ddtalk.github.io

H5性能优化方案 - DingTalk Developer Blog

H5性能优化意义

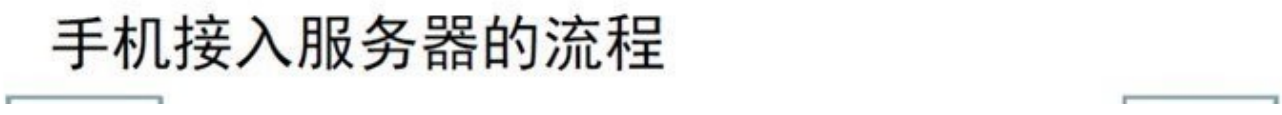
对于一个H5的产品，功能无疑很重要，但是性能同样是用户体验中不可或缺的一环。原本H5的渲染性能就不及native的app，如果不把性能优化做起来，将极大地影响用户使用产品的积极性。

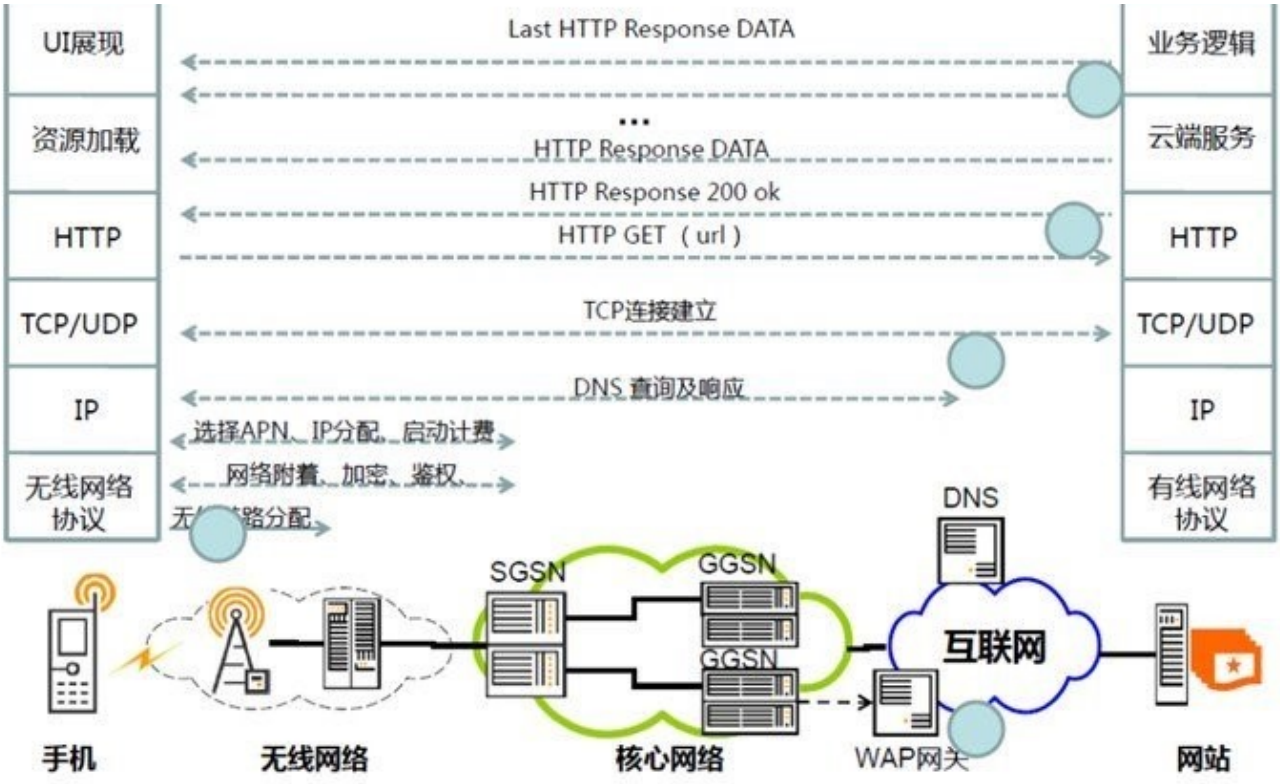
用户感受

当用户能够在1-2秒内打开H5页面，看到信息的展示，或者能够开始进行下一步的操作，用户会感觉速度还好，可以接受；而页面如果在2-5秒后才进入可用的状态，用户的耐心会逐渐丧失；而如果一个界面超过5秒甚至更久才能显示出来，这对用户来说基本是无法忍受的，也许有一部分用户会退出重新进入，但更多的用户会直接放弃使用。

