，開発者及びユーザーが，任意の JavaScript を実行する為に `stringByEvaluatingJavaScriptFromString:` というメソッドが用意されている．これを用いることで，`UIWebView` を経由し，Objective-C から JavaScript を実行することが可能である．なお，`stringByEvaluatingJavaScriptFromString:` の返り値は，JavaScript の実行結果を `NSString`(Objective-C における文字列クラス) に格納したものである．数値や文字列を返すことは可能であるが，テーブルや配列を返すことはできない．

2.2 JavaScript から Objective-C への接続

一方で，JavaScript から Objective-C のコードを呼び出す方法は，Cocoa Touch の中には含まれていない．これを実現するには，`UIWebViewDelegate`[4] のメソッド `webView:shouldStartLoadingWithRequest:navigationType:` を利用する．`webView:shouldStartLoadingWithRequest:navigationType:` は，Web ページが新しい URL を読み込もうとする際に呼び出され，第一引数に呼び出した `UIWebView` のインスタンス，第二引数に URL やリファラー等の情報を含む `NSURLRequest` のインスタンス，第三引数に読み込みの種類 (リンククリック，フォームの送信等) が格納される．このメソッドの返り値は BOOL 型で，NO を返した場合は URL の読み込みを行わない．JavaScript から Objective-C へ連携する手段は，事前にこのメソッドに特殊な URL を読み込んだ場合の処理を記述しておき，JavaScript 側でその特殊な URL を読み込むことで可能となる．この時の URL は，http ではない特殊なスキーム，呼び出す関数の識別子，引数の形で実装すると良い．例えば，`objc://function?arg1&arg2&arg3` といった形である．この URL の形は，Objective-C のクラス，`NSURL` のメソッド `host` や `query` を利用して，関数名や引数を取得することが可能である．

2.3 WebKit の特徴

JavaScript 側で特殊な URL を読み込む際に，単に location 変数を変更しただけは期待通りの動作は行われない．これは，WebKit では location 変数の

