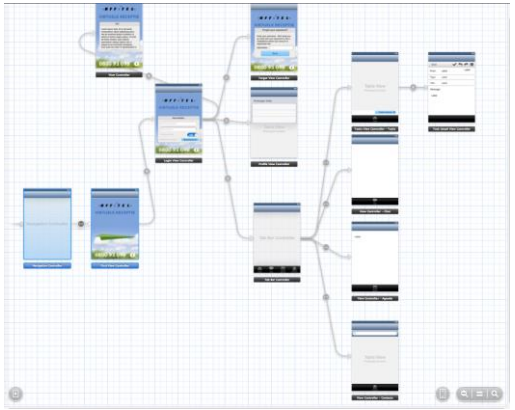
电商APP客户端Hybrid架构实践

秦曲波
1药网B2C技术部技术总监

	Native APP	Web APP	Hybrid APP
优点	性能好，用户体验好	开发门槛低，上线快	结合前两者的优点
缺点	更新不及时	性能差，移动端用户体验不好	对架构要求高

改变Native APP的开发思维模式

热更新



APP并不是一个孤岛

- Native和内嵌Web页面的互通互联
- 应用和应用之间的互通互联
- 线上和线下场景的互通互联

自由的跳转

- 应用内Native页面和内嵌的Web页面之间的跳转
- APP应用之间的跳转
  - 第三方浏览器中采用Deep Linking方式的跳转
  - 社交类应用和我们主APP之间的跳转
- 传统的短信和Push Notification触发的跳转
- 线下和线上各种二维码扫描后触发的跳转，也可以基于LBS信息，跳转不同的页面

页面URI规范

- 和URI的定义一样， 三个部分组成
- scheme: 有两种类型
  - 基于页面的，yhd://
  - 基于功能的，yhdfunction://
- host:对应native的各个页面，search对应搜索页，productdetail对应详情页，等等
- 参数:在Body里以JSON的格式约定，Native页面通过获得Encode后的JSON格式，转换成字典或数据对象

```
xxx://search?body={\"keyword\":\"软包抽纸\"}

xxx://productdetail?body={\"pmId\":21456956,\"promotionId\":\"\"}

xxx://web?body={\"title\":\"新品试用\",\"url\":\"http://m.yhd.com/try/\"}

xxxiosfun://addCart?body={\"pmId\":21456956,\"productId\":18148037}
```

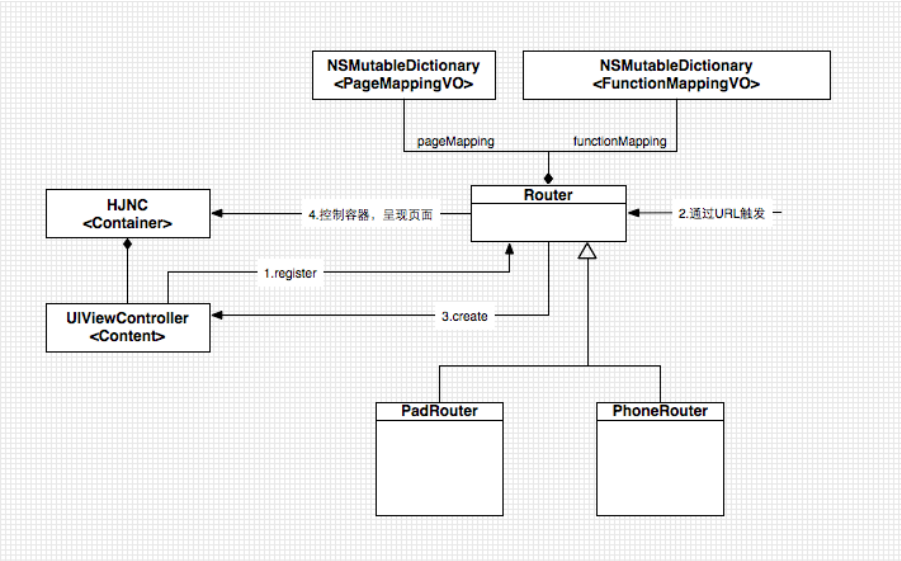
页面的注册机制

```
// 注册落地页面
+ (void)load
{
    PageMappingVO *vo = [PageMappingVO new];
    vo.createdType = MappingClassCreateByCode;
    vo.className = NSStringFromClass(self);
    [[Router singletonInstance] registerRouterVO:vo withKe
y:@"giftCardChoosePay"];
}

// 注册Native功能
+ (void)load
{
    NativeFuncVO *voShare = [NativeFuncVO new];
    voShare.block = (id)^(NSDictionary *params) {
        return [self shareWithParams:params];
    };
    [[OTSRouter singletonInstance] registerNativeFuncVO:voS
hare withKey:@"share"];
}
```