一秒钟法则

移动互联网的架构、通讯机制，与有线网络有着巨大的差异，这也给H5的开发带来了很大的挑战。





这是一张手机端接入服务器的流程。

首先，手机要通过无线网络协议，从基站获得无线链路分配，才能跟网络进行通讯。 无线网络基站、基站控制器这方面，会给手机进行信号的分配，已完成手机连接和交互。 获得无线链路后，会进行网络附着、加密、鉴权，核心网络会检查你是不是可以连接在这个网络上，是否开通套餐，是不是漫游等。核心网络有SGSN和GGSN，在这一步完成无线网络协议和有线以太网的协议转换。 再下一步，核心网络会给你进行APN选择、IP分配、启动计费。 再往下面，才是传统网络的步骤：DNS查询、响应，建立TCP链接，HTTP GET，RTTP RESPONSE 200 OK，HTTP RESPONSE DATA，LAST HTTP RESPONSE DATA，开始UI展现。

可见，通过运营商的网络上网，情况比较复杂，经过的节点太多；运营商的网络信号强度变化频繁，连接状态切换快；网络延

迟高、丢包率高；网络建立连接的代价高，传输速度快慢不等（从2G到4G，相差很大）。

而我们优化的目标，就是所谓的一秒钟法则，即达成以下的标准：

- 2g网络：1秒内完成dns查询、和后台服务器建立连接
- 3g网络：1秒内完成首字显示（首字时间）
- wifi网络：1秒内完成首屏显示（首屏时间）

优化方案

资源加载

首屏加载

用户从点击按钮开始载入网页，在他的感知中，什么时候是“加载完成”？是首屏加载，即在可见的屏幕范围内，内容展现完全，loading进度条消失。因此在H5性能优化中，一个很重要的目的就是尽可能提升这个“首屏加载”的时间，让它满足“一秒钟法则”。

按需加载

首先要明确，按需加载虽然能提升首屏加载的速度，但是可能带来更多的界面重绘，影响渲染性能，因此要评估具体的业务场景

再做决定。

Lazyload

Lazyload，即延迟加载，这并不是一个新的技术，在PC时代也是非常常用的一种性能优化手段。这个方案的原则是让屏幕外，或者不影响整体效果显示的图片、背景等资源，在界面就绪之后再进行网络加载。

滚屏加载

滚屏加载是一种常见的无刷新动态加载数据的方案，通常用在列表形式数据展示中。一方面，数据不是通过翻页进行加载，这样就避免了再一次请求和渲染整个页面；另一方面，数据显示的数量是受限的，例如第一次只请求了10条数据，也就只需要渲染这10条数据，下拉滚屏的时候，再去获得下面10条数据。

Media Query（响应式加载）

响应式设计是现在网站设计的一个流行趋势，随着移动互联网的发展，这项技术也越来越受到重视。通过这项技术，我们能够方便地控制资源的加载与显示，例如说在分辨率不同的手机上，分别使用不同的css，加载不同大小的图片资源。 方案参考：http://www.poluoluo.com/jzxy/201206/167034.html

第三方资源异步加载

第三方资源有的时候不可控，比如说页面统计、地图显示、分享组件等，这些第三方资源使用的时候要慎重选择，充分考察它们对于性能的影响，使用异步加载的方式进行，防止第三方资源的使用影响到页面本身的功能。

Loading进度条

在加载时间较长的时候，务必要让用户明确感知到加载完成的提示，通常是在加载过程中显示Loading的进度条，加载完成的时候隐藏它。从心理上，这会让用户有一种“期盼感”，而不至于太过枯燥。

对于一些重量级的H5应用，例如游戏，开始前需要加载很多资源才能让后面的游戏过程更为流畅，一个带百分比进度显示的进度条就更加重要。

避免30*/40*/50*的http status

- 200是一个正常的response，我们在浏览器中打开一个网页（后面会讲如何针对移动端进行调试），还会看到304，即命中浏览器缓存。这两种状态是正常的http status。

- 302、301跳转是常见的跳转，尤其前一种，在我们进行鉴权的时候有时会用到，但这个做法要尽可能地优化，一个页面访问，最多只进行一次302跳转即可，切忌频繁地跳转。