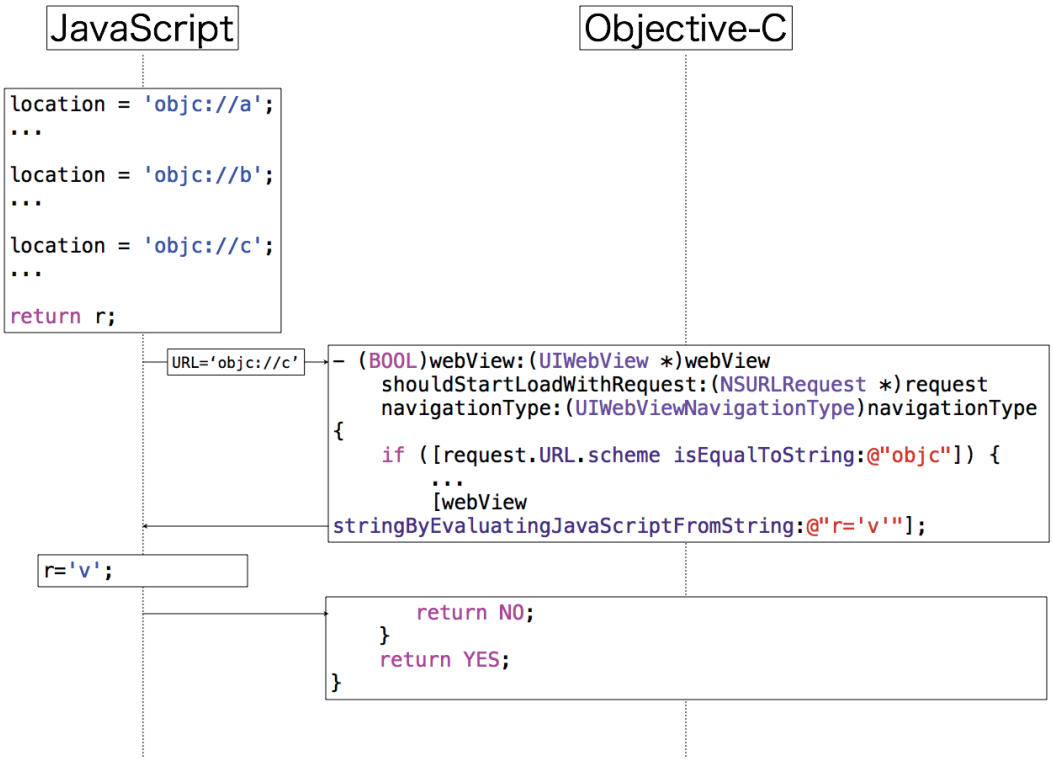

図 1 location 変数を変更した場合は相互的な連携は不可能

変更が，JavaScript の実行後にチェック，読み込みが行われる事による．この場合，一度の JavaScript 実行に対して，複数回 Objective-C のコードを呼びだそうとすると，最後の一つのみが実行されることになる．その上，Objective-C が動作するのが JavaScript 実行後であるため，相互的な連携は不可能である (図 1)．これは，URL の読み込みをリンククリックや，インラインフレームの読み込み，新しいウィンドウの生成とすることで回避が可能である．これらの方法をとった場合，JavaScript の動作は `webView:shouldStartLoadingWithRequest:navigationType:` の終了まで停止する．この手順は，既存のアプリケーションでも幾つか利用されているものであるが，本研究ではこれに加えて，JavaScript の停止中に Objective-C から `stringByEvaluatingJavaScriptFromString:` を利用し，任意の JavaScript の実行を割り込ませることが可能であることを確認した (図 2)．これは，WebKit が JavaScript の実行や URL の読み込み等を，全てメインスレッドで逐次実行していることに依存する．

3. JS2ObjC

本研究では，JavaScript と Objective-C の連携を統括的に管理するために，JS2ObjC と名付けたフレームワークを開発した．JS2ObjC は現在 iOS5.0 以降で動作する．JS2ObjC には，前章で紹介した手法を用いた 2 言語間の連携の他に，2 言語間の変数のキャストや Objective-C から JavaScript の利用を拡張するためのクラスが含まれる．本章ではこの JS2ObjC の機能及び実装方法を紹介する．付録に JS2ObjC のフレームワークを添付したので，利用や詳細な実装については参考にされたい．

3.1 JavaScript と Objective-C の接続機能

Web 側に URL を呼び出す為の JavaScript 関数を設置し，ネイティブ側に URL を解釈するインタプリタを実装することで，Web-ネイティブ間の連携が可能となる．この Web・ネイティブ双方への操作を統括的に管理するのが JS2ObjC の役割である．JS2ObjC では，メソッド `addJSFunctionName:usingBlock:` を利用することで，任意の Objective-C のブロック構文 (Objective-C における関数オブジェクト)

