

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

原 荐 让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCore框架详解

琿少 发表于 1年前 阅读 917 收藏 27 点赞 2 评论 1

收藏

移动开发云端新模式探索实践 >>> HOT

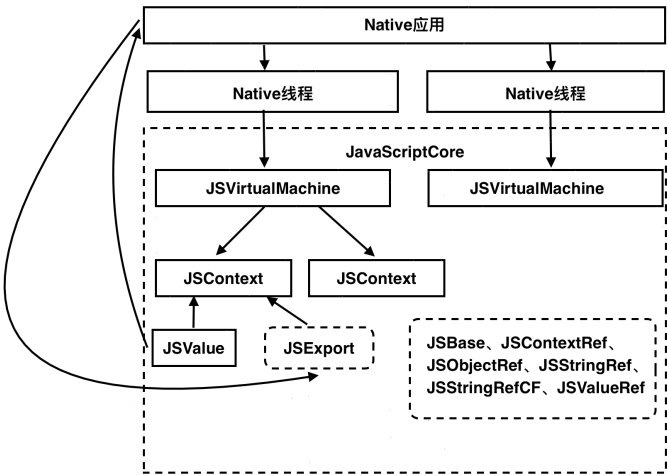
摘要: JavaScriptCore框架提供了在App内部运行JavaScript脚本的能力，支持Objective-C，Swift及其他基于C的应用程序。本篇博客我们将使用Objective-C作为测试环境语言。

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCore框架详解

说到JavaScript脚本，iOS开发者都会想到一个名叫JavaScriptCore的框架。这个框架的确十分强大，其中封装了一套JavaScript运行环境以及Native与JS数据类型之间的转换桥梁。本篇博客主要讨论如何使用此框架来在iOS应用中运行JavaScript脚本。

一、JavaScriptCore框架结构

在学习一个框架时，首先应该先了解整个框架的结构，拿iOS开发来举例，对于一个陌生的框架，第一步需要先搞清楚这里面都包含哪些类，个各类之间是怎样的关系，这个框架和其他的框架间有无联系以及怎样产生的联系。将些问题搞清楚，有了大体上的认识后，我们再来学习其中的每个类即其他细节的应用将非常容易。我们先来看一张JavaScriptCore框架的结构图：



这张图是我手工画的，不是那么美观并且没有文字的解释，但是我觉得它能非常直观的表达JavaScriptCore中包含的类之间的关系。下面我来向你解释这张图究竟表达了什么意思，首先原生的iOS应用是支持多线程执行任务的，我们知道JavaScript是单线程，但这并不代表我们不能在Native中异步执行不同的JavaScript代码。

1.JSVirtualMachine——JavaScript的虚拟机

JavaScriptCore中提供了一个名为JSVirtualMachine的类，顾名思义，这个类可以理解为一个JS虚拟机。在Native中，只要你愿意，你可以创建任意多个JSVirtualMachine对象，各个JSViretualMachine对象间是相互独立的，他们之间不能共享数据也不能传递数据，如果你把他们放在不同的Native线程，他们就可以并行的执行无关的JS任务。

2.JSContext——JavaScript运行环境

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

执行JS代码，在Native和JS间进行数据的传递。

3.JSValue——JavaScript值对象

JavaScript和Objective-C虽然都是面向对象语言，但其实现机制完全不同，OC是基于类的，JS是基于原型的，并且他们的数据类型间也存在很大的差异。因此若要在Native和JS间无障碍的进行数据的传递，就需要一个中间对象做桥接，这个对象就是JSValue。

4.JSExport

JSExport是一个协议，Native中遵守此解析的类可以将方法和属性转换为JS的接口供JS调用。

5.一些用于C语言的结构

你一定注意到了，上图的右下角还有一块被虚线包围的区域，其中的"类"都是C语言风格，JavaScriptCore框架是支持在Objective-C、Swift和C三种语言中使用的。

二、在Native中运行JavaScript脚本代码

我们先来编写一个最简单的例子，使用OC代码来执行一段JS脚本。首先新建一个文件，将其后缀设置为.js，我这里将它命令为main.js，在其中编写如下代码：

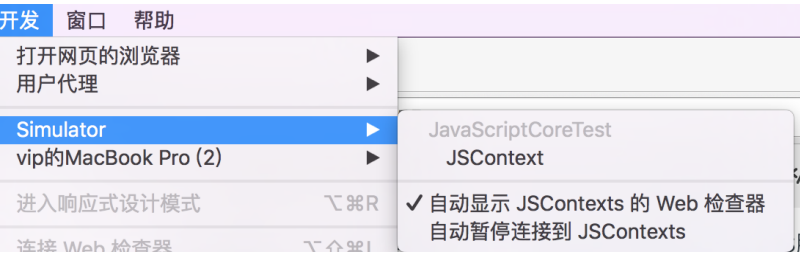
```
(function(){
    console.log("Hello Native");
})();
```