- iOS是通过+load方法，在应用启动的时候完成页面控制器View Controller的在Router中的注册，每一个落地的页面在自己的View Controller的+load方法中将自己注册到Router中。
- 这个时候页面对象并没有创建。只有触发页面跳转的URL到达Router后，通过Router解析才会创建落地页面的实例。

Router的实现结构



- 落地页面在APP启动的时候注册自己到Router中
- APP接收到各种触发事件，获取URL，转换成注册的VO对象

  根据VO对象，创建落地页面实例，若是Native的Function，获取注册的回调函数，由该函数调用完成实际功能的类方法
- Router通知容器完成页面的呈现和加载

简化了页面之间转跳的统计系统埋点

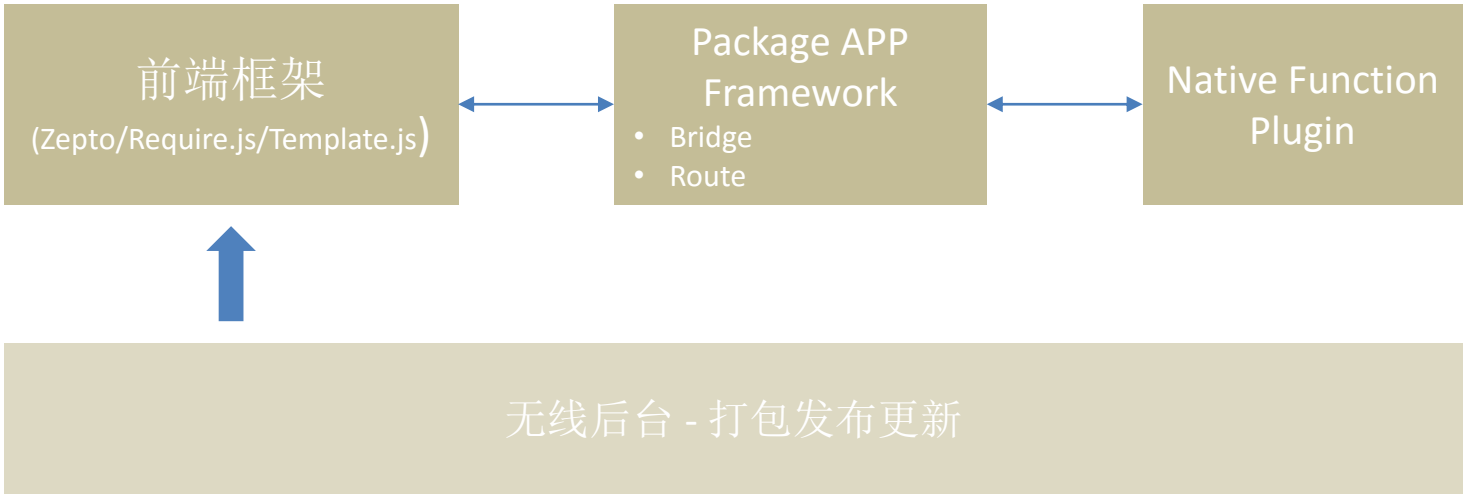
在没有采用Router机制之前，虽然我们提供了页面之间跳转的BI统计库的封装， 但是每个页面还是要插入自己的BI埋点代码。现在Native页面也可以通过Router完成页面的跳转，这部分的埋点不用写在各个分散的页面中，可以集中在Router或HJNC处理。BI所需要的参数，Route可以从落地页面绑定的参数中提取。

```
- (void)doBITrackerFromVC:(UIViewController *)aFromVC destVC:(UIViewController *)aDestVC andIsPop:(BOOL)pop
{
    if ([aFromVC isKindOfClass:[OTSVC class]] && [aDestVC isKindOfClass:[OTSVC class]]) {
        OTSVC *fromVC = (id)aFromVC;
        OTSVC *toVC = (id)aDestVC;

        // 监听有打开渠道的router
        [[OTSBITracker sharedInstance] generateSessionId];
        NSString *btu = aDestVC.extraData[@"tracker_u"];
        if (btu) {
            [OTSUserDefault setValue:btu forKey:BI_OpenTrucku];
        }
        // 监听CMS形式to Native的参数回传
        NSString *tpString = aDestVC.extraData[@"tp"];
        if (tpString) {
            NSArray *tpArr = [tpString componentsSeparatedByString:@"."];
            NSString *pageTypeId = [tpArr safeObjectAtIndex:0];
            NSString *pageValue = [tpArr safeObjectAtIndex:1];
            NSString *tpa = [tpArr safeObjectAtIndex:2];
            NSString *tpi = [tpArr safeObjectAtIndex:4];
```