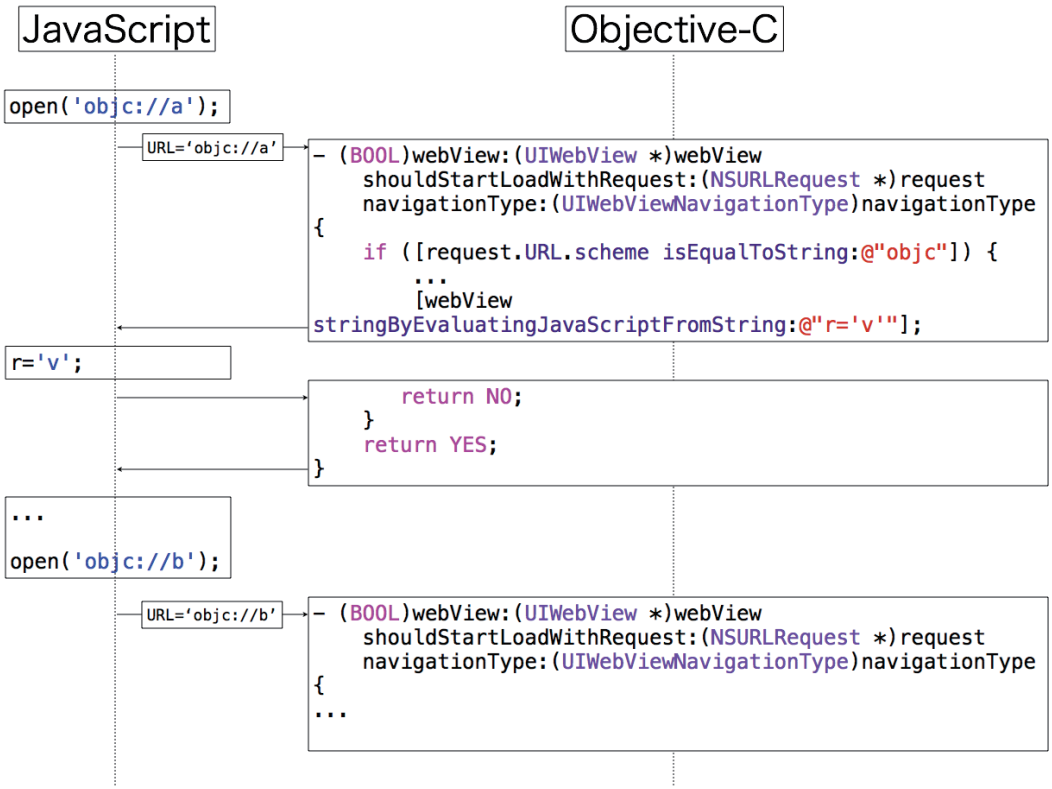

図 2 新しい window を生成する場合は任意に JavaScript を割りこませることが可能

を JavaScript の関数に紐付けることができる．addJSFunctionName:usingBlock:を呼び出すと，NSDictionary(Objective-C におけるテーブルクラス)の関数テーブルに関数名をキーとしてブロック構文が登録される．このブロック構文は JavaScript 側で対応する関数が実行された際に呼び出され，第一引数に JavaScript の引数を並べた NSArray，第二引数に JavaScript を実行した UIWebView が与えられる．対応する関数からブロック構文を呼び出す方法は，A タグを生成してクリックイベントを発生させる手法を採用した．A タグが持つ URL の規定は JS2ObjC://Function/ID?Arguments である．Function は関数名であり，関数テーブルからブロック構文を取り出すキーとして利用される．ID は，ページ読み込み毎に JavaScript で生成されるランダムな文字列で，インラインフレーム等からの操作を阻止する役割を持つ．Arguments は JavaScript 側で与えられた引数で，引数を配列として JSON 文字列化したものである．これをパースして NSArray に格納し，ブロック構文の第一引数に与える．また，ブロック構文の返り値は JavaScript 側で関数の実行結果として受け取る事が可能である．これは，JS2ObjC で設定した JavaScript の関数を呼び出した際に，JavaScript のグローバル変数に呼び出した関数自身を一時的に格納し，このグローバル変数から関数内のプロパティを参照し，ブロック構文の返り値を stringByEvaluatingJavaScriptFromString:を利用して代入することで実現している．(図 3).

3.2 クラスに対する拡張機能

JavaScript は new 演算子を用いることで，関数をクラスとして利用することが可能である．JS2ObjC では，JavaScript において JS2ObjC で実装した関数をクラスとして利用する場合，クラス内への関数や変数の実装を支援する機能を持っている．JSClass インスタンスに対して addJSFunctionName:usingBlock:を呼び出す事で，JavaScript のクラス内に関数を追加して実装できる．JSClass は，JS2ObjC のスーパークラスであり，addJSFunctionName:usingBlock:の返り値としてインスタンスを得る．JSClass インスタンスに対してこのメソッドを呼び出した場合，prototype を利用して，JSClass インスタンスに対

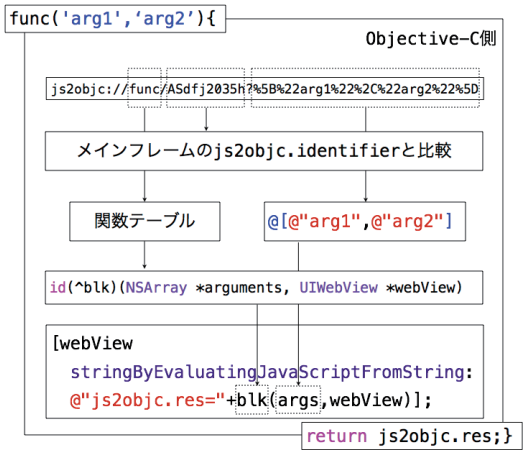

図 3 JS2ObjC で追加した関数の動作イメージ

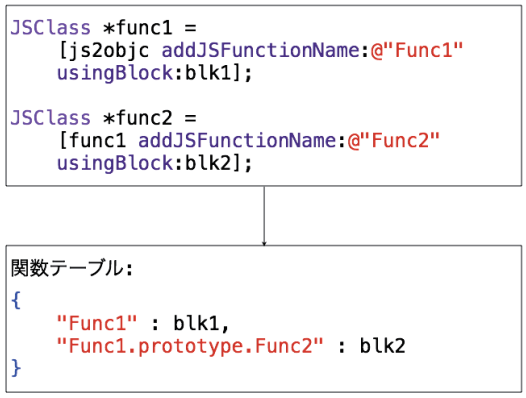

図 4 JSClass インスタンスでメソッドを呼ぶと，その prototype に関数が実装され，関数テーブルのキーは prototype を含めたものとなる．