上面是一个自执行的函数，其中打印了“Hello Native”字符串。在Native中编写如下代码：

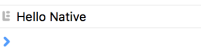
```
- (void)viewDidLoad {
    [super viewDidLoad];
    NSString * path = [[NSBundle mainBundle] pathForResource:@"main" ofType:@"js"];
    NSData * jsData = [[NSData alloc] initWithContentsOfFile:path];
    NSString * jsCode = [[NSString alloc] initWithData:jsData encoding:NSUTF8StringEncoding];
    self.jsContext = [[JSContext alloc] init];
    [self.jsContext evaluateScript:jsCode];
}
```

需要注意，其实这里我将创建的JSContext对象作为当前视图控制器的属性，这样做的目的仅仅是为了方便调试，不过不对此context对象进行引用，当viewDidLoad函数执行完成后，JS运行环境也将被销毁，我们就无法在Safari中直观的看到JS代码的执行结果了。

运行工程，记得要打开Safari浏览器的自动显示JSContent检查器，如下图：



当iOS模拟器跑起来后，Safari会自动弹出开发者工具，在控制台里面可以看到来自JavaScript的真挚问候：



刚才我们只是简单的通过原生调用了一段JS代码，但是如果Native在调JS方法时无法传参那也太low了，我们可以直接将要传递的参数格式化到字符串中，修改main.js文件如下：

```
function put(name){
    console.log("Hello "+name);
};
put(%@);
```

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"main" ofType:@"js"];
NSData * jsData = [[NSData alloc] initWithContentsOfFile:path];
NSString * jsCode = [[NSString alloc] initWithData:jsData encoding:NSUTF8StringEncoding];
NSString * finiString = [NSString stringWithFormat:jsCode,name];
[self.jsContext evaluateScript:finiString];
}
```

在viewDidLoad中进行调用，如下：

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.jsContext = [[JSContext alloc] init];
    [self runJS_Hello:@"阿凡达"];
}
```

运行再看Safari控制台的结果，编程了Hello 阿凡达~：

Hello 阿凡达



其实evaluateScript函数执行后会将JS代码的执行结果进行返回，是JSValue类型的对象，后面会再介绍。

三、在JavaScript中调用Native方法

有来无往非君子，同样也可以在原生中编写方法让JS来调用，示例如下：

```
- (void)viewDidLoad {
    [super viewDidLoad];
    void(^block)() = ^(){
        NSLog(@"Hello JavaScript");
    };
    self.jsContext = [[JSContext alloc] init];
    [self.jsContext setObject:block forKeyedSubscript:@"oc_hello"];
}
```

上面setObject:forKeyedSubscript:方法用来向JSContext环境的全局对象中添加属性，这里添加了一个函数属性，取名为oc_hello。这里JavaScriptCore会自动帮我们把一些数据类型进行转换，会将OC的函数转换为JS的函数，运行工程，在Safari的控制台中调用oc_hello函数，可以看到在Xcode控制台输出了对JavaScript的真挚问候，如下：

2017-03-09 16:54:53.907 JavaScriptCoreTest[1894:240992] Hello JavaScript

同样，如果声明的block是带参数的，JS在调用此OC方法时也需要传入参数，如果block有返回值，则在JS中也能获取到返回值，例如：

```
BOOL (^block)(NSString *) = ^(NSString *name){
    NSLog(@"%@", [NSString stringWithFormat:@"Hello %@",name]);
    return YES;
};
self.jsContext = [[JSContext alloc] init];
[self.jsContext setObject:block forKeyedSubscript:@"oc_hello"];
```

四、深入JSContext类

看到这，你已经学会最基础的OC与JS互相问好(交互)。下面我们再来深入看下JSContext中的属性和方法。

创建JSContext对象有如下两种方式：

```
//创建一个新的JS运行环境
- (instancetype) init;
//创建一个新的JS运行环境 并关联到某个虚拟机对象上
- (instancetype) initWithVirtualMachine:(JSVirtualMachine *)virtualMachine;
```

执行JS代码有如下两个方法：

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```
- (JSValue *)evaluateScript:(NSString *)script withSourceURL:(NSURL *)sourceURL NS_AVAILABLE(10_10, {
```

下面的属性和方法可以获取到JS运行环境中的一些信息：

```
//当前的JS运行环境 当JS调用OC方法时，在OC方法中可以用此方法获取到JS运行环境
+ (JSContext *)currentContext;
//获取当前执行的JS函数，当JS调用OC方法时，在OC方法中可以用此方法获取到执行的函数
+ (JSValue *)currentCallee;
//获取当前执行的JS函数中的this指向的对象
+ (JSValue *)currentThis;
//获取当前执行函数的参数列表，当JS调用OC方法时，在OC方法中可以用此方法获取到执行的函数的参数列表
+ (NSArray *)currentArguments;
//获取当前JS运行环境的全局对象
@property (readonly, strong) JSValue *globalObject;
//当运行的JavaScript代码抛出了未捕获的异常时，这个属性会被赋值为抛出的异常
@property (strong) JSValue *exception;
//设置为一个异常捕获的block，如果异常被此block捕获，exception属性就不再被赋值了
@property (copy) void(^exceptionHandler)(JSContext *context, JSValue *exception);
//当前运行环境所关联的虚拟机
@property (readonly, strong) JSVirtualMachine *virtualMachine;
//当前运行环境名称
@property (copy) NSString *name;
//获取当前JS运行环境全局对象上的某个属性
- (JSValue *)objectForKeyedSubscript:(id)key;
//设置当前JS运行环境全局对象上的属性
- (void)setObject:(id)object forKeyedSubscript:(NSObject <NSCopying> *)key;
//将C语言环境的JS运行环境转换为OC环境的JS运行环境
+ (JSContext *)contextWithJSGlobalContextRef:(JSGlobalContextRef)jsGlobalContextRef;
//C语言环境的JS运行上下文
@property (readonly) JSGlobalContextRef jsGlobalContextRef;
```

五、深入JSValue类

JSValue是JavaScript与Objective-C之间的数据桥梁。在Objective-C中调用JS脚本或者JS调用OC方法都可以使用JSValue来传输数据。其中属性和方法示例如下：