URL Router其他应用场景的扩展

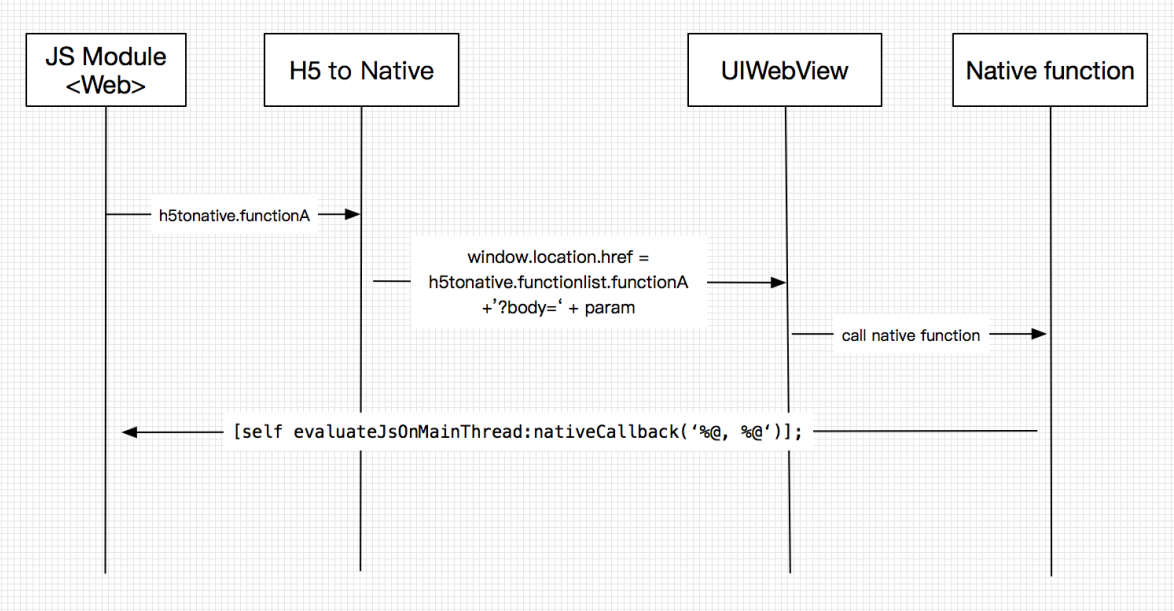
- 将注册的页面信息通过接口，在后台配置下发到APP，可以实现页面的动态配置。例如同一个购物路径上，可以配置不同的实现的页面，可以根据不同的用户配置不同的页面，再根据返回的BI统计数据分析不同配置下的效果。真正可以做到千人千面。
- 也可以根据具体的时间和活动要求配置动态页面，撤下一些过时的功能
- **必要时**若某个Native页面也重大的BUG，也可以实时的用H5页面替换，线上实时修复BUG，不需要发包，等待Appstore审核。


前端框架
(Zepto/Require.js/Template.js)
Package APP Framework
• Bridge
• Route
Native Function Plugin
无线后台 - 打包发布更新

从JS-Bridge到Package APP

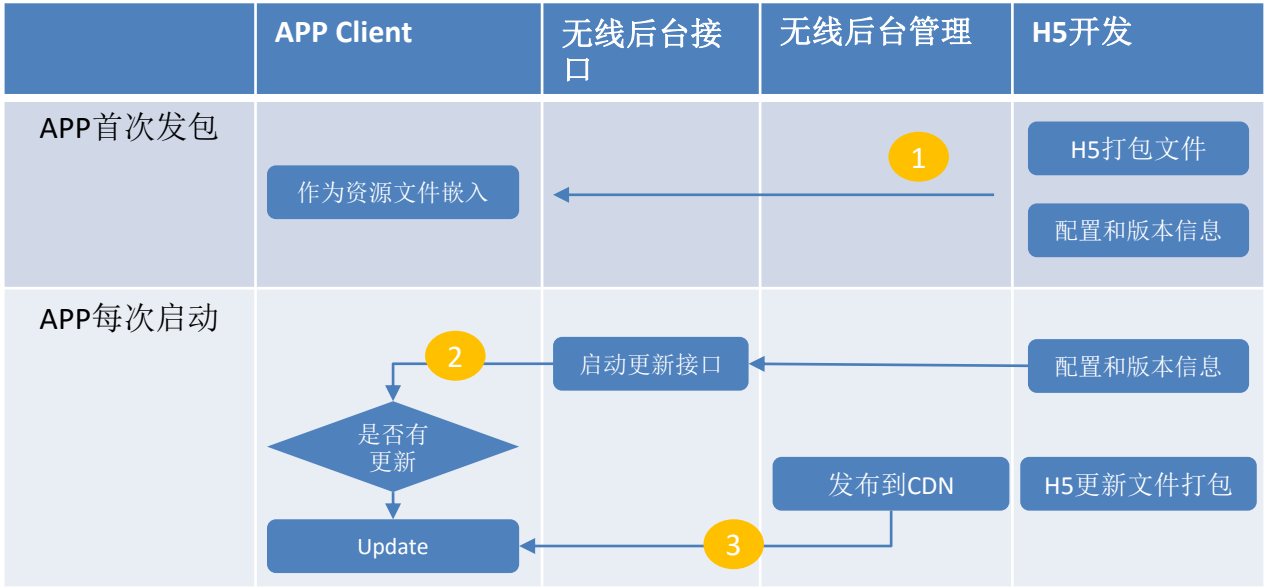
- 我们将H5开发中用到的JS和CSS的开源框架和技术团队稳定和常用的JS库，打包在APP中，最大限度降低加载Web页面所需要的下载资源。
- JQuery用适合移动端的Zepto替代。
- 用Require.js管理H5模块之间的依赖。
- JS端封装了JS和Native的通讯的模块 h5tonative, 并定义了callback的规范。
- 采用轻量级的前端模板template.js，实现了前端开发的MVC模型。
- Native提供H5所需要的数据模型，数据访问的接口都有Native完成，保证了接口数据的安全性
- 对内嵌的H5应用设计了一套版本更新和安装机制