応する JavaScript クラスに関数を追加する．(図 4)． また，UIWebView に setJSProperty:forKey:，jsPropertyForKey:，jsSelfObject のメソッドを実装した．これらは全て addJSFunctionName:usingBlock:に与えたブロック構文内で利用するもので，setJSProperty:forKey:によりクラス変数の追加，編集，jsPropertyForKey:で任意のクラス変数の取得，jsSelfObject で関数の呼び出し元を参照する事ができる．

3.3 JavaScript と Objective-C 間における変数のキャスト

JS2ObjC では JavaScript と Objective-C の間において，テーブルや配列での値の受け渡しを可能とするため，双方向で の変数のキャストをサポートしている．JSON を利用することで，JavaScript-Objective-C 間において，Array と NSArray，Object と NSDictionary をそれぞれ変換することを可能とした．Objective-C 側では NSJSONSerialization[5] を利用し，JSON 文字列から Objective-C のインスタンスを生成する．JavaScript 側では，stringByEvaluatingJavaScriptFromString:で JavaScript を実行する際に JSON 文字列の評価が行われるので，特に実装する必要はない．JSON 文字列の生成は，次項に記述している関数の実行を実現するため，JSON フレームワークのものではなく自作の関数で行った．変数が循環構造を持つ場合は JSON 文字列を生成することができない．

3.4 JavaScript 関数を実行するための拡張機能

JS2ObjC では，JavaScript 側から関数の引数として，別の関数を与える場合を想定し，JavaScript 側から受け取った関数を実行するためのクラスとして JSFunction クラスを実装した．JSFunction クラスは JSClass のサブクラスであり，インスタンスの生成時に関数名と動作する UIWebView インスタンスを与える．メソッド runWithArguments:，makeNewValue:withArguments:で，JavaScript 側で関数として実行，或いはグローバル環境にインスタンスを生成することができる．関数として実行した場合は，Objective-C 側で返り値を処理するため，前項の変数のキャストを用いて値を変換する．JSFunction インスタンスに与える関数名は，グローバル環境から参照できる必要がある．JavaScript 側から関数を受け取る際に，一次関数や局所関数を受け取った場合，グローバル環境から参照することができなくなるが，JS2ObjC では JSON 文字列を生成する際に，オブジェクトが関数であった場合に，グローバル変数に用意した配列に一度格納して，その変数名を JSON 文字列に含めることでこれを回避している．(図 5)．Objective-C 側で JSON をパースした後に，生成されたインスタンスを探索し，この変数名が含まれる場合，JSFunction インスタンスを生成して置換する．この JSFunction インスタンスを runWithArguments:メソッドに引数として与えた場合，JSON 文字列を生成する際に関数名をダブルクオーテーションで囲まずに含めることで，JavaScript で解釈した際に関数として扱うことが可能となる．また，JSFunction クラスが開放される際に，JavaScript 側に通知を投げ，関数を格納する配列のインデックスに対して不使用フラグを立て，次の関数を格納する際に再利用することで効率を改善している．

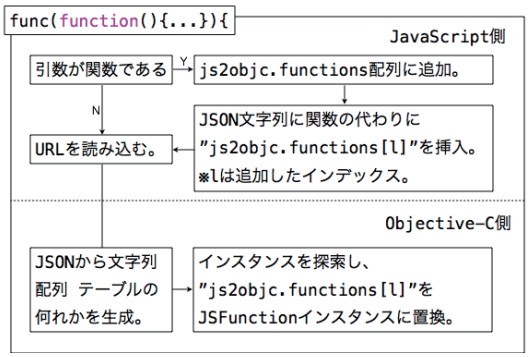

図 5 引数に関数を含む場合の処理.