```
//所对应的JS运行环境
@property (readonly, strong) JSContext *context;
//在指定的JS运行环境中创建一个JSValue对象
+ (JSValue *)valueWithObject:(id)value inContext:(JSContext *)context;
//创建布尔值
+ (JSValue *)valueWithBool:(BOOL)value inContext:(JSContext *)context;
//创建浮点值
+ (JSValue *)valueWithDouble:(double)value inContext:(JSContext *)context;
//创建32位整型值
+ (JSValue *)valueWithInt32:(int32_t)value inContext:(JSContext *)context;
//创建32位无符号整形值
+ (JSValue *)valueWithUInt32:(uint32_t)value inContext:(JSContext *)context;
//创建空的JS对象
+ (JSValue *)valueWithNewObjectInContext:(JSContext *)context;
//创建空的JS数组
+ (JSValue *)valueWithNewArrayInContext:(JSContext *)context;
//创建JS正则对象
+ (JSValue *)valueWithNewRegularExpressionFromPattern:(NSString *)pattern flags:(NSString *)flags inContext:(JSContext *)context;
//创建JS错误信息
+ (JSValue *)valueWithNewErrorFromMessage:(NSString *)message inContext:(JSContext *)context;
//创建JS null值
+ (JSValue *)valueWithNullInContext:(JSContext *)context;
//创建JS undefined值
+ (JSValue *)valueWithUndefinedInContext:(JSContext *)context;
```

JavaScript中的数据类型和Objective-C的数据类型还是有着很大的差异，其中对应关系如下：

Objective-C	JavaScript
nil	undefined
NSNull	null
NSString	string
NSNumber	number boolean
NSDictionary	Object

让你的iOS应用程序支持运行JavaScript脚本： JavaScriptCor...

琿少 发表于1年前

Block	Function
id	Object
Class	Object

下面这些方法可以将JSValue值转换为Objective-C中的数据类型：

