**JSPatch 使用说明**

**框架功能：iOS线上APP 紧急bug热修复第三方框架**

**最近时间：2016 – 09 – 02**

**当前版本：1.0.0**

**修改记录**

|  |  |  |
| --- | --- | --- |
| 版本：1.0.0 | 编辑：聂小波 | 首次创建时间：2016/9/2 |
|  |  |  |
|  |  |  |

**目录**

一、什么是JSPatch 2

二、仓库设置方法（后台文件管理） 3

三、ios前端配置 3

四、更新频率，及App更新方法代码名称 4

五、请求数据（前后台接口设置） 5

六、前端下载和使用 6

1）本地使用 6

2）下载使用 7

七、JSPatch语法 7

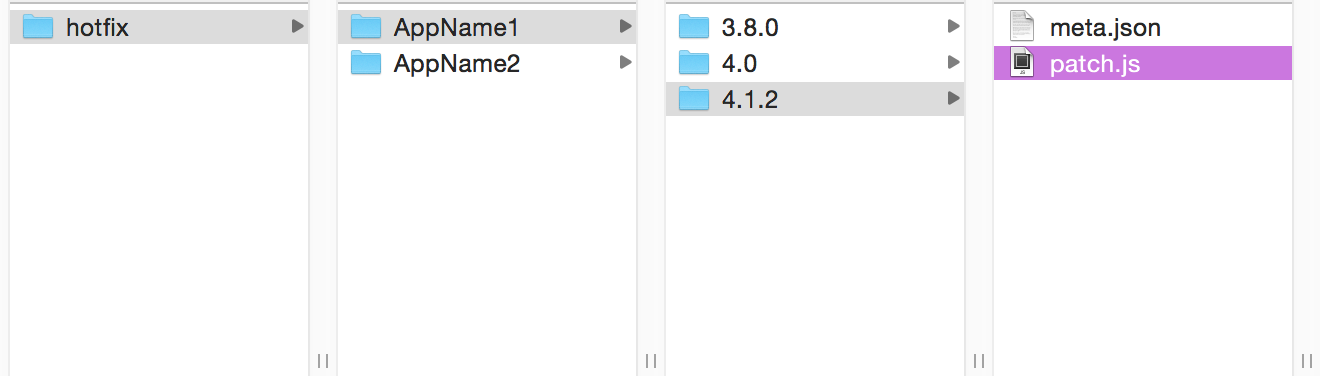
八、JSPatch 部署安全策略 8

## 一、什么是JSPatch

一个ios 动态修复bug补丁的第三方库，针对已经上线的APP，苹果更新审核速度较慢，而bug却需要紧急修复的情况，JSPatch能够很好的热修复处理这个问题。

## 二、仓库设置方法（后台文件管理）

js文件肯定不能随便往后台某个文件夹一放就让前端去下载了，虽然使用方便但是在App或者版本较多时容易混乱。建议专门搭建一个远端仓库，仓库里 主要就是文件夹和js文件，当需要提交js文件时，从主干迁出一个分支，在合适的地方新建文件夹并添加js文件，然后给主干提Pull Request， 这应该是一个麻烦但是规范的流程。文件夹结构参考下图:



第三层文件夹里，可以用版本名称也可以使用build号。

## 三、ios前端配置

* 项目中导入该库：

pod 'JSPatch'

