2016年6月13日 星期一

Hybrid框架文档

文档目的

本文档主要针对我们自组的hybrid框架进行简单技术的实现介绍和实现约定。

纯Native架构与Hybrid架构的技术分界

首先是技术没有优劣，无需纠结语言和实现的优劣，取各家所长才是正确的发展方向。亦不比较市面上列如Iconic,ReactNative等需要编译转换的Hybrid框架。

在做Hybrid架构的时候需要清楚的了解到Native与前端的技术实现优劣在哪，首先在移动APP开发中Native永远都是宿主环境，前端在宿主环境不暴露接口的情况下起到辅助作用，但是在暴露接口的情况下前端可以实现控制大部分交互和界面展现。

前端技术实现优势：

- 快速的界面实现
- 跨平台
- 快速发布
- 入门难度低

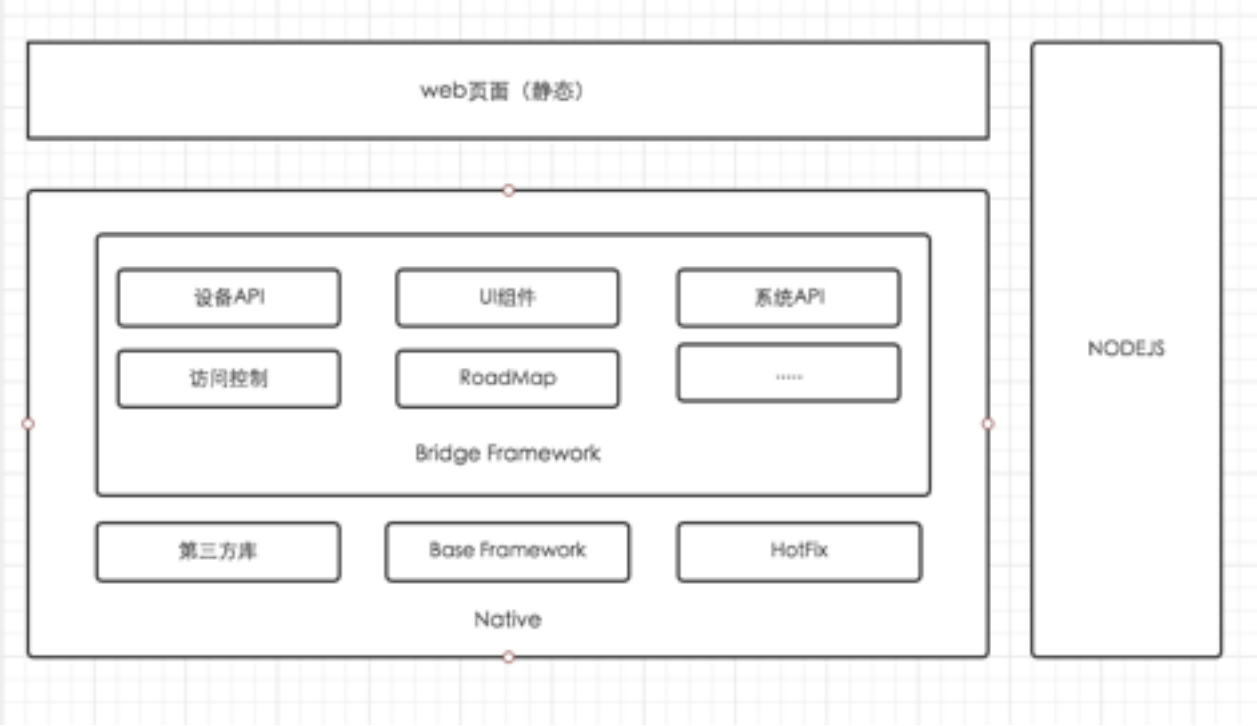
Native技术实现优势：

- 良好的交互体验
- 完整的设备控制权限
- 性能优化上限高

由于各自有各自的优势，所以才出现了Hybrid架构的实现，由前端来实现页面的布局、数据呈现、底层API调用，由Native来实现底层API封装、转场动作、复杂特效、网络、访问控制等。