```
//将JSValue转换为OC对象
- (id)toObject;
//将JSValue转换成特定OC类的对象
- (id)toObjectOfClass:(Class)expectedClass;
//将JSValue转换成布尔值
- (BOOL)toBool;
//将JSValue转换成浮点值
- (double)toDouble;
//将JSValue转换成32位整型值
- (int32_t)toInt32;
//将JSValue转换成32位无符号整型值
- (uint32_t)toUInt32;
//将JSValue转换成NSNumber值
- (NSNumber *)toNumber;
//将JSValue转换成NSString值
- (NSString *)toString;
//将JSValue转换成NSDate值
- (NSDate *)toDate;
//将JSValue转换成NSArray值
- (NSArray *)toArray;
//将JSValue转换成NSDictionary值
- (NSDictionary *)toDictionary;
//获取JSValue对象中某个属性的值
- (JSValue *)valueForProperty:(NSString *)property;
//设置JSValue对象中某个属性的值
- (void)setValue:(id)value forProperty:(NSString *)property;
//删除JSValue对象中的某个属性
- (BOOL)deleteProperty:(NSString *)property;
//判断JSValue对象中是否包含某个属性
- (BOOL)hasProperty:(NSString *)property;
//定义JSValue中的某个属性 这个方法和JavaScript中Object构造函数的defineProperty方法一致
/*
第2个参数设置此属性的描述信息 可以设置的键值如下：
NSString * const JSPROPERTY_DESCRIPTOR_WRITABLE_KEY; //设置布尔值 是否可写
NSString * const JSPROPERTY_DESCRIPTOR_ENUMERABLE_KEY; //设置布尔值 是否可枚举
NSString * const JSPROPERTY_DESCRIPTOR_CONFIGURABLE_KEY; //设置布尔值 是否可配置
NSString * const JSPROPERTY_DESCRIPTOR_VALUE_KEY; //设置此属性的值
NSString * const JSPROPERTY_DESCRIPTOR_GET_KEY; //设置此属性的get方法
NSString * const JSPROPERTY_DESCRIPTOR_SET_KEY; //设置此属性的set方法
以上set、get方法的键和value、可写性的键不能同时存在，其语法是JavaScript保持一致
*/
- (void)defineProperty:(NSString *)property descriptor:(id)descriptor;
//获取JS数组对象某个下标的值
- (JSValue *)valueAtIndex:(NSUInteger)index;
//设置JS数组对象某个下标的值
- (void)setValue:(id)value atIndex:(NSUInteger)index;
//判断此对象是否为undefined
@property (readonly) BOOL isUndefined;
//判断此对象是否为null
@property (readonly) BOOL isNull;
//判断此对象是否为布尔值
@property (readonly) BOOL isBoolean;
//判断此对象是否为数值
@property (readonly) BOOL isNumber;
//判断此对象是否为字符串
@property (readonly) BOOL isString;
//判断此对象是否为object对象
@property (readonly) BOOL isObject;
//判断此对象是否为数组
@property (readonly) BOOL isArray;
//判断此对象是否为日期对象
@property (readonly) BOOL isDate;
//比较两个JSValue是否全相等 对应JavaScript中的===
- (BOOL)isEqualToObject:(id)value;
//比较两个JSValue对象的值是否相等 对应JavaScript中的==
- (BOOL)isEqualWithTypeCoercionToObject:(id)value;
//判断某个对象是否在当前对象的原型链上
- (BOOL)isInstanceOf:(id)value;
//如果JSValue是Function对象 可以调用此方法 和JavaScript中的call方法一致
- (JSValue *)callWithArguments:(NSArray *)arguments;
//如果JSValue是一个构造方法对象 可以调用此方法 和JavaScript中使用new关键字一致
- (JSValue *)constructWithArguments:(NSArray *)arguments;
//用此对象进行函数的调用 当前对象会被绑定到this中
```

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```
+ (JSValue *)valueWithRange:(NSRange)range inContext:(JSContext *)context;
//将CGRect转换为JSValue对象
+ (JSValue *)valueWithRect:(CGRect)rect inContext:(JSContext *)context;
//将CGSize转换为JSValue对象
+ (JSValue *)valueWithSize:(CGSize)size inContext:(JSContext *)context;
//转换成CGPoint数据
- (CGPoint)toPoint;
//转换成NSRange数据
- (NSRange)toRange;
//转换成CGRect数据
- (CGRect)toRect;
//转换为CGSize数据
- (CGSize)toSize;
//将C风格的JSValueRef对象转换为JSValue对象
+ (JSValue *)valueWithJSValueRef:(JSValueRef)value inContext:(JSContext *)context;
```

其实在JavaScriptCore框架中还有一个JSManagerValue类，这个的主要作用是管理内存。虽然我们在编写Objective-C代码时有强大的自动引用技术(ARC技术)，我们一般无需关心对象的内存问题，在编写JavaScript代码时也有强大的垃圾回收机制(这种机制下甚至连循环引用都不是问题)，但是在OC和JS混合开发时，就容易出现问题了，比如一个JS垃圾回收机制释放掉的对象OC中却还在用，反过来也是一样。JSManagerValue对JSValue进行了一层包装，它可以保证在适合时候使用这个对象时对象都不会被释放，其中方法如下：

```
//创建JSValue对象的包装JSManagerValue
+ (JSManagedValue *)managedValueWithValue:(JSValue *)value;
+ (JSManagedValue *)managedValueWithValue:(JSValue *)value andOwner:(id)owner;
- (instancetype)initWithValue:(JSValue *)value;
//获取所包装的JSValue对象
@property (readonly, strong) JSValue *value;
```

六、Objective-C与JavaScript复杂对象的映射

我们在使用JavaScript调用Objective-C方法的实质是将一个OC函数设置为了JS全局对象的一个属性，当然我们也可以设置非函数的属性或者任意JSValue(或者可以转换为JSValue)的值。例如：

```
self.jsContext = [[JSContext alloc] init];
//向JS全局对象中添加一个获取当前Native设备类型的属性
[self.jsContext setObject:@"iOS" forKeyedSubscript:@"deviceType"];
```

但是如果我们想把OC自定义的一个类的对象设置为JS全局对象的某个属性，JS和OC有着完全不同的对象原理，如果不做任何处理，JS是接收不到OC对象中定义的属性和方法的。这时就需要使用到前面提到的JSEXport协议，需要注意，这个协议不是用来被类遵守的，它里面没有规定任何方法，其是用来被继承定义新的协议的，自定义的协议中约定的方法和属性可以在JS中被获取到，示例如下：

OC中新建一个自定义的类：

```
@protocol MyObjectProtocol <JSEXport>

@property(nonatomic, strong) NSString * name;
@property(nonatomic, strong) NSString * subject;
@property(nonatomic, assign) NSInteger age;
-(void)sayHi;

@end

@interface MyObject : NSObject<MyObjectProtocol>
@property(nonatomic, strong) NSString * name;
@property(nonatomic, strong) NSString * subject;
@property(nonatomic, assign) NSInteger age;
@end
@implementation MyObject
-(void)sayHi{
    NSLog(@"Hello JavaScript");
}
@end
```

