JavaScriptCore

Nate Cook撰写、 April Peng翻译、 发布于2015年1月19日

这个星期流行编程语言的最新排名结果是，Swift 迅速从第 68 位跃升到 22 位，而 Objective-C 仍然稳固的占据在第 10 位。但是，说到允许在 iOS 上运行的其他语言 上，这两个都被甩的很远：当前的冠军是 JavaScript。

OS X Mavericks 和 iOS 7 引入了 JavaScriptCore 库，它把 WebKit 的 JavaScript 引擎用 Objective-C 封装，提供了简单，快速以及安全的方式接入世界上最流行的语言。不管你爱它还是恨它，JavaScript 的普遍存在使得程序员、工具以及融合到 OS X 和 iOS 里这样超快的虚拟机中资源的使用都大幅增长。

这样一来，先抛开动态和类型安全的痛苦辩论，让我带你一起来做一个 *JavaScriptCore 的观光*。

`JSContext` / `JSValue`

`JSContext` 是运行 JavaScript 代码的环境。一个 `JSContext` 是一个全局环境的实例，如果你写过一个在浏览器内运行的 JavaScript，`JSContext` 类似于 `window`。创建一个 `JSContext` 后，可以很容易地运行 JavaScript 代码来创建变量，做计算，甚至定义方法：

Objective-C Swift
Objective-C Swift

Objective-C Swift

```
let context = JSContext()
context.evaluateScript("var num = 5 + 5")
context.evaluateScript("var names = ['Grace', 'Ada', 'Margaret']")
context.evaluateScript("var triple = function(value) { return value * 3 }")
let tripleNum: JSValue = context.evaluateScript("triple(num)")
```

代码的最后一行，任何*出自* `JSContext` 的值都被包裹在一个 `JSValue` 对象中。像 JavaScript 这样的动态语言需要一个动态类型，所以 `JSValue` 包装了每一个可能的 JavaScript 值：字符串和数字；数组、对象和方法；甚至错误和特殊的 JavaScript 值诸如 `null` 和 `undefined`。

`JSValue` 包括一系列方法用于访问其可能的值以保证有正确的 Foundation 类型，包括：

JavaScript Type	`JSValue` method	Objective-C Type	Swift Type

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`	`NSDate!`
Array	`toArray`	`NSArray`	`[AnyObject]!`
Object	`toDictionary`	`NSDictionary`	`[NSObject : AnyObject]!`
Object	`toObject` `toObjectOfClass:`	*custom type*	*custom type*

从上面的例子中得到 `tripleNum` 的值，只需使用适当的方法：

Objective-C Swift

Objective-C Swift

Objective-C Swift

```
println("Tripled: \(tripleNum.toInt32())")
// Tripled: 30
```

下标值

对 `JSContext` 和 `JSValue` 实例使用下标的方式我们可以很容易地访问我们之前创建的 `context` 的任何值。`JSContext` 需要一个字符串下标，而 `JSValue` 允许使用字符串或整数标来得到里面的对象和数组：

Objective-C Swift

Objective-C Swift

Objective-C Swift

```
let names = context.objectForKeyedSubscript("names")
let initialName = names.objectAtIndexedSubscript(0)
println("The first name: \(initialName.toString())")
// The first name: Grace
```

> Swift 展示了它的青涩，在这里，Objective-C 代码可以利用下标表示法，Swift 目前只公开原始方法来让下标成为可能：`objectAtKeyedSubscript()` 和 `objectAtIndexedSubscript()`。

调用方法

`JSValue` 包装了一个 JavaScript 函数，我们可以从 Objective-C / Swift 代码中使用 Foundation 类型作为参数来直接调用该函数。再次，JavaScriptCore 很轻松的处理了这个桥接：

Objective-C Swift

Objective-C Swift

Objective-C Swift

```
let tripleFunction = context.objectForKeyedSubscript("triple")
let result = tripleFunction.callWithArguments([5])
println("Five tripled: \(result.toInt32())")
```

错误处理

`JSContext` 还有另外一个有用的招数：通过设置上下文的 `exceptionHandler` 属性，你可以观察和记录语法，类型以及运行时错误。 `exceptionHandler` 是一个接收一个 `JSContext` 引用和异常本身的回调处理：

Objective-C Swift

Objective-C Swift

```
context.exceptionHandler = { context, exception in
    println("JS Error: \(exception)")
}

context.evaluateScript("function multiply(value1, value2) { return value1 * value2 ")
// JS Error: SyntaxError: Unexpected end of script
```

JavaScript 调用

现在我们知道了如何从 JavaScript 环境中提取值以及如何调用其中定义的函数。那么反向呢？我们怎样才能从 JavaScript 访问我们在 Objective-C 或 Swift 定义的对象和方法?