工程架构

如图所示



NODEJS

在整个工程中，NODEJS负责制作，打包，和管理资源包。强制升级，HotFix，还有其他与APP相关的页面工作。

Native

第三方库：友盟，ShareSDK，ChinaPay等等

Base Framework:基础业务处理封装

HotFIx:在线热补丁

Bridge Framework:封装设备API，系统API，访问控制，RoadMap，静态资源管理等

Hybrid开发分解

开发分解的基础点在IOS7.0和Android4.2之后，之前的版本实现和兼容不做介绍，基础依赖点为javascriptcore的引用。

2016年6月13日 星期一

初期仅为了满足目前开发需要进行定义，以后随着业务的需要视情况而定封装其他的接口

Bridge Framework接口定义

NFBridge.NGlobal 全局类

NFBridge.NGlobal.ready

注入状态方法，主要用于判断注入是否完成。Native端只实现注入，JS通过判断全局对象是否存在可以知道是否可以开始进行操作

NFBridge.NGlobal.error

错误捕捉方法

NFBridge.NGlobal.debug

调试捕捉方法

NFBridge.Auth 认证类

具体定义以代码中封装的设备类接口为准，主要用于认证当前调起来源的鉴权和授权设定

NFBridge.Device 设备类

-电池

-摄像头

-加速器

-陀螺仪感应器

-指南针

具体定义以代码中封装的设备类接口为准，主要用于设备API的封装

NFBridge.Sys 系统类

-调起应用

-网络连接状态

-手机通讯录

-设备信息

-文件系统

-地理位置

-系统语言信息

-设备通知

-二维码

-截屏

-数据存储

-拦截器

-缓存

具体定义以代码中封装的设备类接口为准，主要用于系统信息的封装

NFBridge.Map RoadMap类

原则上根据产品来定义页面路径的解释，可以根据业务自行定义schema，原则上，如 NOAH://to/产品详情 这样。

NFBridge.UI UI组件类

具体定义以代码中封装的通用类接口为准，主要用于创建webview

这里解释下为何将创建webview给封装起来，考虑如下任务

比如现在BAT此类公司的软件中，经常会出现一些营销性业务，比如说什么红包雨之类的会出现在APP下拉刷新这种场景上展现，经常会在当前的页面上创建一层webview的蒙版，然后由前端工程师再实现此类活动。

NFBridge.Plugin

预留插件类

NFBridge.callHandler(handlerName,data,callback)

此方法为Native端使用，用于与JS通讯，真实开发情况中，需要由Native调用JS的场景几乎不再存在，预留只为不时之需

NFBridge.registerHandler(handlerName,callback)

此方法为JS端使用，用于调Native方法

格式参考:

IOS Native定义（大概这样定义）

```
func Alert(titile:String,params:[String: AnyObject],callback:[String: AnyObject]){
}
```

Android Native定义

请补充

JS端调起定义

```
NFBridge.Device.Alert({
    title:"测试",
    params:{},
    callback:function (data){
        //do something
    }
})
```

打包发布和部署

打包

在考虑用户使用性的情况下，在每个版本发布的时候直接把静态资源打包进APP端，格式建议使用7z。

NODEJS 部署

NODEJS部署由PM2管理，一般来说 内核*2 = 进程实例数。

NATIVE

部署各自的市场，Android封版后由测试给运维发布。

安全

爱加密移动加密解决方案

多WebView展望

在目前比较成熟的Hybrid框架（非编译转换型）中实际是非常常见的，因为在单webview的情况下面，内存控制一直是一个问题，在实现复杂的逻辑或展现的时候，经常会内存暴涨，有时会导致APP的崩溃，而且Native由于和js的回收方式不同，还无法控制（比如IOS是引用计数，如果用webview的话内部执行什么东西都是托管给执行环境的GC的，这种随机回收的机制让作为容器的Native端无法掌控）。但是由前端控制的创建多个webview的层叠的话，可以在代码上写的比较精，以此来隐式控制容器的内存，保持一个线性增长的趋势。但是此方法也非常依赖于Native容器的封装，封装的越成熟，那么使用此类技术的人员也就越爽，最终由js管理一切。