3.5 Method Swizzling による開発者の負担軽減

前章でも述べたように，JS2ObjC の実装にあったって，`UIWebViewDelegate` のメソッド，`webView:shouldStartLoadingWithRequest:navigationType:` における URL の監視は必須である．しかし，`UIWebViewDelegate` は，開発者によって任意に設定したいものであり，そのそれぞれの `webView:shouldStartLoadingWithRequest:navigationType:` に開発者自身の手によって URL 判定用のメソッドを差し込むのは僅かではあるが手間である．JS2ObjC では，この手間を無くすために，Method Swizzling(`http://cocoadev.com/wiki/MethodSwizzling` 2012/09/05 確認) を利用した．Method Swizzling とは，実装済みであるメソッドを，任意のメソッドと入れ替える事のできる Objective-C のプログラミング手法である．Objective-C では，クラスが保有するメソッドはメソッド名を表す SEL と，メソッドを実行する C の関数である IMP で構成されている．Method Swizzling では，2 つのメソッドにおいて，お互いの IMP を交換することで，メソッドの入れ替えを実現している．JS2ObjC では，インスタンスの初期化時に，これを利用して `UIWebView` の 2 つのメソッド，`init` と `setDelegate:` を Swizzling する．`init` はインスタンスの初期化メソッドである．これを呼び出した際に，`UIWebViewDelegate` に JS2ObjC を設定し，`stringByEvaluatingJavaScriptFromString:` により登録された JavaScript 関数を読み込む．`setDelegate:` はデリゲート指定メソッドである．これを呼び出した際に，第一引数に指定されるデリゲートのインスタンスが格納されているので，デリゲートクラスの `webView:shouldStartLoadingWithRequest:navigationType:` を Swizzling する．また，デリゲートクラスに対しての Method Swizzling では，クラスが複数個ある場合も考えられるので，入れ替えるメソッドをクラス毎に動的に追加している．`JS2ObjC` と `UIWebView` のインスタンスを初期化するとメソッドの動きは図 6 の様になる．`JS2ObjC` のインスタンスの初期化により，UIWebView のメソッドが Method Swizzling され，この状態で `UIWebView` のインスタンスを初期化すると，`UIWebViewDelegate` に `JS2ObjC` インスタンスが設定され，さらに `JS2ObjC` クラスの `UIWebViewDelegate` メソッドが Method Swizzling される．入れ替えたメソッドを呼び出す場合，入れ替えたメソッドの最初か最後で必ずオリジナルのメソッドも呼び出すので，プログラムの動作に支障をきたす事はない．

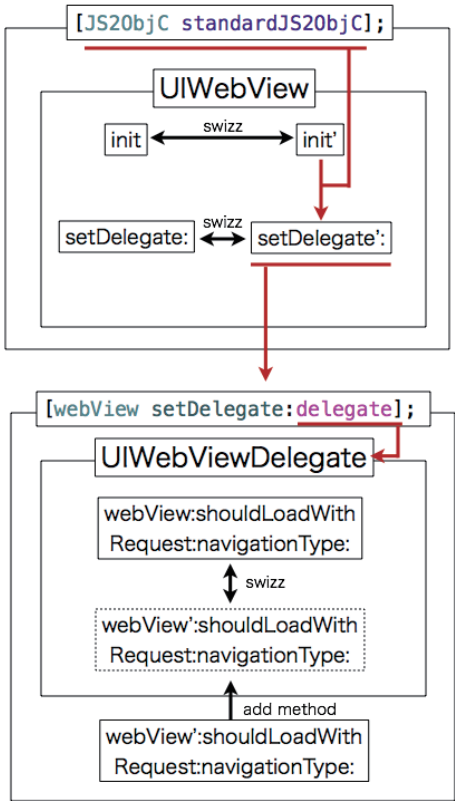

図 6 UIWebView と UIWebViewDelegate の Method Swizzling

3.6 JavaScript 関数の設置方法

JS2ObjC は登録した JavaScript の関数を Web ページ内に追加する機能が必要である．これを実現するために，URL loading system 内に例外処理を追加した．Cocoa Touch の URL loading system[6] では，何らかのクラスが URL からデータを呼び出す場合に，`NSURLProtocol`[7] インスタンスを生成し，データをダウンロードする．この `NSURLProtocol` は，クラスメソッド `registerClass:` に `NSURLProtocol` サ