让 `JSContext` 访问我们的本地客户端代码的方式主要有两种：block 和 `JSExport` 协议。

Blocks

当一个 Objective-C block 被赋给 `JSContext` 里的一个标识符，JavaScriptCore 会自动的把 block 封装在

JavaScript 函数里。这使得在 JavaScript 中可以简单的使用 Foundation 和 Cocoa 类，所有的桥接都为你做好了。见证了 `CFStringTransform` 的强大威力，现在让我们来看看 JavaScript：

Objective-C Swift
Objective-C Swift

Objective-C Swift

```
let simplifyString: @objc_block String -> String = { input in
    var mutableString = NSMutableString(string: input) as CFMutableStringRef
    CFStringTransform(mutableString, nil, kCFStringTransformToLatin, Boolean(0))
    CFStringTransform(mutableString, nil, kCFStringTransformStripCombiningMarks, Boolean(0))
    return mutableString
}
context.setObject(unsafeBitCast(simplifyString, AnyObject.self), forKeyedSubscript: "simplifyStri

println(context.evaluateScript("simplifyString('안녕하세요!')"))
// annyeonghasaeyo!
```

> 在这儿，Swfit 还有一个坑，请注意，这仅适用于 *Objective-C 的 block*，而不是 Swift 的闭包。要在 `JSContext` 中使用 Swift 闭包，它需要（a）与 `@ objc_block` 属性一起声明，以及（b）使用 Swift 那个令人恐惧的 `unsafeBitCast()` 函数转换为 `AnyObject` 。

内存管理

由于 block 可以保有变量引用，而且 `JSContext` 也强引用它所有的变量，为了避免强引用循环需要特别小心。避免保有你的 `JSContext` 或一个 block 里的任何 `JSValue`。相反，使用 `[JSContext currentContext]` 得到当前上下文，并把你需要的任何值用参数传递。

`JSExport` 协议

另一种在 JavaScript 代码中使用我们的自定义对象的方法是添加 `JSExport` 协议。无论我们在 `JSExport` 里声明的属性，实例方法还是类方法，继承的协议都会*自动的*提供给任何 JavaScript 代码。我们将在下一节看到。

JavaScriptCore 实战

让我们做一个使用了所有这些不同的技术的示例 - 我们将定义一个 `Person` 模型符合 `JSExport` 子协议 `PersonJSExports` 的例子，然后使用 JavaScript 从 JSON 文件中创建并填充实例。都有一个完整的 JVM 在那儿了，谁还需要 `NSJSONSerialization`?

1) `PersonJSExports` 和 `Person`

我们的 `Person` 类实现了 `PersonJSExports` 协议，该协议规定哪些属性在 JavaScript 中可用。

> 由于 JavaScriptCore 没有初始化，所以 `create...` 类方法是必要的，我们不能像原生的 JavaScript 类型那样简单地用 `var person = new Person()`。

Objective-C Swift
Objective-C Swift

Objective-C Swift

```swift
// Custom protocol must be declared with `@objc`
@objc protocol PersonJSExports : JSExport {
    var firstName: String { get set }
    var lastName: String { get set }
    var birthYear: NSNumber? { get set }

    func getFullName() -> String

    /// create and return a new Person instance with `firstName` and `lastName`
    class func createWithFirstName(firstName: String, lastName: String) -> Person
}

// Custom class must inherit from `NSObject`
@objc class Person : NSObject, PersonJSExports {
    // properties must be declared as `dynamic`
    dynamic var firstName: String
    dynamic var lastName: String
    dynamic var birthYear: NSNumber?

    init(firstName: String, lastName: String) {
        self.firstName = firstName
        self.lastName = lastName
    }

    class func createWithFirstName(firstName: String, lastName: String) -> Person {
        return Person(firstName: firstName, lastName: lastName)
    }

    func getFullName() -> String {
        return "\(firstName) \(lastName)"
    }
}
```

2) `JSContext` 配置

之前，我们可以用我们已经创建的 `Person` 类，我们需要将其导出到 JavaScript 环境。我们也将借此导入 Mustache JS library，我们将应用模板到我们的 `Person` 对象。

