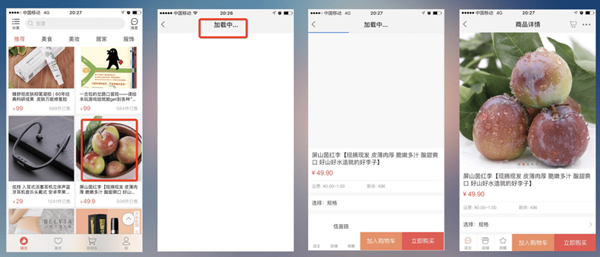
**浅谈H5前端性能测试实践**

有赞技术

H5 页面发版灵活，轻量，又具有跨平台的特性，在业务上有很多应用场景。但是同时对比 App，H5 的性能表现总是要逊色一筹，比如页面打开往往会出现白屏，滑动列表等交互场景下也不如 Native 页面流畅。针对这些白屏、卡慢之类的问题，我们测试该从哪些方面去展开测试分析和数据对比呢？接下来笔者分享一些 H5 前端测试实践的经验，抛砖引玉，希望大家一起谈论，一起挖掘更多有价值的课题。

### 一、开篇：H5 页面加载过程浅析

如下图所示，是精选平台打开 H5 页面的几个过程截图。

[](http://s4.51cto.com/oss/201808/29/a1836e6f5007d17806552f090ffcd6c3.png-wh_651x-s_3077574673.png)

图一到图四可以简单分类，图一是 App 负责做的事情，主要是初始化 Webview 上下文；后面三张图则是一个H5页面加载的过程。其中，App 这个阶段的耗时，主要是 Native 代码的耗时，这里先不展开讨论，我们重点放在后面几个阶段。第四个图是用户直观看到的第一屏页面，我们通常称为首屏。

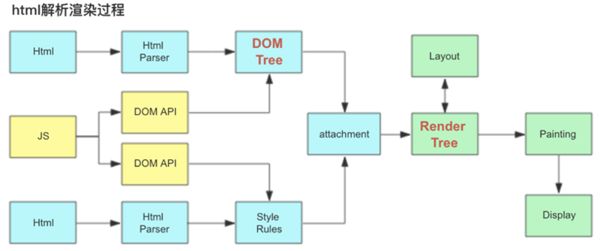
[](http://s5.51cto.com/oss/201808/29/7261a7675afa1e15105645bd4d3c4355.png)

1）加载网络请求

这个过程主要是 Webview 拿到 H5 页面 url 之后，调用 loadUrl 方法，开始去网络上请求第一个资源文件。这个阶段主要包含 dns 解析、建立网络链接、数据传输的耗时。

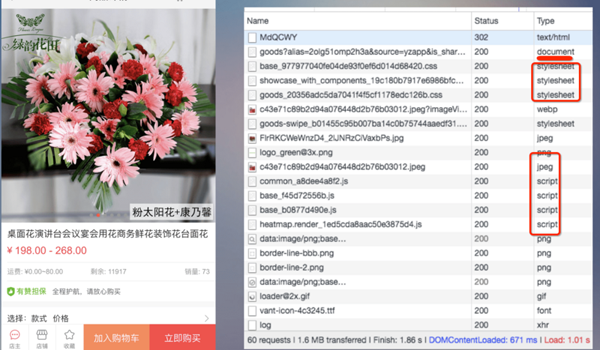
2）html 解析

Webview 拿到 html 返回后，需要从上至下解析 html 中的标签和内容，识别外链资源、计算页面框架的布局，并渲染绘制出来。在这个过程中会构建出负责页面结构的 DOM Tree 和负责页面布局展示的 Render Tree，如下图所示：

[](http://s2.51cto.com/oss/201808/29/55b3a6771ab9fc437bd02f419fc6f44d.png)

