build:0.3.2 - 2013.10.22

Performance Report of Baidu iPad Client

Syllabus

- 现有性能情况
    - Native
    - WebView
- 性能现况分析结论
- 优化方案
    - Native
    - WebView
    - Common Issues(Place 项目存着的共通问题)

现有性能情况

- **Native**
    - **实验尝试**
        - 载入简单的页面(50个 `<li>SomeText</li>` 组成),客户端载入页面后卡顿现象依旧严重，仅有不到 **20fps** 的滚动，该较为极端的页面表面webview *滚动效率* 低下的 **主要原因** 并非由webview页面的逻辑造成
- **WebView**
    - **加载以及滚动**
        - 首屏显示情况
            - 数据拉取时常占70%，具体数据采集如下表(POI点涵盖几个热门类别的POI点,iPad3,IOS5.1)
            - 针对数据裁剪可以较为高性价比的取得首页加载速度
            - (*)代表该POI点为首屏进入

页面内容展现时长 (ms)	请求时长 (ms)	数据绑定代码运行时长 (ms)
4304	(*)2413	1891
2731	1359	1372
1162	668	494
1072	737	335
2197	(*)1796	401
1137	833	303
2458	1498	960
1194	799	395
1078	720	358
2807	1487	1320
1985	1062	923
2007	(*)1667	340
2410	1410	1000
1051	760	291
1062	755	307
1066	780	346
--	**~70%**	**~30%**

- **交互响应速度**
  - 在低端机器上(<=iPad2,IOS5/6),性能有着较为大的问题
  - 在>=iPad3(IOS5/6)上，流程度属于基本可接受，滚动帧数为～23fps以上
  - Webview本身的事件代码构建问题导致的交互响应速度缓慢，主要原因是Common Issues里面提及的click事件[1]

性能现况分析结论

- **WebView的效率依赖与Native效率的提升**
- 在WebView层面上 以及 native与webview的交互上(如通信方式，首次加载的方案)有比较**多(不是大)** 的提升地方，但实际上，在这个层面上的投入仅仅能优化可能不到20%(乐观估计)
- FE 这边能推动的性价比最高的两个地方是 **数据裁剪** 与 **交互性能改进**

优化方案

- **Native 性能优化**
  - Note
    - 因为没有对Native的代码有深入的了解，仅仅从业界以及结合FE这边的情况进行猜想与方案构建
  - Native性能比较差[2],初步 **猜测** 其导致打开的webview所持资源不足以流畅交互,单独的webview iOS app流畅运行place页面
  - 关于调整webview高度的策略
    - 现有情况[3]
    - 方案(参考Google Currents的方案[4])
      - 与Native的RD进行沟通联调,*pending*
  - 消息传递
    - 现有方案[5]
    - 方案(参考Google Currents的方案):
      - 创建一个hidden iframe，通过更改iframe的hash值进行传递
  - 初始化UIWebView策略
    - 现有策略[6]
    - 方案
      - 在进入ListView部分，进行空模板绑定加载
- **Webview 性能优化**
  - 预渲染
    - 现有情况[7]
    - 方案
      - 图区(等高)，简介部分的外部框架可提前渲染，后续内容填充其内
      - 适当调整图区与行业相关的dom位置，可减少一次reflow,repain
    - **预期收益**
      - 提高首次加载模板的速度的~30%
    - Cons
      - 客户端需要相应的支持，沟通成本上升
  - 持续优化Dom树
    - 方案
      - 在前期优化的基础上，对图区，地图显示dom进行优化，减少层级
    - **预期收益**
      - 根据不同行业页面交互的复杂程度有不同程度的提升，如复杂的酒店部分在native成功优化的基础上能提升~15%的交互响应/滚动速度
  - less cool looking:

        - 现有情况[8]
        - 方案
            - 在权衡nice looking 与 performance 的程度上去除高耗rendering属性[9]
    - character set 的指定，减少decode时间
        - 如在*.html头部添加 `<meta http-equiv="content-type" content="text/html; charset=UTF-8;charset=utf-8">`
        - 预期收益数值有待策略(pending)
    - POI点图片的加载,现有的方式导致多次渲染问题[10]
        - 方案:
            - 根据客户端型号请求size与container等同的图片,指定img的width和height,缺点是设备iPad关联性较大
            - 请求一张稍大的图片，直接采用css控制size与居中
            - 尝试Responsive设计
        - **收益**
            - 去除页面加载后图片闪动问题

- **Common Issues**
    - Remove unused css
        - 现有情况

```
540 rules (70%) of CSS not used by the current page.
detail.css: 83% is not used by the current page.
index.css: 36% is not used by the current page.
scope.css: 89% is not used by the current page.
```

        - 方案
            - 半自动剔除(历史遗留问题导致修改成本较大)
        - **预期收益**
            - 根据不同机型的webview有不同程度的提升，提升页面渲染速度~10%
            - place包的大小相应的css文件大小平均减少30%
    - 包的大小
        - 代码压缩
            - 现有情况[11]
            - 方案:
                - uglify
            - **预期达到的效果(by grunt uglify)**

```
  ➜ dist git:(master) ✗ du -sh ipad-webview-test.js
  376K    ipad-webview-test.js
  ➜  dist git:(master) ✗ du -sh ipad-webview-
test.min.js
  132K    ipad-webview-test.min.js
```

        - 代码依赖精简
            - 某些行业不需要依赖的库的引入导致的script-block-rendering[12]
    - 大量的parseHTML导致的渲染性能低下
        - 现有情况[13]
            - 同样的情况出现在现有的组件平台
        - 方案
            - 建立一个由外层容器负责渲染的机制
            - 旧有的Place详情页
                - 历史遗留问题导致修改成本巨大
            - 组件平台
                - 在导入模板引擎后可进行该项工作(10.18号后)
        - **预期达到的效果**
            - 根据不同机型提升效果不同，预计能提升~100ms
    - click 事件更换为 touch 事件
        - click 响应缓慢[14], 建议开发tap事件轻量lib，统一事件调用方式, ele.addEventListener("tap",func);
        - **预期达到的效果**
            - 每次关于手指点击的交互的提升能符合

$$\frac{t - 300ms}{t}$$

交互相应速度提升效果显著

    - 延时加载
        - *pending*（非首屏页面部分图片较少,收益不明显,故定位pending）

Power By MarkDown

Full Version DOCS

---

1. 该方式的转变是所有修改中性价比最高的方式，预计投入2天能完成,收获的交互速度能有较大的提升↩
2. Instrument记录打开webview后的情况,注意到有内存泄漏↩

3. 每次渲染一个模块的时候，必须向native发送消息，多次让[UIWeview setFrame]↩
4. Podcast #36 – “Google Currents”↩
5. 直接通过*window.location.href=command*，在OC里面捕获该URL Redirection↩
6. 第一次点击进入详情页的时候，首次渲染导致的体验已经速度不友好，退出详情页的时候，UIWebview保持当前内容，下次进入另外的POI点之时，重新导入已经绑定数据，没有对整个页面进行重新的渲染↩
7. 数据返回后html生成后一同跟数据显示↩
8. iPad应用高级消耗css属性比较多↩
9. 如box-shadow,border-radius↩
10. 图片首次显示渲染，resize之后的渲染，外加javascript对图片size的计算时间↩
11. 仅仅做空格空行等的文本压缩操作↩
12. 如iscroll,tangram↩
13. 这里和这里↩
14. click事件在mobile响应模型里面，处于一个在call stack比较低的地方，前面发生了touchstart,touchmove,touchend等事件，浏览器大概需要300ms才会响应click事件(用以检测是否存着double click事件)，采用touch事件预期在100ms内响应callback↩