添加到JS全局对象中：

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```
object.subject = @"0C";  
[jsContext setObject:object forKeyedSubscript:@"deviceObject"];
```

在JS运行环境中可以完整的到deviceObject对象，如下：

```
> deviceObject  
◀ ▶ MyObject {name: "Jaki", subject: "0C", age: 25} = $1  
> |
```

七、C语言风格的API解释

JavaScriptCore框架中除了包含完整的Objective-C和Swift语言的API外，也提供了对C语言的支持。

与JS运行环境相关的方法如下：

```
//创建一个JSContextRef组  
/*  
JSContextRef相当于JSContext，同一组中的数据可以共享  
*/  
JSContextGroupRef JSContextGroupCreate(void);  
//内存引用  
JSContextGroupRef JSContextGroupRetain(JSContextGroupRef group);  
//内存引用释放  
void JSContextGroupRelease(JSContextGroupRef group);  
//创建一个全局的运行环境  
JSGlobalContextRef JSGlobalContextCreate(JSCClassRef globalObjectClass);  
//同上  
JSGlobalContextRef JSGlobalContextCreateInGroup(JSContextGroupRef group, JSCClassRef globalObjectClass);  
//内存引用  
JSGlobalContextRef JSGlobalContextRetain(JSGlobalContextRef ctx);  
//内存引用释放  
void JSGlobalContextRelease(JSGlobalContextRef ctx);  
//获取全局对象  
JSObjectRef JSContextGetGlobalObject(JSContextRef ctx);  
//获取JSContextRef组  
JSContextGroupRef JSContextGetGroup(JSContextRef ctx);  
//获取全局的运行环境  
JSGlobalContextRef JSContextGetGlobalContext(JSContextRef ctx);  
//获取运行环境名  
JSStringRef JSGlobalContextCopyName(JSGlobalContextRef ctx);  
//设置运行环境名  
void JSGlobalContextSetName(JSGlobalContextRef ctx, JSStringRef name);
```

与定义JS对象的相关方法如下：

```
//定义JS类  
/*  
参数JSCClassDefinition是一个结构体 其中可以定义许多回调  
*/  
JSCClassRef JSCClassCreate(const JSCClassDefinition* definition);  
//引用内存  
JSCClassRef JSCClassRetain(JSCClassRef jsClass);  
//释放内存  
void JSCClassRelease(JSCClassRef jsClass);  
//创建一个JS对象  
JSObjectRef JSObjectMake(JSContextRef ctx, JSCClassRef jsClass, void* data);  
//定义JS函数  
JSObjectRef JSObjectMakeFunctionWithCallback(JSContextRef ctx, JSStringRef name, JSObjectCallAsFunctionCallback callback);  
//定义构造函数  
JSObjectRef JSObjectMakeConstructor(JSContextRef ctx, JSCClassRef jsClass, JSObjectCallAsConstructorCallback callback);  
//定义数组  
JSObjectRef JSObjectMakeArray(JSContextRef ctx, size_t argumentCount, const JSValueRef arguments[], JSObjectCallAsFunctionCallback callback);  
//定义日期对象  
JSObjectRef JSObjectMakeDate(JSContextRef ctx, size_t argumentCount, const JSValueRef arguments[], JSObjectCallAsFunctionCallback callback);  
//定义异常对象  
JSObjectRef JSObjectMakeError(JSContextRef ctx, size_t argumentCount, const JSValueRef arguments[], JSObjectCallAsFunctionCallback callback);  
//定义正则对象  
JSObjectRef JSObjectMakeRegExp(JSContextRef ctx, size_t argumentCount, const JSValueRef arguments[], JSObjectCallAsFunctionCallback callback);  
//定义函数对象  
JSObjectRef JSObjectMakeFunction(JSContextRef ctx, JSStringRef name, unsigned parameterCount, const JSValueRef arguments[], JSObjectCallAsFunctionCallback callback);  
//获取对象的属性  
JSValueRef JSObjectGetPrototype(JSContextRef ctx, JSObjectRef object);  
//设置对象的属性  
void JSObjectSetPrototype(JSContextRef ctx, JSObjectRef object, JSValueRef value);  
//检查对象是否包含某个属性  
bool JSObjectHasProperty(JSContextRef ctx, JSObjectRef object, JSStringRef propertyName);
```


让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```
//删除对象属性
bool JSObjectDeleteProperty(JSContextRef ctx, JSObjectRef object, JSStringRef propertyName, JSValueRef value);
//获取数组值
JSValueRef JSObjectGetPropertyAtIndex(JSContextRef ctx, JSObjectRef object, unsigned propertyIndex, JSValueRef* exception);
//设置数组值
void JSObjectSetPropertyAtIndex(JSContextRef ctx, JSObjectRef object, unsigned propertyIndex, JSValueRef value, JSValueRef* exception);
//获取私有数据
void* JSObjectGetPrivate(JSObjectRef object);
//设置私有数据
bool JSObjectSetPrivate(JSObjectRef object, void* data);
//判断是否为函数
bool JSObjectIsFunction(JSContextRef ctx, JSObjectRef object);
//将对象作为函数来调用
JSValueRef JSObjectCallAsFunction(JSContextRef ctx, JSObjectRef object, JSObjectRef thisObject, size_t argumentCount, const JSValueRef* arguments, JSValueRef* exception);
//判断是否为构造函数
bool JSObjectIsConstructor(JSContextRef ctx, JSObjectRef object);
//将对象作为构造函数来调用
JSObjectRef JSObjectCallAsConstructor(JSContextRef ctx, JSObjectRef object, size_t argumentCount, const JSValueRef* arguments, JSValueRef* exception);
//获取所有属性名
JSPROPERTY_NAME_ARRAY_REF JSObjectCopyPropertyNames(JSContextRef ctx, JSObjectRef object);
//进行内存引用
JSPROPERTY_NAME_ARRAY_REF JSPROPERTY_NAME_ARRAY_RETAIN(JSPROPERTY_NAME_ARRAY_REF array);
//内存释放
void JSPROPERTY_NAME_ARRAY_RELEASE(JSPROPERTY_NAME_ARRAY_REF array);
//获取属性个数
size_t JSPROPERTY_NAME_ARRAY_GET_COUNT(JSPROPERTY_NAME_ARRAY_REF array);
//在属性名数组中取值
JSStringRef JSPROPERTY_NAME_ARRAY_GET_NAME_AT_INDEX(JSPROPERTY_NAME_ARRAY_REF array, size_t index);
//添加属性名
void JSPROPERTY_NAME_ACCUMULATOR_ADD_NAME(JSPROPERTY_NAME_ACCUMULATOR_REF accumulator, JSStringRef propertyName, JSValueRef value, JSValueRef* exception);
```