- 404、500，我们对自己开发的代码比较注意，一般不会发生，但是有的时候，加载第三方库，尤其是第三方库中有自己

load组件的操作，这时，404和500错误可能会在你不知不觉的时候发生。例如钉钉的第三方微应用中，就遇到过dojo的组件加载问题，加载的一些子组件失败了，但是又没有影响页面显示，这就很容易被忽略。后面也会再讲，如何去测试和发现这样的隐患。

Favicon.ico

如果我们没有设置图标ico，则会加载默认的图标：域名目录下的favicon.ico。很多开发者没有注意到这一点，就会导致这个请求404或者500。

通常，我们在应用内部打开网页，不会显示这个图标出来（除非放到浏览器中显示网页），我们需要保证这个图标存在，尽可能地小（一般4KB以下），并且设置一个较长的缓存过期时间。

图片的使用

格式选择

显示效果较好的图片格式中，有webp、jpg和png24/32这几种常见的图片格式。一般来说，webp的图片最小，但在iOS或者android4.0以下的系统中可能会有兼容性问题需要解决。

- Jpg是我们最常使用的方案，大小适中，解码速度快，兼容性问题也基本不存在，是我们在H5的应用中使用起来性价比最高的方案。

- Png24或png32，一般来说，显示效果肯定会比jpg更好，但是实际上人眼很难感知出来，所以在H5应用中要避免这种格式的大图片。

对于少量的图片，推荐用智图或者tinypng等工具来帮助自己选择合适的大小、格式。

像素控制

在H5应用中，图片的像素要严格控制，一般来说不建议宽度超过640px。

小图片合并

在html网页中，如果有多个小图片需要加载，不妨试试CSS Sprites方案，尤其是一些基本不变，大小差不多的操作类型图标。

避免html代码中的大小重设

在html或者css中，如果有类似width: **px这样的代码，就要注意看一看了，如果说图片显示的效果是宽度100px，而下载的图片却是200px宽度，那这大小基本上就是所需要的4倍面积了，所以在H5应用中，使用图片的一个原则就是需要显示成多大，就下载多大的资源。

避免DataURL

DataURL是用Base64的方式，将图片变成一串文本编码放入代码的方式。这种方式有好处，因为它可以减少一次http交互的请求，对于一些体积特别小的图片，或者是动态生成的图片可以考虑使用。但在H5应用中，一般情况下，我们都是需要避免DataURL的，因为它的数据体积通常比二进制图片的格式大1/3，而且它不会被浏览器缓存，每次页面刷新都需要重新加载这部分数据。

使用图片的替代(css3, svg, iconfont)

CSS3和svg可以更好地使用GPU进行渲染加速，而且会避免增加图片资源导致的http请求增加。例如一些div的圆角效果，就完全可以用用css来实现。

Iconfont，可以认为是一种矢量类型的操作字体。如果页面中有较多的操作图标，可以考虑使用iconfont来替代图片资源。

域名/服务端部署

Gzip

服务端要开启Gzip压缩。

资源缓存，长cache

合理设置资源的过期时间，尤其对一些静态的不需要改变的资源，将其缓存过期时间设置得长一些。

分域名部署(静态资源域名)

将动态资源和静态资源放置在不同的域名下，例如图片，放在自己特定的域名下。这样的好处是，静态资源请求时，不会带上动态域名中所设置的cookie头信息，从而减少http请求的大小。

减少Cookie

尽量减少Cookie头信息的大小，因为这部分数据使用的是上行流量，上行带宽更小，所以传输速度更慢，因此要尽量精简其大小。

CDN加速

部署CDN服务器，或者使用第三方的CDN加速服务，优化不同地域接入网站的带宽速度。

代码资源

Javascript, CSS合并

尽量将所有的js和css合并，减少资源请求的次数。

外联使用js, css

外联使用js和css，这样可以有效地利用缓存，避免html页面刷新后重新加载这部分代码。

压缩html, js, css

压缩代码，尤其是js和css资源，压缩后的大小可以降低至原来的1/3以下，有效节约流量。

资源的版本更新

库js、css通常不会更新，但是我们的业务js和css可能会有更新，如果命中浏览器缓存，可能会让一些新的特性不能及时展现，甚至可能导致逻辑上的冲突。

因此对于这些js、css的资源引入，最好用版本号或者更新时间来作为后缀，这样的话，后缀不变，命中缓存；后缀改变，浏览器自动更新最新的代码。

Css位置

CSS要放到html代码的开头的head标签结束前。如果网页是动态生成的，那么在head代码完成后可以强制输出（例如php的flush()操作），这样的话，浏览器就会更快地解析出来head中的内容，开始下载css文件资源。