客户端Native和JS调用和回调机制的实现



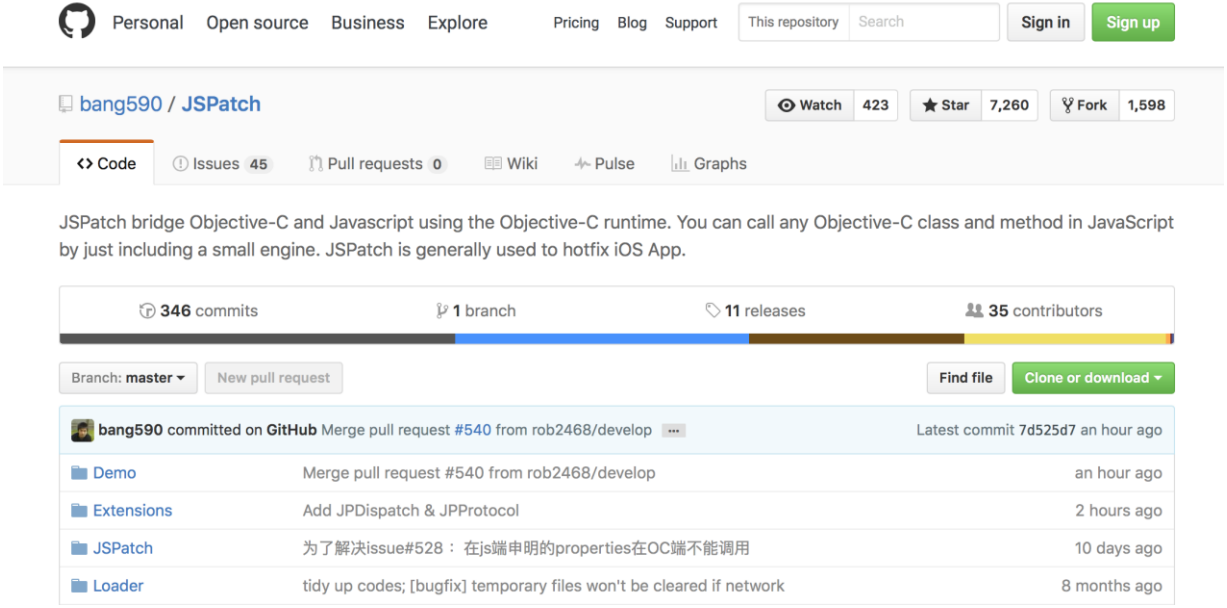
- Param中第一个参数是回调JS的ID
- 第二个参数是返回的值

版本更新机制的实现



- 支持按照H5模块单独更新
- 支持MD5校验和签名，防止H5功能被篡改
- iOS7以后支持通过静默推送，触发应用后台下载H5更新包，用户在没有感知的情况下升级H5的功能
- 下发路由表

iOS动态更新方法 - JS Patch



- 在开源的基础上做了一定的封装，
- 支持签名和校验，
- 支持对特定版本的Native包的Patch
- 支持后台下载Patch包等

What's Next – Reactive Native

- 学习成本
- 维护成本
- 调试问题

谢谢

QQB

上海 浦东新区

扫一扫上面的二维码图案，加我微信