* 安装命令：

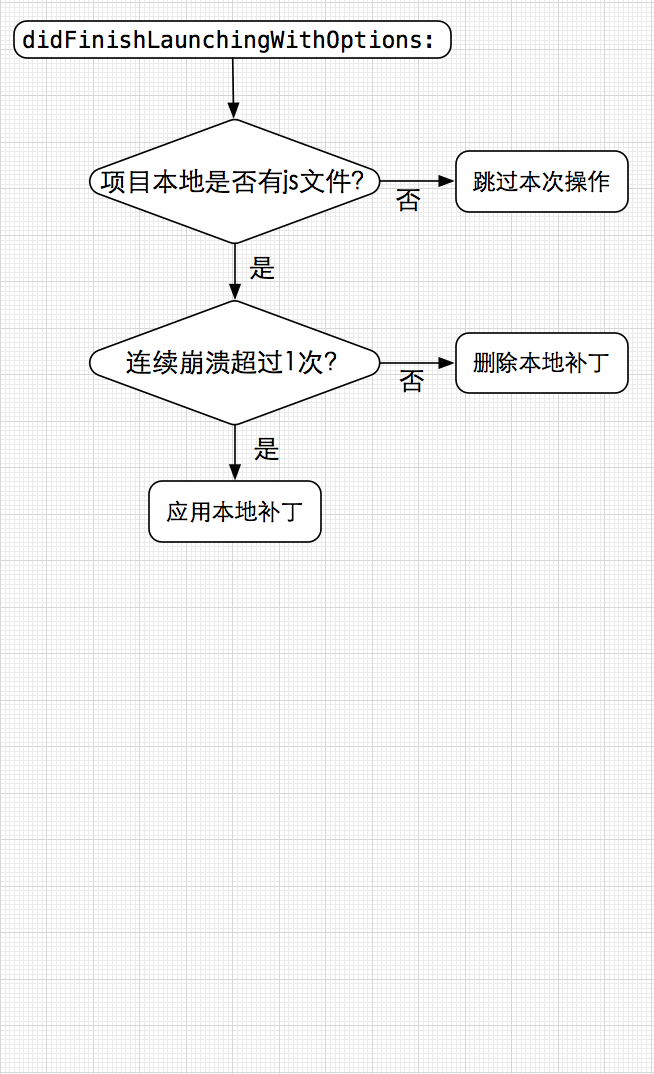
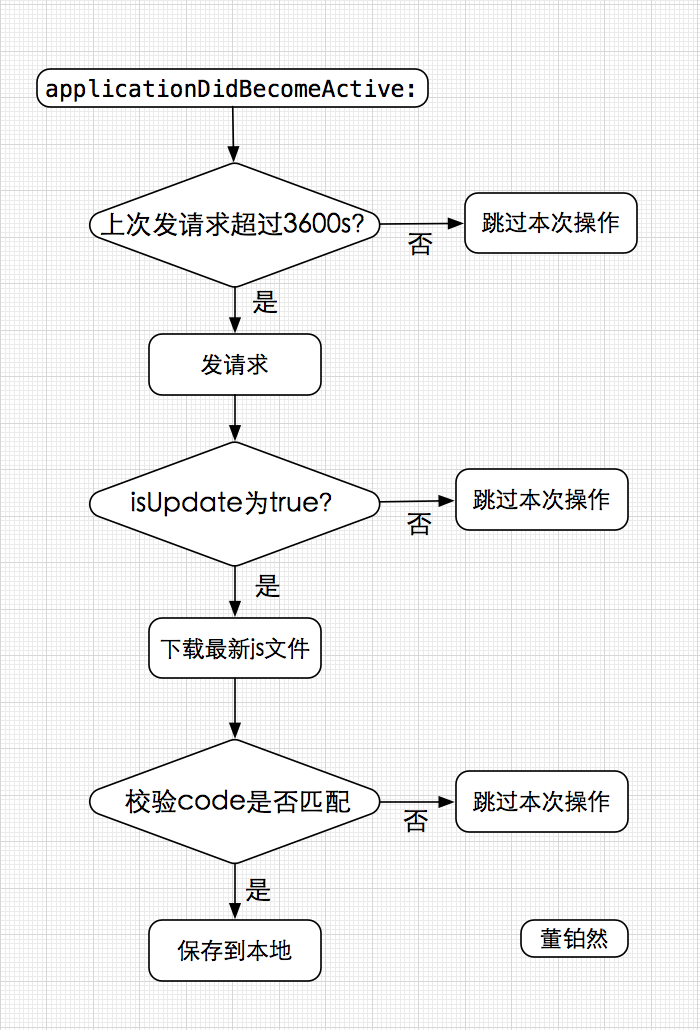
pod install

* 参考demo：<https://github.com/niexiaobo/JSPatchUse>

## 四、更新频率，及App更新方法代码名称

我之前看到很多人把使用js和下载js的代码都放在了didFinishLaunchingWithOptions： 这个方法。我觉得有所不妥，因为如果这个app用户一直放在手机的后台（比如微信），并且也没出现内存警告的话，这个方法应该一直不会调用。我建议的是： 使用js文件的代码放在didFinishLaunchingWithOptions： 而下载js文件的代码放在applicationDidBecomeActive: 因为这个方法在程序启动和后台回到前台时都会调用。并且我建议设置一个间隔时间，根据一些数据和权衡之后我们采用的是间隔时间设为1小时。 也就是说每次来到这个方法时，先要检测是距离上次发请求的时间间隔是否超过1小时，超过则发请求，否则跳过。