Js位置

Js放到</body>前，这样的话，js的加载不会影响初始页面的渲染。

代码规范

避免空src

图片设置为空的src地址，在某些浏览器中可能会导致增加一个无效的http请求，因此要避免。

避免css表达式

可能会让页面多次执行计算，造成卡顿等性能问题。

避免空css规则

降低css渲染计算的成本

避免直接设置元素style

直接设置style属性，一方面在html代码中不利于缓存，另一方面也不利于样式的复用，因此要避免，通过指定id或者class的方式，在css代码块中进行样式调整。

服务端接口

接口合并

如果页面需要请求两部分以上的数据接口，建议将其合并，否则会增加一次http请求。

减少接口数据量

有的时候，服务端会把一些无关紧要的数据返回回来，尤其是类似于更新时间、状态等信息，如果在客户端不影响内容的逻辑展示，不妨在接口返回的数据中直接去掉这些内容。

缓存

缓存接口数据，在一些数据新旧敏感性不高的场景下很有作用，在非首次加载数据时候优先使用上次请求来的缓存数据，可以让页面更加快速地渲染出来，而不用等待一个新的http请求结束之后再渲染。这一点我们在后面还会再次提及。

其他一些建议

- 合理使用css
- 正确使用Display属性 Display属性会影响页面的渲染，因此请合理使用
  - display:inline后不应该再使用width、height、margin、padding以及float
  - display:inline-block后不应该再使用float
  - display:block后不应该再使用vertical-align
  - display:table-*后不应该再使用margin或者float
- 不滥用float

- 不声明过多的font-size
- 值为0时不需要单位
- 标准化各种浏览器前缀
  - 无前缀应放在最后
    - CSS动画只用（-webkit- 无前缀）两种即可
    - 其它前缀为 -webkit- -moz- -ms- 无前缀 四种，（-o-Opera浏览器改用blink内核，所以淘汰）
- 选择器
- 避免让选择符看起来像是正则表达式。高级选择器不容易读懂，执行耗时也长
- 尽量使用ID选择器
- 尽量使用css3动画
- 资源加载
- 使用srcset
- 首次加载不超过1024KB（或者可以说是越小越好）
- html和js

- 减少重绘和回流
- 缓存dom选择和计算
- 缓存列表.length
- 尽量使用事件代理,避免批量绑定事件
- 使用touchstart，touchend代替click
- Html使用viewport
- 减少dom节点
- 合理使用requestAnimationFrame动画代替setTimeOut
- 适当使用Canvas动画
- TouchMove, Scroll事件会导致多次渲染

更快一步

前面的很多建议与普通的PC端web网页的开发是一致的，但是在移动互联网应用下，仅仅做到这些，可能只有60分，那么怎样才能得到80分甚至更高？

单页应用

钉钉的审批微应用，使用的就是单页架构。在这种架构下，基本

不存在页面跳转的等待时间，只需要执行js逻辑触发界面变化，最多进行一次网络请求，获得服务端数据，其他资源均不需要再次请求。

资源离线

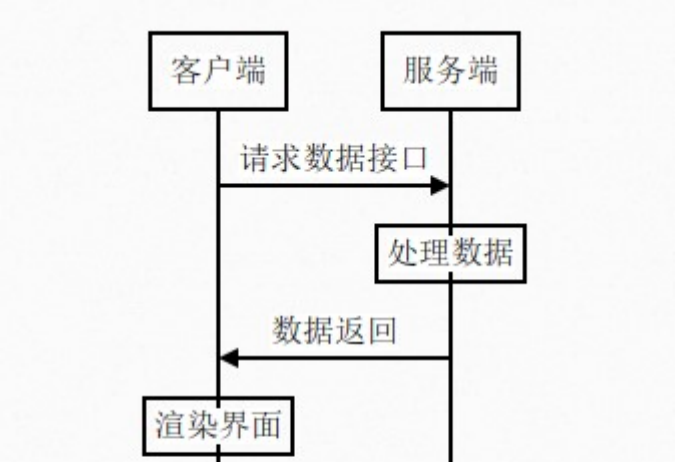
再快的网络交互，毕竟也是跨越了数个网络节点，因此一张图片、一个js，优化到了极致，也照样可能需要几百毫秒的时间来获得。因此想要打破这个极限，就要使用资源离线的策略。