JS数据类型相关定义在JSValueRef中，如下：

```
//获取值的类型
/*
枚举如下：
typedef enum {
    kJSTypeUndefined,
    kJSTypeNull,
    kJSTypeBoolean,
    kJSTypeNumber,
    kJSTypeString,
    kJSTypeObject
} JSType;
*/
JSType JSValueGetType(JSContextRef ctx, JSValueRef value);
//判断是否为undefined类型
bool JSValueIsUndefined(JSContextRef ctx, JSValueRef value);
//判断是否为null类型
bool JSValueIsNull(JSContextRef ctx, JSValueRef value);
//判断是否为布尔类型
bool JSValueIsBoolean(JSContextRef ctx, JSValueRef value);
//判断是否为数值类型
bool JSValueIsNumber(JSContextRef ctx, JSValueRef value);
//判断是否为字符串类型
bool JSValueIsString(JSContextRef ctx, JSValueRef value);
//判断是否为对象类型
bool JSValueIsObject(JSContextRef ctx, JSValueRef value);
//是否是类
bool JSValueIsObjectOfClass(JSContextRef ctx, JSValueRef value, JSCClassRef jsClass);
//是否是数组
bool JSValueIsArray(JSContextRef ctx, JSValueRef value);
//是否是日期
bool JSValueIsDate(JSContextRef ctx, JSValueRef value);
//比较值是否相等
bool JSValueIsEqual(JSContextRef ctx, JSValueRef a, JSValueRef b, JSValueRef* exception);
//比较值是否全等
bool JSValueIsStrictEqual(JSContextRef ctx, JSValueRef a, JSValueRef b);
//是否是某个类的实例
bool JSValueIsInstanceOfConstructor(JSContextRef ctx, JSValueRef value, JSObjectRef constructor, JSValueRef* exception);
//创建undefined值
JSValueRef JSValueMakeUndefined(JSContextRef ctx);
//创建null值
JSValueRef JSValueMakeNull(JSContextRef ctx);
//创建布尔值
JSValueRef JSValueMakeBoolean(JSContextRef ctx, bool boolean);
//创建数值
JSValueRef JSValueMakeNumber(JSContextRef ctx, double number);
//创建字符串值
JSValueRef JSValueMakeString(JSContextRef ctx, JSStringRef string);
//通过JSON创建对象
```


让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```
bool JSValueToBoolean(JSContextRef ctx, JSValueRef value);
//进行数值转换
double JSValueToNumber(JSContextRef ctx, JSValueRef value, JSValueRef* exception);
//字符串赋值
JSStringRef JSValueToStringCopy(JSContextRef ctx, JSValueRef value, JSValueRef* exception);
//值与对象的转换
JSObjectRef JSValueToObject(JSContextRef ctx, JSValueRef value, JSValueRef* exception);
```

在C风格的API中，字符串也被包装成了JSStringRef类型，其中方法如下：