前端更新逻辑：



## 五、请求数据（前后台接口设置）

安全相关工作如果没有做好，最惨的情况是人家可以通过js文件调用你的任何OC方法，我们肯定不能允许此类事情发生。一般在js文件提交到仓库以后 后端应该对这一段js代码进行 md5或者更高手段的编码，并将这段编码与文件存在一起，上图中得meta.json里存的就是这一段编码。发请求下载的时候应该是需要拼上项目appname，version等参数。

之后在发请求的返回值的结构应该是大致如下：

{

data: {

isUpdate: "true",

content: "require('MTPoiFeedbackM')

defineClass('MTFeedbackRankCell',{

setPoiFeedback:function(poiFeedback){

self.ORIGsetPoiFeedback(poiFeedback)

var temColor = require('UIColor').lightGrayColor();

self.detailLbl().setTextColor(temColor);

}

})",

code: "9c944f39e57f2e50bdb85deb878cc0f798efb9b0"

}

}

就是首先有个字段告诉我们较上次下载的js文件是否有更新。如果为true再检测下方返回的code与内容编码后得到的code是否相同。当然这个内容也可以不直接返回而是返回一个下载的url也是完全可以的。

{

data: {

isUpdate: "true",

content: "url 下载地址",

code: "9c944f39e57f2e50bdb85deb878cc0f798efb9b0"

}

}

## 六、前端下载和使用

1）本地使用

[JPEngine startEngine];

NSString \*sourcePath = [[NSBundle mainBundle] pathForResource:@"demo" ofType:@"js"];

NSString \*script = [NSString stringWithContentsOfFile:sourcePath encoding:NSUTF8StringEncoding error:nil];

[JPEngine evaluateScript:script];

2）下载使用

[NSURLConnection sendAsynchronousRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:@"http://cnbang.net/test.js"]] queue:[NSOperationQueue mainQueue] completionHandler:^(NSURLResponse \*response, NSData \*data, NSError \*connectionError) {

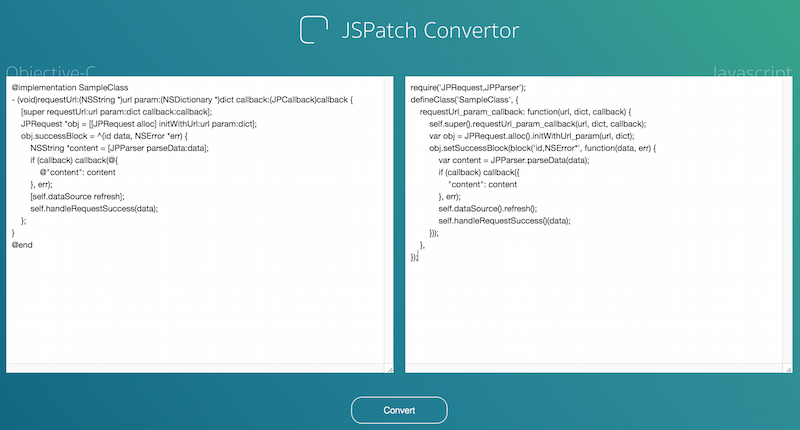
    NSString \*script = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];

    [JPEngine evaluateScript:script];

}];

## 七、JSPatch语法

这个JSPatch语法 并不是一个正式的语种，大家不会投入太大的精力来仔细学习，所以作者本人也提供了一个简单粗暴的OC到JS直接转换地址：<http://bang590.github.io/JSPatchConvertor/>



这个地址亲测一些简单的写法是正确转换的，但是比较复杂的语法还是不能让机器直接搞定，还是需要人工修改的。作者也在不断完善这个工具。

## 八、[JSPatch 部署安全策略](https://segmentfault.com/a/1190000003689114)

使用 [JSPatch](https://github.com/bang590/JSPatch) 有两个安全问题：

1. 传输安全：JS 脚本可以调用任意 OC 方法，权限非常大，若被中间人攻击替换代码，会造成较大的危害。
2. 执行安全：下发的 JS 脚本灵活度大，相当于一次小型更新，若未进行充分测试，可能会出现 crash 等情况对 APP 稳定性造成影响。

详细参考博客：<https://segmentfault.com/a/1190000003689114>