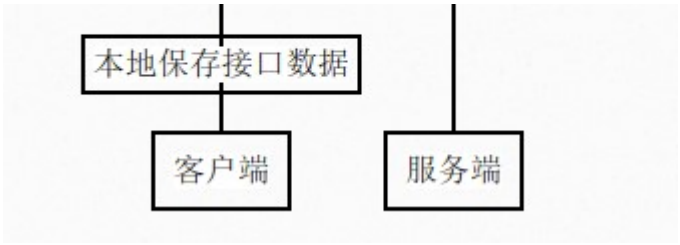
在钉钉的微应用中，就使用了这样的一个“离线包”策略。一些固定的图片、js库等，被打包放入app中（或根据需要，在app启动的时候进行下载更新）。

微应用中，网页代码里面加载网络资源的需求，就变成了直接加载本地文件，速度自然得到再一次巨大的提升。

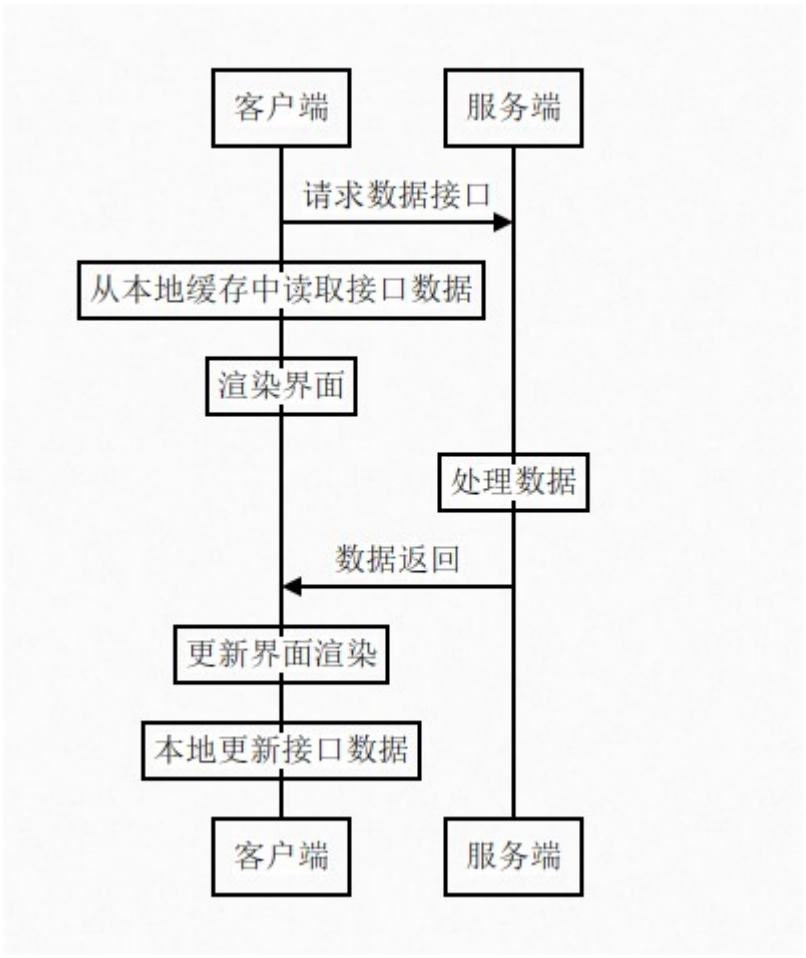
本地数据持久化和更新机制(版本管理)

对于一些时效性没有那么高的数据，可以考虑将接口数据缓存。那么页面的渲染将变成这样的流程：





而非首次进入界面，流程如下：



可以看出，在非首次进入界面的时候，页面不需要等待网络数据返回，就可以进行界面渲染，渲染的初始数据来自于本地的缓存，页面可以“秒开”。而当服务端的数据返回之后，本地的渲染会再次更新，缓存也被更新。