Objective-C Swift
Objective-C Swift

Objective-C Swift

```swift
// export Person class
context.setObject(Person.self, forKeyedSubscript: "Person")
```

```
// load Mustache.js
if let mustacheJSString = String(contentsOfFile:..., encoding:NSUTF8StringEncoding, error:nil) {
    context.evaluateScript(mustacheJSString)
}
```

3) JavaScript 数据和进程

下面就来看看我们简单的 JSON 例子，这段代码将创建新的 `Person` 实例。

> 注意：JavaScriptCore 转换的 Objective-C / Swift 方法名是 JavaScript 兼容的。由于 JavaScript 没有参数 名称，任何外部参数名称都会被转换为驼峰形式并且附加到函数名后。在这个例子中，Objective-C 的方法 `createWithFirstName:lastName:` 变成了在 JavaScript中的 `createWithFirstNameLastName()`。

JSON JavaScript
JSON JavaScript

JSON JavaScript

```
var loadPeopleFromJSON = function(jsonString) {
    var data = JSON.parse(jsonString);
    var people = [];
    for (i = 0; i < data.length; i++) {
        var person = Person.createWithFirstNameLastName(data[i].first, data[i].last);
        person.birthYear = data[i].year;

        people.push(person);
    }
    return people;
}
```

4) 加到一起

剩下的就是加载 JSON 数据，调用 `JSContext` 将数据解析成 `Person` 对象的数组，并用 Mustache 模板呈现每个 `Person`：

Objective-C Swift
Objective-C Swift

Objective-C Swift

```
// get JSON string
if let peopleJSON = NSString(contentsOfFile:..., encoding: NSUTF8StringEncoding, error: nil) {

    // get load function
    let load = context.objectForKeyedSubscript("loadPeopleFromJSON")
    // call with JSON and convert to an array of `Person`
    if let people = load.callWithArguments([peopleJSON]).toArray() as? [Person] {
```

```
        // get rendering function and create template
        let mustacheRender = context.objectForKeyedSubscript("Mustache").objectForKeyedSubscript(
        let template = "{{getFullName}}, born {{birthYear}}"

        // loop through people and render Person object as string
        for person in people {
            println(mustacheRender.callWithArguments([template, person]))
        }
    }
}

// Output:
// Grace Hopper, born 1906
// Ada Lovelace, born 1815
// Margaret Hamilton, born 1936
```

在你的应用程序如何使用 JavaScript？JavaScript 代码段可能是附带应用一起发布的基本的用户定义的插件。如果你的产品开始在 web 上发布，你可能将现有的代码稍加改动即可适应需求。或者，如果*你*开始成为一个 web 程序员，你可能会享受追溯脚本根源的机会。无论是哪种情况，JavaScriptCore 都是精心打造和强大的，不容忽视。

作者

Nate Cook

Nate Cook (@nnnnnnnn) is an independent web and application developer who writes frequently about topics in Swift, and the creator of SwiftDoc.org.

翻译者

April Peng

做 iOS / Mac / Web 开发的大白羊妹子~

下一篇文章

Swift & the Objective-C Runtime

即使一行 Objective-C 代码也不写，每一个 Swift app 都会在 Objective-C runtime 中运行，开启动态任务分发和运行时对象关联的世界。更确切地说，可能在仅使用 Swift 库的时候只运行 Swift runtime。但 Objective-C runtime 与我们共处了如此长的时间，我们也应该将其发挥到极致。

本周的 NShipster 我们将以 Swift 视角来观察这两个运行时中关于关联对象和方法交叉的技术。

相关文章

- CFBag
- UISplitViewController
- NSIndexSet
- NSNotification & NSNotificationCenter

本站文章由 Croath Liu 、 、 Delisa Mason 、 Jack Flintermann 、 Mattt Thompson 、 、 Mike Lazer-Walker 、 Natasha Murashev 和 Nate Cook 撰写、 Andrew Yang 、 April Peng 、 Bob Liu 、 Candyan 、 Chester Liu 、 Croath Liu 、 Daniel Hu 、 David Liu 、 GWesley 、 Henry Lee 、 JJ Mao 、 Lin Xiangyu 、 Ricky Tan 、 Sheldon Huang 、 Tiny Tian 、 Tony Li 、 Yifan Xiao 、 Yu Jin 和 Zihan Xu 翻译。