3）外链资源加载

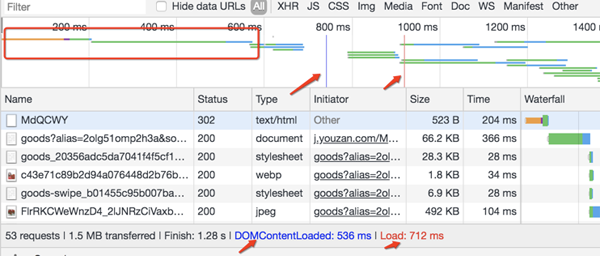
这部分主要是从网络上加载外链的 css、图片和 js 等，再重新填充到 html 中。之后重新进行一次 layout 布局计算和页面渲染绘制，此时看到的才是有完整内容的页面。如下图所示，页面需要等图片和 css 加载出来后才能展示，js 也是外链资源，不过一般来说，只要放在 html 底部加载，就不会阻塞页面的渲染和展示。

[](http://s5.51cto.com/oss/201808/29/294299001e3509db5218d0b2428b6dc1.png)

### 二、实例分析：白屏问题

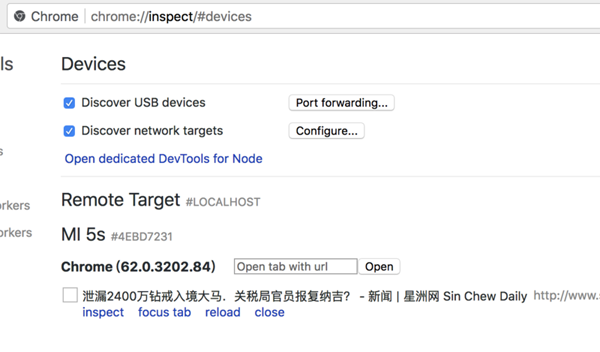
前面我们已经了解了 H5 页面加载过程，接下来如果遇到白屏，我们自然会问，怎么才能知道页面当前处在哪个阶段，每个阶段耗时多长，以及整体首屏加载的耗时呢？

首先看下通过 PC Chrome 模拟 H5 页面的情况。Chrome Devtool 提供的 Performance 工具，可以录制页面从第一个请求到加载完成的所有事件，通过这种方式可以很详细的看到各阶段做的事情和具体的耗时。

[](http://s1.51cto.com/oss/201808/29/bcdc1bee3c4e6b09754905401a85d694.png)

其中两个最关键的首屏耗时指标：domContentLoaded（首屏页面可见）和onLoad（首屏加载完成）的耗时，除了图示的方法，还可以通过在 console 里打印全局变量window.performance.timing，拿到时间戳并计算得到。

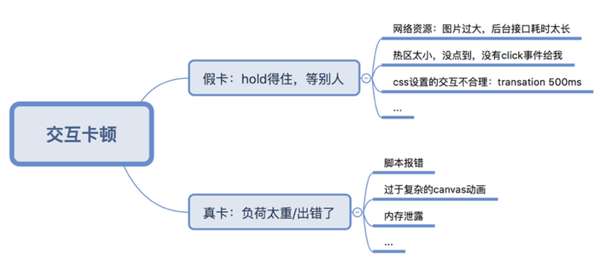
但实际我们要的是移动设备的真机数据，这个才能真实反应页面性能和用户体验。想要获取 H5 真机耗时，一种方式是 js 代码进行上报；另一种是对于 Android 设备，可以用 remote-debug 的方式远程调试真机页面。只需要保证 webview 调试开关打开 & 与 PC USB 连接且开启 USB 调试，就可以在 PC Chrome 访问 chrome://inspect 来获取调试对象。之后参考 PC Chrome 模拟 H5 的方法即可拿到数据。

[](http://s1.51cto.com/oss/201808/29/b47e39d3abe27144d23b90cad3a5a82a.png)