采用这样的方案有利有弊，好处显而易见，首屏加载的速度简直太快了——静态资源来自本地，数据接口来自本地，这在2G、3G或者其他网络速度较慢的时候，也可以让用户在极短的时间内

就看到内容。但是这种方案也并非万能。

1. 首次加载不可避免，还是会请求网络。

2. 服务端有更新的时候，客户端不能够快速感知，页面可能还停留在一个“旧的版本”上，尤其是网络速度较慢时，可能还是需要经过好几秒，页面才会更新至最新版本。因此如果应用对数据的新旧很敏感的话，这种方案就不适合

3. 数据更新后，需要重新渲染界面，界面刷新的性能消耗比正常情况更多，而且增加了程序的复杂度，容易出错。

预加载

有时，我们能够通过用户的行为统计，预判出用户下一步可能进行的操作。假设，我们统计出来针对某个微应用，用户首页渲染完成之后，大部分会点击列表中的第一个项目查看详情。那么在首页渲染完成之后，我们就可以先预先加载第一个项目的部分内容，那么针对这部分用户，他们实际点击之后，立即就能看到新的页面中的内容。

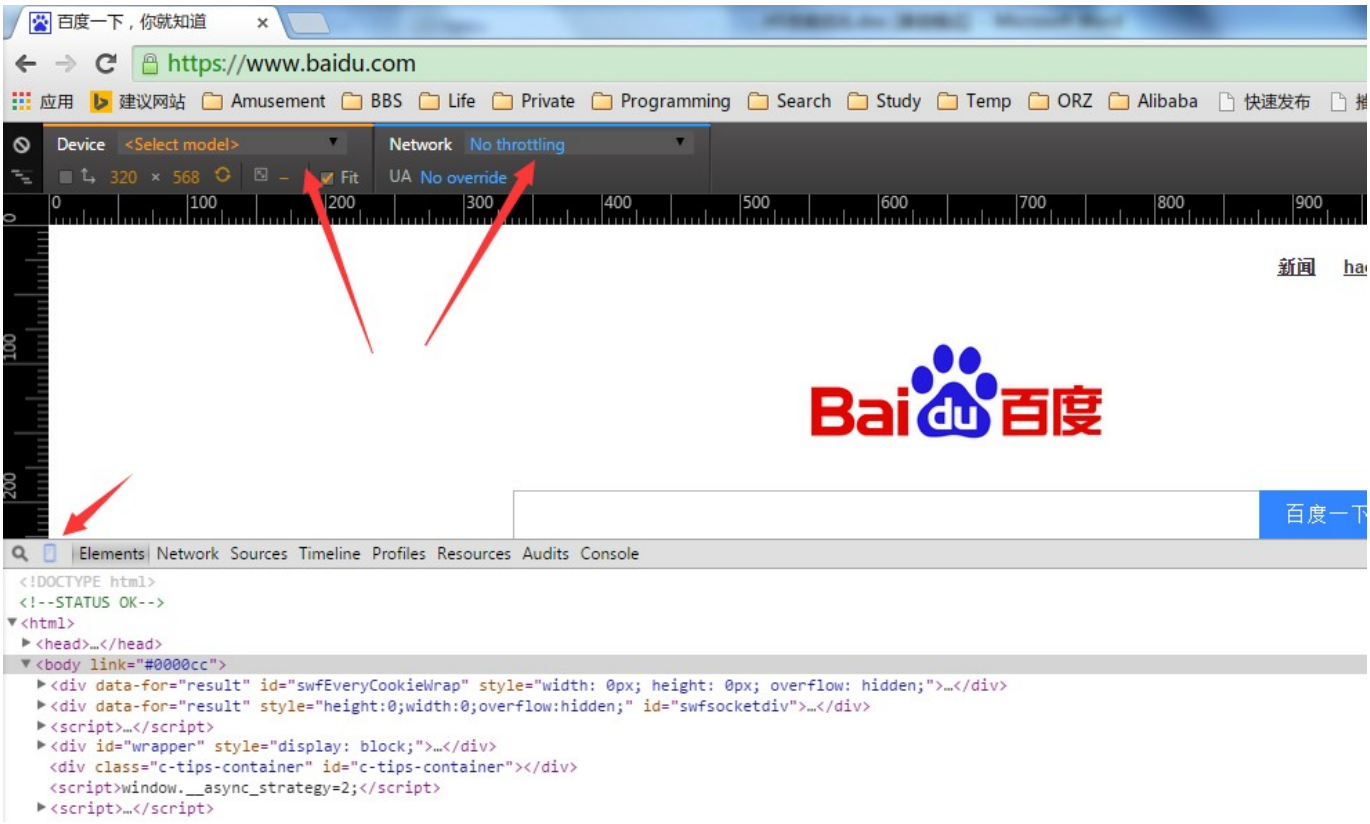
当然，这个方式也并不是在所有场景下都使用。一方面，需要做好充分的用户调研，掌握用户的使用习惯；另一方面，对于小部分用户而言，预加载所带来的就是不必要的流量消耗。