```
//创建js字符串
JSStringRef JSStringCreateWithCharacters(const JSChar* chars, size_t numChars);
JSStringRef JSStringCreateWithUTF8CString(const char* string);
//内存引用于释放
JSStringRef JSStringRetain(JSStringRef string);
void JSStringRelease(JSStringRef string);
//获取字符串长度
size_t JSStringGetLength(JSStringRef string);
//转成UTF8字符串
size_t JSStringGetUTF8CString(JSStringRef string, char* buffer, size_t bufferSize);
//字符串比较
bool JSStringIsEqual(JSStringRef a, JSStringRef b);
bool JSStringIsEqualToUTF8CString(JSStringRef a, const char* b);
```

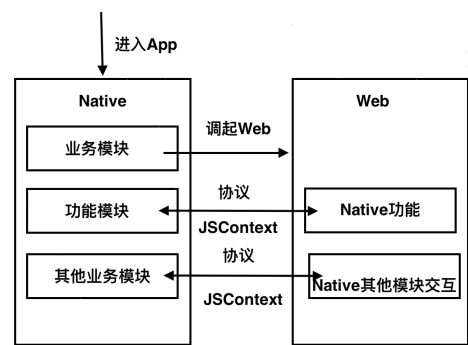
八、Hybird App 构建思路

Hybird App是指混合模式移动应用，即其中既包含原生的结构有内嵌有Web的组件。这种App不仅性能和用户体验可以达到和原生所差无几的程度，更大的优势在于bug修复快，版本迭代无需发版。3月8日苹果给许多开发者发送了一封警告邮件，主要是提示开发者下载脚本动态更改App原本行为的做法将会被提审拒绝。其实这次邮件所提内容和Hybird App并无太大关系(对ReactNative也没有影响)，苹果警告的是网络下发脚本并且使用runtime动态修改Native行为的应用，Hybird App的实质并没有修改原Native的行为，而是将下发的资源进行加载和界面渲染，类似WebView。

关于混合开发，我们有两种模式：

1.Native内嵌WebView，通过JS与OC交互实现业务无缝的衔接。

无论是UIWebView还是WKWebKit，我们都可以在其中拿到当前的JSContext，然是使用前面介绍的方法便可以实现数据互通与交互。这种方式是最简单的混合开发，但其性能和原生相比要差一些。示意图如下：



2.下发JS脚本，使用类似ReactNative的框架进行原生渲染

这是一种效率非常高的混合开发模式，并且ReactNative也本身支持android和iOS公用一套代码。我们也可以使用JavaScriptCore自己实现一套解析逻辑，使用JavaScript来编写Native应用，要完整实现这样一套东西太复杂了，我们也没有能力完成一个如此庞大的工程，但是我们可以做一个小Demo来模拟其原理，这样可以更好的帮助我们理解Hybird App的构建原理。

我们打算实现这样的功能：通过下发JS脚本创建原生的UILabel标签与UIButton控件，首先编写JS代码如下：

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```

        return render();
    })();

//JS标签类
function Label(rect,text,color){
    this.rect = rect;
    this.text = text;
    this.color = color;
    this.typeName = "Label";
}

//JS按钮类
function Button(rect,text,callFunc){
    this.rect = rect;
    this.text = text;
    this.callFunc = callFunc;
    this.typeName = "Button";
}

//JS Rect类
function Rect(x,y,width,height){
    this.x = x;
    this.y = y;
    ...
}

//渲染方法 界面的渲染写在这里面
function render(){
    var rect = new Rect(20,100,280,30);
    var color = new Color(1,0,0,1);
    var label = new Label(rect,"Hello World",color);
    var rect2 = new Rect(20,150,280,30);
    var color2 = new Color(0,1,0,1);
    var label2 = new Label(rect2,"Hello Native",color2);
    var rect3 = new Rect(20,200,280,30);
    var color3 = new Color(0,0,1,1);
    var label3 = new Label(rect3,"Hello JavaScript",color3);
    var rect4 = new Rect(20,240,280,30);
    var button = new Button(rect4,"我是一个按钮",function(){
        var randColor = new Color(Math.random(),Math.random(),Math.random(),1);
        Globle.changeBackgroundColor(randColor);
    });

    //将控件以数组形式返回
    return [label,label2,label3,button];
}

```

创建一个Objective-C类绑定到JS全局对象上，作为OC方法的桥接器：

```

//.h
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import <JavaScriptCore/JavaScriptCore.h>
@protocol GloblePrptocol <JSExport>
-(void)changeBackgroundColor:(JSValue *)value;
@end
@interface Globle : NSObject<GloblePrptocol>
@property(nonatomic,weak)UIViewController * ownerController;
@end
//.m
#import "Globle.h"

@implementation Globle
-(void)changeBackgroundColor:(JSValue *)value{
    self.ownerController.view.backgroundColor = [UIColor colorWithRed:value[@"r"].toDouble green:val
}
@end

```

在ViewController中实现一个界面渲染的render解释方法，并建立按钮的方法转换，如下：

```

//
// ViewController.m
// JavaScriptCoreTest
//
// Created by vip on 17/3/6.