对于传统页面而言，实际分析发现大部分耗时还是在移动网络请求这部分，所以最直接有效的方式就是对页面进行直出改造，也就是改变先加载 html、再加载 css 等数据的情况，先在后端（比如 nodejs）并行加载首屏依赖的所有 css、js 和后台接口数据，拼装好一个完成的最终要呈现的 html 再回给前端，达到秒开的效果。

### 三、实例分析：卡慢问题

有时候用户在页面交互的过程中会遇到卡慢，比如上下滑动列表、左右切换或者轮播等。这个过程无非也是执行 js、请求资源、计算新的页面布局并渲染绘制这几件事。通过 Performance 分析就会发现，卡慢其实并不全是很多人认为的“移送设备性能就是差”，有时候其实是假性卡顿。

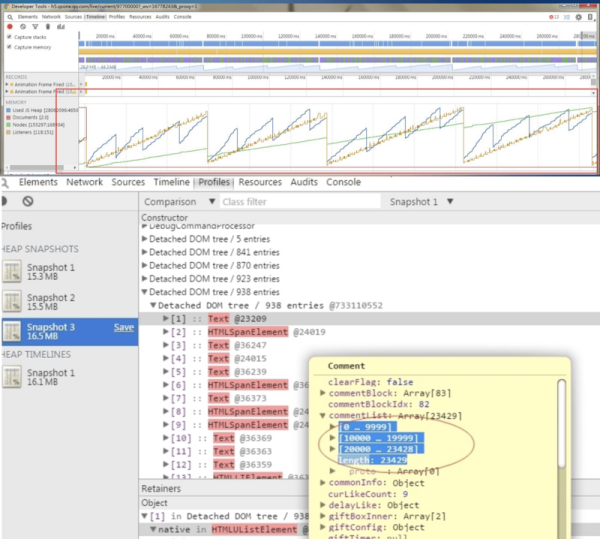
[](http://s4.51cto.com/oss/201808/29/756731f574850651df2763d606b3f02d.png)

比如下面这个就是热区过小的问题：

[](http://s1.51cto.com/oss/201808/29/812116dd035fc941dd6858450f2a5319.png)

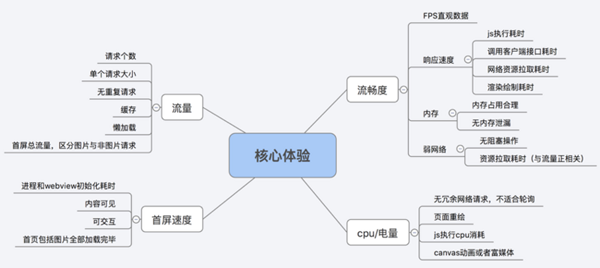
真卡的情况，往往脚本报错占了很大比重，直观表现就是页面是卡死，而不是变慢。其他的诸如内存问题，通常表现是页面越来越卡，因为使用时间越长，资源消耗越大。比如页面使用了比较复杂的 canvas 动画、比较耗性能的 iframe 元素，或者直播流媒体，这种情况下容易出现内存泄漏。

下面这个就是 dom 节点引发的内存泄漏,不使用的 commentList 列表没有释放，越积越多到长度几万个的时候开始卡顿。

[](http://s1.51cto.com/oss/201808/29/5cf993389845f0d97b9bd624e5e9d59f.png-wh_600x-s_2771299390.png)

### 四、总结：H5 前端性能测试方案

当然，前端性能不仅仅表现在白屏、卡顿问题，也有可能是手机过度发热等等。从用户核心体验出发，我们认为，H5 前端性能最重要的参考标准就是：要以最轻量的方式，给用户最好的体验。从这个方向出发，我们积累了一些测试经验，其中最重要的必过项是首屏速度（不仅提升用户体验，还可以提升业务的转化率），其次流畅度、流量和 CPU 等，某些场景下也是需要重点考量的点。

[](http://s3.51cto.com/oss/201808/29/0297fd4dd7f08d5b077c677193a84ee8.png)