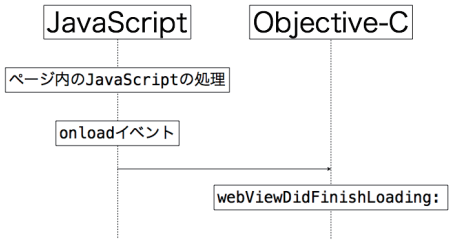

図 7 JavaScript の処理の時系列

ブクラスを与えることで，任意のプロトコルを実装することが可能となっている．`NSURLProtocol` サブクラスのクラスメソッド，`canInitWithRequest:`内に通常の URL として読み込まないものとして分類することで，URL loading system に例外を追加することができる．JS2ObjC では，例外としてパスが `js2objc.js` であれば，リンク先からダウンロードせずに `JS2ObjC` インスタンス内で生成した JavaScript の文字列を読み込む様にプログラムしている．これにより，JS2ObjC を利用する Web ページの HTML に，`<script type="text/javascript" src="js2objc.js"></script>`を記述するだけで JS-2ObjC で実装した機能が利用できるようになる．また，開発者及びユーザーが，任意の Web ページにおいて JS2ObjC の機能を利用することも考えられるので，`UIWebViewDelegate` のメソッド，`webViewDidFinishLoading:`を Method Swizzling し，その中で `stringByEvaluatingJavaScriptFrom-String:`を利用してページ読み込み後に JavaScript 関数を追加する方法を併用した．ただし，`webViewDidFi-nishLoading:`を Method Swizzling する方法のみで実装した場合，図 7 の様に，ページ内の JavaScript の処理や，onload イベントが完了してからページ内に JS2ObjC で実装した関数が追加されるので，この方法のみで実装するのは不適切である．

4. 拡張ブラウザアプリケーション

前章で紹介した JS2ObjC フレームワークを利用し，JavaScript から Cocoa Touch フレームワークの一部を利用できる拡張ブラウザアプリケーションを開発した．本章ではこの拡張ブラウザアプリケーションの追加機能と，実装における注意点を紹介する．拡張ブラウザアプリケーションの追加機能である JavaScript は，基本的に引数の省略はできない設計とした．JS2ObjC を利用することで，拡張ブラウザアプリケーションは簡易に JavaScript から Cocoa Touch の機能を呼び出すことを実現している．実装した機能のうち，比較的簡単なコードを紹介する．図 8 は，JavaScript にバッテリー API を実装しているものである．単純な API のバイパスであれば，JS2ObjC を利用することで非常に簡単に実現することが可能となっている例である．

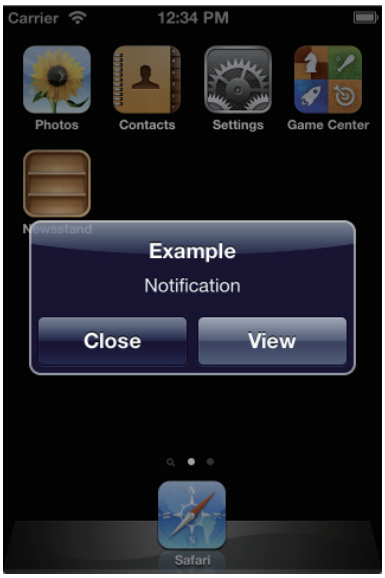

図 10 `localNotification` 関数の動作例

図 9 は，通知センターを利用する関数を作成しているものである．通知をタップした場合に，notifyAc-tions テーブルから通知のハッシュ値をキーとして参照し，`JSFunction` クラスを取り出し実行する設計となっている．通知センターを利用できたか否かで，返り値を変化させ，Web アプリケーション側でこの結果を用いて分岐することを可能とした．アプリケーションがバックグラウンドにある場合，図 10 の様に通知が表示される．

図 11 は，UIActionSheet を表示する関数を作成しているものである．`BlockActionSheet` はブロック構文でボタンに対応するアクションを記述できる `UIActionSheet` のサブクラスである．この関数は，`UIActionSheet` の処理が完了するまで，45-47 行目で `NSRunLoop` を用いて JavaScript の動作を停止させている．JS2ObjC では，`NSRunLoop` を用いることで任意に JavaScript の動作を停止させ，その間に別の UI を動作させることが可能である．`actionSheet` を利用することで，図 12 の様に `UIActionSheet` を利用することができる．

これらのフレームワークは，Safari 上で動作する Web アプリケーションでは利用することのできないものである．JS2ObjC を利用し，事前に API を準備

```
66: JSClass *battery = [js2objc addJSFunctionName:@"Battery" usingBlock:^id(NSArray *arguments, UIWebView *webView)
67: {
68:     [UIDevice currentDevice].batteryMonitoringEnabled = YES;
69:     [webView setJSProperty:@([UIDevice currentDevice].batteryLevel) forKey:@"level"];
70:     [webView setJSProperty:@([UIDevice currentDevice].batteryState) forKey:@"state"];
71:     return @"1";
72: }];
73:
74: [battery addJSFunctionName:@"reload" usingBlock:^id(NSArray *arguments, UIWebView *webView) {
75:     [webView setJSProperty:@([UIDevice currentDevice].batteryLevel) forKey:@"level"];
76:     [webView setJSProperty:@([UIDevice currentDevice].batteryState) forKey:@"state"];
77:     return @"1";
78: }];
```