测试方案

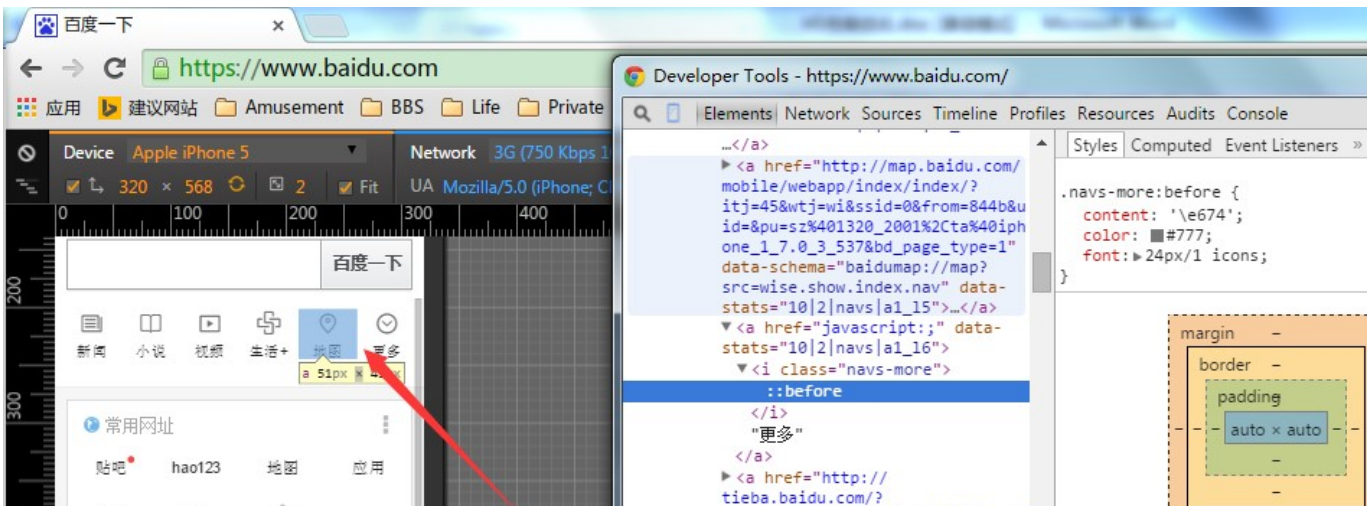
工具准备

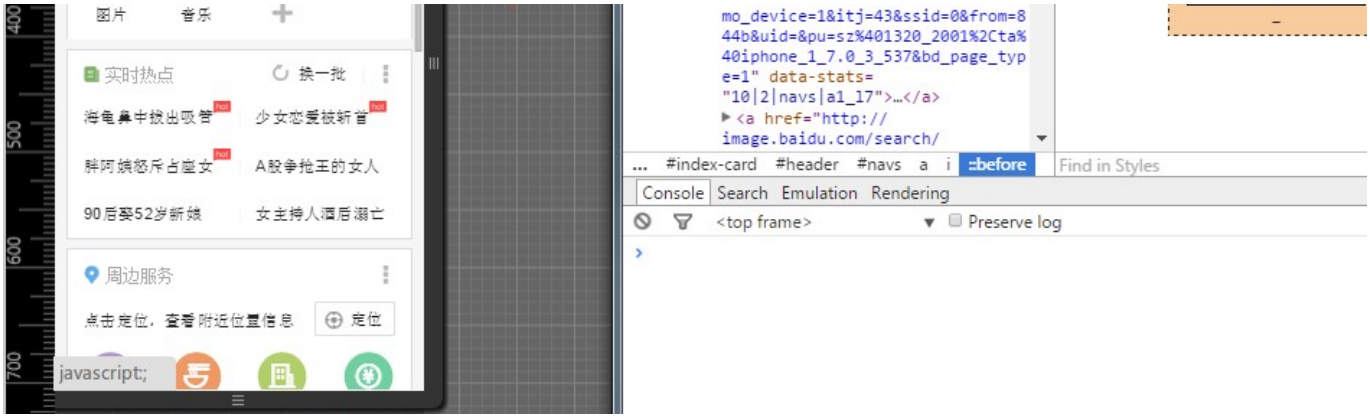
Chrome

在功能测试中，我们通常可以用chrome来测试不同的分辨率下，或是不同的设备上，网页的展现情况。