```

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前

```

#import <JavaScriptCore/JavaScriptCore.h>
#import "Globle.h"
@interface ViewController ()

@property(nonatomic,strong)JSContext * jsContext;
@property(nonatomic,strong)NSMutableArray * actionArray;
@property(nonatomic,strong)Globle * globle;
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    //创建JS运行环境
    self.jsContext = [JSContext new];
    //绑定桥接器
    self.globle = [Globle new];
    self.globle.ownerController = self;
    self.jsContext[@"Globle"] = self.globle;
    self.actionArray = [NSMutableArray array];
    [self render];
}

//界面渲染解释器
-(void)render{
    NSString * path = [[NSBundle mainBundle] pathForResource:@"main" ofType:@"js"];
    NSData * jsData = [[NSData alloc] initWithContentsOfFile:path];
    NSString * jsCode = [[NSString alloc] initWithData:jsData encoding:NSUTF8StringEncoding];
    JSValue * jsVlaue = [self.jsContext evaluateScript:jsCode];
    for (int i=0; i<jsVlaue.toArray.count; i++) {
        JSValue * subValue = [jsVlaue objectAtIndexedSubscript:i];
        if ([[subValue objectForKeyedSubscript:@"typeName"].toString isEqualToString:@"Label"]) {
            UILabel * label = [UILabel new];
            label.frame = CGRectMake(subValue[@"rect"][@"x"].toDouble, subValue[@"rect"][@"y"].toDouble,
            label.text = subValue[@"text"].toString;
            label.textColor = [UIColor colorWithRed:subValue[@"color"][@"r"].toDouble green:subValue
            [self.view addSubview:label];
        }else if ([[subValue objectForKeyedSubscript:@"typeName"].toString isEqualToString:@"Button"])
            UIButton * button = [UIButton buttonWithType:UIButtonTypeSystem];
            button.frame = CGRectMake(subValue[@"rect"][@"x"].toDouble, subValue[@"rect"][@"y"].toDouble,
            [button setTitle:subValue[@"text"].toString forState:UIControlStateNormal];
            button.tag = self.actionArray.count;
            [button addTarget:self action:@selector(buttonAction:) forControlEvents:UIControlEventTouchUpInside];
            [self.actionArray addObject:subValue[@"callFunc"]];
            [self.view addSubview:button];
        }
    }
}

//按钮转换方法
-(void)buttonAction:(UIButton *)btn{
    JSValue * action = self.actionArray[btn.tag];
    //执行JS方法
    [action callWithArguments:nil];
}

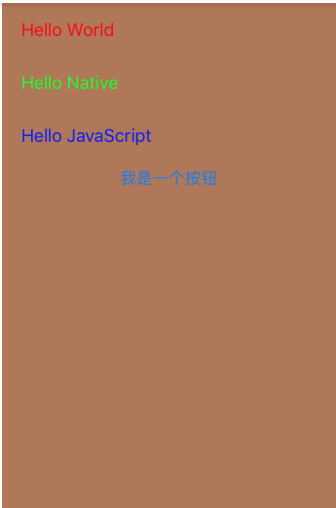
@end

```

运行工程，效果如下图所示，点击按钮即可实现简单的界面颜色切换：

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前



上面的示例工程我只实现了UILabel类与UIButton类的JS-OC转换，如果将原生控件和JS对象再进行一层绑定，并且实现大部分JS类与原生类和他们内部的属性，则我们就开发了一套Hybird App开发框架，但并没有这个必要，如果你对更多兴趣，可以深入学习下ReactNative。

文中的示例Demo我放在了Github上，地址如下：<https://github.com/ZYHshao/Demo-Hybrid>。

前端学习新人，有志同道合的朋友，欢迎交流与指导，QQ群:541458536

© 著作权归作者所有

分类：iOS之逻辑初窥 字数：6162

标签： iOS Hybrid原理 JavaScriptCore OC与JS交互 iOS混合开发

小程序开发套餐11元/月起

云端架构，简单易开发;丰富的小程序模板,满足电商/餐饮/O2O等多
cloud.tencent.com

打赏

点赞

收藏

分享

举报



琿少


iOS工程师 上海

+ 关注 粉丝 831 | 博文 371 | 码字总数 434111




相关博客

iOS7新JavaScriptCore框架入门介绍

 北方人在上海


46 0

JavaScriptCore框架在iOS7中的对象交互和管理

 北方人在上海

30 0

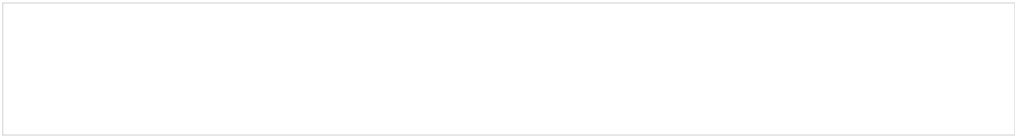
iOS UIWebView获取当前JSContext

 SoulJa

145 0

让你的iOS应用程序支持运行JavaScript脚本：JavaScriptCor...

琿少 发表于1年前



Ctrl+Enter 发表评论



iamOkay

1楼 2017/03/10 21:46

其实我想说，这种技术一点都不实用，我们公司早前使用了这种框架，最后全部改用了UIWebView和WKWebView

社区

- 开源项目
- 技术问答
- 动弹
- 博客

- 开源资讯
- 技术翻译
- 专题
- 招聘

众包

- 项目大厅
- 软件与服务
- 接活赚钱

码云

- Git代码托管
- Team
- PaaS
- 在线工具

活动

- 线下活动
- 发起活动
- 源创会

关注微信公众号



下载手机客户端