図 8 JS2ObjC による JavaScript 拡張例 1

```
54: [js2objc addJSFunctionName:@"localNotification" usingBlock:^id(NSArray *arguments, UIWebView *webView) {
55:     if (application.applicationState == UIApplicationStateBackground) {
56:         UILocalNotification *notif = [[UILocalNotification alloc] init];
57:         notif.alertBody = [arguments objectAtIndex:0];
58:         [_self.notifyActions setObject:[arguments objectAtIndex:1] forKey:@(notif.hash)];
59:         [application presentLocalNotificationNow:notif];
60:         return @"1";
61:     }
62:     return @"0";
63: }];
```

図 9 JS2ObjC による JavaScript 拡張例 2

```
27: [js2objc addJSFunctionName:@"actionSheet" usingBlock:^id(NSArray *arguments, UIWebView *webView) {
28:     __block NSInteger _return;
29:     BlockActionSheet *sheet = [[BlockActionSheet alloc] initWithTitle:[arguments objectAtIndex:0]
    usingBlock:^(NSInteger buttonIndex, BlockActionSheet *actionSheet) {
30:         _return = buttonIndex;
31:         _self.activeActionSheet = NO;
32:     }];
33:     sheet.destructiveButtonIndex = [[arguments objectAtIndex:1] integerValue];
34:     sheet.cancelButtonIndex = [[arguments objectAtIndex:2] integerValue];
35:     if (arguments.count > 3) {
36:         NSArray *titles = [arguments objectAtIndex:3];
37:         [titles enumerateObjectsUsingBlock:^(NSString *title, NSUInteger idx, BOOL *stop) {
38:             [sheet addButtonWithTitle:title];
39:         }];
40:     } else {
41:         [sheet addButtonWithTitle:@" "];
42:     }
43:     _activeActionSheet = YES;
44:     [sheet showInView:webView];
45:     while (_activeActionSheet) {
46:         [[NSRunLoop currentRunLoop] runUntilDate:[NSDate dateWithTimeIntervalSinceNow:0.1]];
47:     }
48:     return @(_return);
49: }];
```

図 11 JS2ObjC による JavaScript 拡張例 3

しておく事で，JavaScript から Objective-C のコードを呼び出すことに成功した．こういった Web-ネイティブ間連携を利用することで，ハイブリッドアプリケーションのプログラムの可能性は飛躍的に向上すると考えられる．

5. PhoneGap との比較

本研究と類似したフレームワークとして，Adobe 社の PhoneGap が挙げられる．PhoneGap は HTML5 で作成した Web アプリケーションを，iOS や Android 等，マルチプラットフォームに対応したネイティブアプリケーションへパッケージングするためのフレームワークであり，JavaScript を拡張して，Cocoa Touch の API を呼び出す，或いはプラグインを開発し任意の Objective-C のコードを実行することが可能となっている．PhoneGap と JS2ObjC が大きく異る点は，拡張した JavaScript 関数が同期的な実行が可能であるか否かという点である．PhoneGap で拡張した JavaScript 関数は，返り値を持たず，実行結果はコールバック関数に引数として格納される非同期的な実行である．一方，JS2ObjC は拡張した JavaScript 関数は返り値を持ち，コールバック関数を必要としない同期的な実行が可能である．JS2ObjC はこの点において，JavaScript では PhoneGap よりも短くコードが書けるという利点が挙げられる．これは JavaScript と Objective-C 間の通信方法に違いがあるためである．JS2ObjC は前述のとおり，A タグに対するクリックイベントを `webView:shouldLoadRequest:navigationType:` で

```
var i = actionSheet(
  'ActionSheet',  // タイトル
  0,              // メインボタンの位置
  2,              // キャンセルボタンの位置
  ['MainButton','SubButton','Cancel']
                  // ボタンのタイトル
);
/*iに押したボタンのインデックスが格納される。*/
if (i==0) {
  ...
} else if (...
```

