Extending your app with **JavaScriptCore**

Alberto Lagos T.

alberto.lagos@gmail.com

MODERNIZING MEDICINE

MODERNIZING MEDICINE

Details

1. What is JSC?
2. Description
    1. JSContext
    2. JSValue
    3. JSManagedValue
    4. JSVirtualMachine
3. What can we do with this?
4. Demo

What's about that?

JavaScriptCore is an Objective-C wrapper around the WebKit engine. Is an easy / fast / safe way to access to the javascript world.

In Apple's Terms:

The JavaScriptCore Framework provides the ability to evaluate JavaScript programs from within Swift, Objective-C, and C-based apps. You can also use JavaScriptCore to insert custom objects to the JavaScript environment.

JSContext

One instance represent an execution environment. Here you can create and use javascript values (vars, functions, etc) and also you can create your own custom native (obj-c & Swift) objects and pass it to the environment. Is analogous to the **window** variable in a browser.

A variable defined in the JavaScript global context is exposed through keyed subscript access in your context (**JSContext**), so basically you can access to them like you access a **NSDictionary.**

```
JSContext *jsContext = [JSContext currentContext];
jsContext[@“age”];
```

JSValue

One instance of a JSValue is a Javascript value. You can use it to convert values like numbers or string between JS and Objc-c or Swift. Also you can wrap native objects to convert or expose it to Javascript (we will see later).

In the last diapo:

```
JSContext *jsContext = [JSContext
currentContext];
JSValue *jsAgeValue = jsContext[@“age”];
int ageInt = [jsAgeValue toInt32];
```

JSValue

JavaScript	JSValue	Objective-C	Swift
String	toString	NSString	String!
Boolean	toBool	BOOL	Bool
Number	toNumber toDouble toInt32 toUInt32	NSNumber double int32_t uint32_t	NSNumber! Double Int32 UInt32
Date	toDate	NSDate	Date!
Array	toArray	NSArray	[Any]!
Object	toDictionary	NSDictionary	[AnyHashable : Any]!
Object	toObject toObjectOfClass:	Custom type	Custom type as Any!

JSManagedValue

JavaScriptCore uses GB so all the references in JS are strong. So **JSValue** will always keep alive as long as you use it.

If you plan to use a **JSValue** as a ivar is better to use a **JSManagedValue** because in simple terms is a *weak* reference to the value that refers to.

But you need to tell to Virtual Machine that the value reference needs to convert to a GC reference.

addManagedReference:withOwner:

JSVirtualMachine

Basically is a self contained environment for JavaScript execution.

If you use this class probably is because you need a concurrent execution of your JS Code or you want to keep track of the object that you're bridging between Obj-c and Swift

One **JSVirtualMachine** can have many **JSContext**.

JSValues can be passed between **JSContext** in the same **JSVirtualMachine**.

JSValues can't be passed between different **JSVirtualMachine**.

MODERNIZING MEDICINE

JSExport

Is a protocol that you implement in your Objective-C/Swift classes to export the instance /class methods and properties to JavaScript.

```
@protocol MMIPointJSExport <JSExport>
    @property (nonatomic, copy) NSString *name;
    @property (nonatomic) CGFloat x;
    @property (nonatomic) CGFloat y;

  - (void)moveTo:(MMIPoint)aPoint;
@end
```

JSExport

So, if we have a **MMIPoint** class that implement **MMIPointJSExport**. We just need to pass the class to the **JSContext**.

```
jsContext[“MMIPoint”] = [MMIPoint class];
```

And then, you can easily use it in JavaScript:

```
var point1 = new MMIPoint();
Var point2 = new MMIPoint();

point1.moveTo(point2);
```

What can we do with this?

1. Well, right now, the best example you can find out there is React Native. This includes Android Devices.
2. BaseCamp iOS app is a WKWebView with bridging!.
3. You can expose some functionalities to be adopted in Javascript so you can make the base and let your users extend the app as they want. (Plugin System).

https://m.signalvnoise.com/basecamp-3-for-ios-hybrid-architecture-afc071589c25

http://nshipster.com/javascriptcore

Thanks