Hybrid APP 本地缓存技术预研一期总结

1. 预期目标

> 客户端将前端所需要的静态资源进行缓存，并截获前端请求，根据与前端制定的路由接口，获取本地缓存资源和线上资源返回给webview；
>
> 对比非本地缓存方案，比较首屏渲染时间，分析结果，得出本地缓存客户端与前端最佳实践。

2. 任务安排与完成情况

客户端：
陈松&李良燕（完成100%）：

> 客户端截获http请求
> 客户端截获https请求
> 客户端router机制实现，并读取本地缓存
> 联合调试

前端：
汤亮（完成100%）

> 用react重构personal页面（首页、列表页）
> react组件（slide，下拉刷新）
> react基础controller、model

张宇航（完成100%）

> 获取当前personal H5页面（首页、列表页） 储存到本地
> 设计前端和客户端接口制定
> 构建react本地、开发、测试环境
> static server 开发环境搭建
> 联合调试

3. 性能对比实验结果：

三组对比实验：

3.1 内网访问小网页（<500K）

客户端webview使用本地缓存所用时间：大约300ms（页面渲染完所用时间）(所有资源加载完共需要500ms)
客户端webview不使用本地缓存所用时间：大约1s

3.2 外网访问小网面 (<500K)

客户端webview使用本地缓存所用时间：大约300ms（页面渲染完所用时间）(所有资源加载完共需要600ms)

客户端webview不使用本地缓存所用时间：> 2s

3.3 外网访问大网页（>30M）

客户端webview使用本地缓存所用时间： 大约400ms（页面渲染完所用时间)(所有资源加载完共需要650ms)
客户端webview不使用本地缓存所用时间： > 2min

说明：
1). 以上所用时间均为客户端webview启动到渲染页面所用时间
2). 客户端webview的网络环境为接入公司内网的WIFI网络
3). 不使用本地缓存方案所用时间的计算方式为没有启用webview缓存机制。

4. 实验结果分析

使用本地缓存技术，app从webview启动到首页渲染时间大约为300ms。（我这里所说的渲染指的是用户看到有样式的DOM渲染的时间，即为DOMLoaded时间＋css加载所需要的时间。）相比于不使用本地缓存技术，所渲染的首屏渲染时间有了大幅度缩短。然而，300ms与之前的预期还是有了一定出入。

备注：以下为现在手淘的首屏渲染数据：
App全链路实现 1S 法则：强网（4G/Wifi）实现1S首屏（包括图片）加载、3G 1S首包返回、2G 1S建连，并且实现高复用 从底层到前端。

5. 实验中遇到的问题分析与解决：

客户端loading去除时机（即首屏渲染完的时间）

对于非类似reactjs，建议在css加载完成即可去除loading图标，渲染html页面；

而对于reactjs由js驱动渲染页面的框架，则建议在js加载后，去除loading图标，渲染html页面。

webview网络下载线程个数（IO并行）

根据android和ios打出的日志，webview并行3个线程进行加载（与基于webkit内核的chrome6个线程不同）。
除此之外，单个资源的加载时间(从本地缓存读取)也很少，IOS为<5ms & android为<20ms。由此可见，使用本地缓存技术，资源加载时延已经不再是webview首屏页面渲染耗时瓶颈。

webview启动时间

由于webview的启动时间在100-150ms左右，虽然本次试验并没有涉及页面切换速度的检测。但是由已有的数据和结论，我们可以得出结论：在非首屏页面加载的情况下，其他页面渲染时间应该小于300ms（去除webview启动时间，应该在150ms－200ms）。

300ms耗时是如何产生的?

IOS下：webview启动时间（80ms）、html＋css的加载时间20ms，其他时间130ms；

Android下：webview启动时间（120ms）、html＋css的加载时间20ms，其他时间160ms。

其中，其他时间包括HTML的解析时间、DOM树建立的时间、CSSDOM树建立时间，以及部分页面的渲染时间。

然而，从数据看出从css下载完成到所有资源下载完成，除去期间加载js、图片的加载时间（<50ms），依旧有250ms的延时。这部分时耗主要是消耗在了HTML解析、图片渲染、页面重排、页面重绘等过程，但是超出了之前的预期，也称为页面渲染的另一个瓶颈。

6. 结论

目前用hybrid app本地啊缓存方案可以将页面从app启动到页面渲染时间为大约300ms，相比于之前不用本地缓存方案有较大的提升。

然而，进一步提升目前来看已经较为困难，如果想继续提升的话，可以从减缓webview的启动时间、增加webview请求线程个数、增加客户端读取sd的IO速度、加快页面渲染速度等方面进行进一步提升，但是每一方面都需要进一步深入调研，投入产出是否合算需要考虑。

另外，现在实验数据的300ms，并不意味着之后真实情况下，应用首页渲染时间一定会小于300ms。因为在真实情况下，由于考虑缓存包是为最新等问题，需要通过网络请求进行确认造成一定的时延。

7. 遗留问题

现在HTML解析和DOM渲染成为了，性能优化的主要瓶颈，占据了大约200ms－300ms的时耗。后续有时间的话可以对其进行进一步的预研和优化。

8. 二期计划

updating机制

预期目标updating server和客户端制定更新接口，实现客户端实现静态资源更新，讨论得出资源更新方案最佳实践。

react-native实践

实现react－native客户端demo，以及updating的客户端和服务端的基本实现。