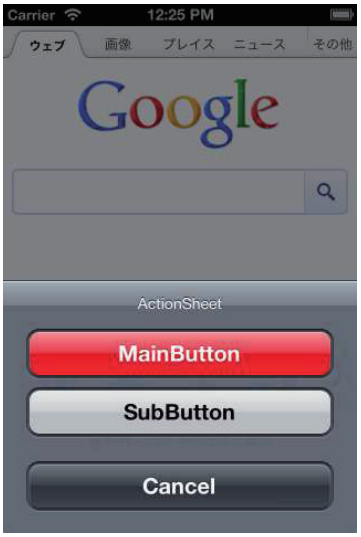

図 12 actionSheet 関数の実装，動作例．

フックする方法を用いているが，PhoneGap では XMLHttpRequest を用いてデータをリクエストのヘッダーに埋め込み，これを NSURLProtocol で処理するという方法を採っている．NSURLProtocol での処理中は，JavaScript の動作を停止させることはできず，このためコールバック関数が必要となっている．また，処理中に別の処理を実行させないために，関数キューが必要となるなど，複雑な実装となっている．また，Web 環境に JavaScript 関数を追加する方法も異なる．PhoneGap では別途 JavaScript のファイルを読み込んでいるが，JS2ObjC は前述のとおりフレームワークが JavaScript の文字列を生成し，js2objc.js を参照することで URL ローディングの代わりに文字列を読み込む仕様となっている．これにより，JS2ObjC はプログラム実行中の動的なフレームワークの変更に対応することが容易となった．

6. おわりに

本研究で開発した，JS2ObjC を利用することで，簡単に JavaScript のコードから Objective-C のコードを呼び出し，相互的な連携を取ることが可能となった．また，それを応用し，いくつかの Cocoa Touch のフレームワークを組み込んだ Web アプリケーションからコントロールすることが可能であることを確認した．JS2ObjC を利用することで，ハイブリッドアプリケーションにおける Web-ネイティブ間の連携が非常に簡単に設計できるようになる．本研究の課題を以下に挙げる．

・言語間通信の拡張

本研究では，カメラモジュールや Objective-C で記述された画像処理系のフレームワークと JavaScript との連携は実装できていない．現段階の JS2ObjC は，JavaScript のコードと Objective-C のコードの通信が文字列に限られているので，これらのフレームワークを利用する場合は，画像を JSON 化して通信するか，アプリケーションが Objective-C とサーバーとの間で直接通信を行うか，アプリケーション自体をサーバー化し，Web アプリケーションからアクセスするような形で実装する必要がある．JS2ObjC にサーバー機能を実装することで，より高度なフレームワークの利用を可能にすることが期待できる．現在の実装では，JS2ObjC が生成した JavaScript の文字列を js2objc.js として読み込む際に NSURLProtocol を利用し，URL loading system に例外を追加しているが，これを応用することで Objective-C で生成したデータを，XMLHttpRequest を利用して Web 側で取り出すことが可能となる．

・プラグインの実装

PhoneGap は豊富なプラグインを有しており，アプリケーションを作成するための地盤が完成しているのに対して，JS2ObjC はその基礎しか実装できていない．同期的な実行が可能という利点があっても，プラグインが無ければ開発には利用しにくい．今後は Cocoa Touch の API と接続するプラグインを実装し，これを提供していきたいと考えている．

参考文献

1) Apache Cordova Documentation, http://docs.phonegap.com/en/2.2.0/index.html

2) Cocoa Foundamentals Guide,
https://developer.apple.com/library/prerelease/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/Introduction/Introduction.html
3) UIWebView Class Reference,
https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIWebView_Class/Reference/Reference.html
4) UIWebViewDelegate Protocol Reference,
https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIWebViewDelegate_Protocol/Reference/Reference.html
5) NSJSONSerialization Class Reference,
https://developer.apple.com/library/ios/#documentation/Foundation/Reference/NSJSONSerialization_Class/Reference/Reference.html
6) URL Loading System Programming Guide,
https://developer.apple.com/library/prerelease/ios/#documentation/Cocoa/Conceptual/URLLoadingSystem/Concepts/URLOverview.html
7) NSURLProtocol Class Reference,
https://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSURLProtocol_Class/Reference/Reference.html
8) Threading Programming Guide
https://developer.apple.com/library/prerelease/ios/#documentation/Cocoa/Conceptual/Multithreading/RunLoopManagement/RunLoopManagement.html

付　　録

A.1 JS2ObjC

https://github.com/tarunon/JS2ObjC