在我们做性能优化的时候，也可以用这种方式来进行调试，方便地观察在特定设备上，静态资源是否按照我们想象的那样去加载了。



例如，我们想看下百度首页在某个设备下的表现。 通过F12进入控制台，点击图中的短箭头标示出来的那个设备图标，然后就可以在Device和Network中选择不同的设备和网络状况。

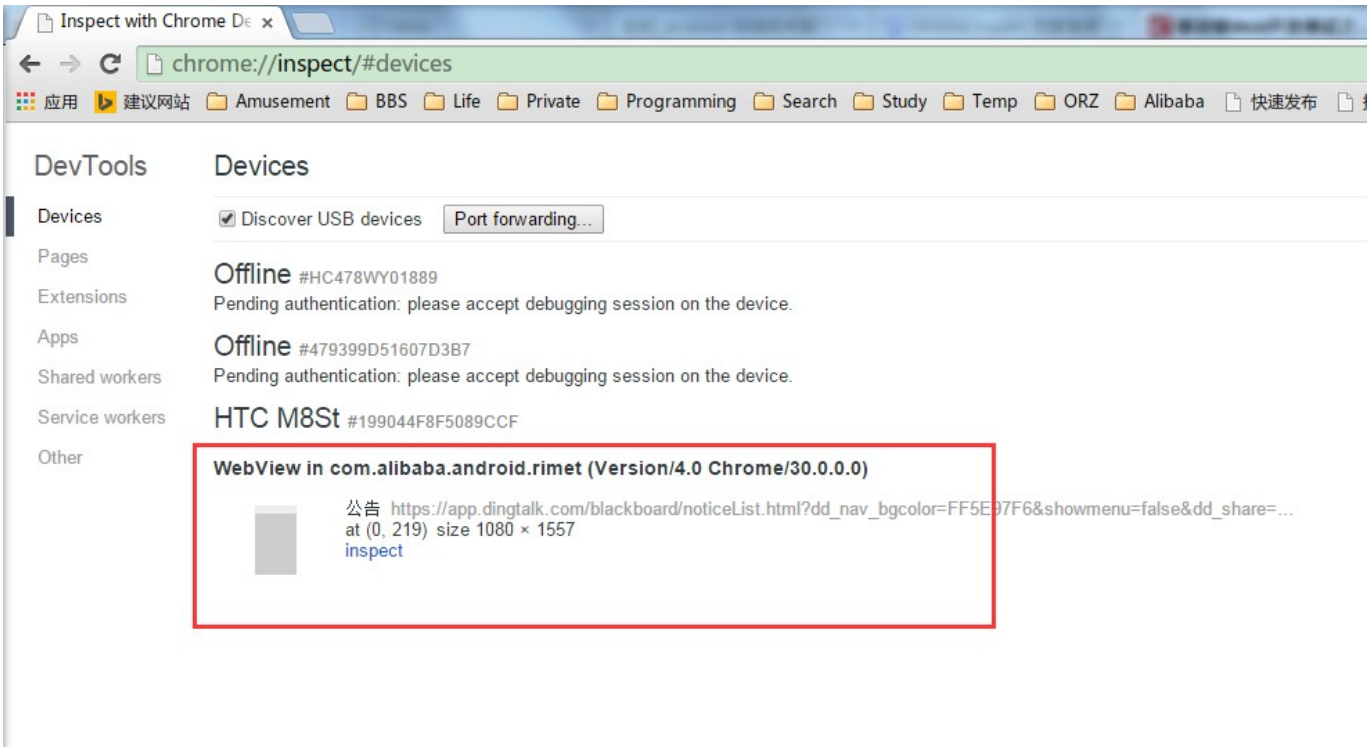




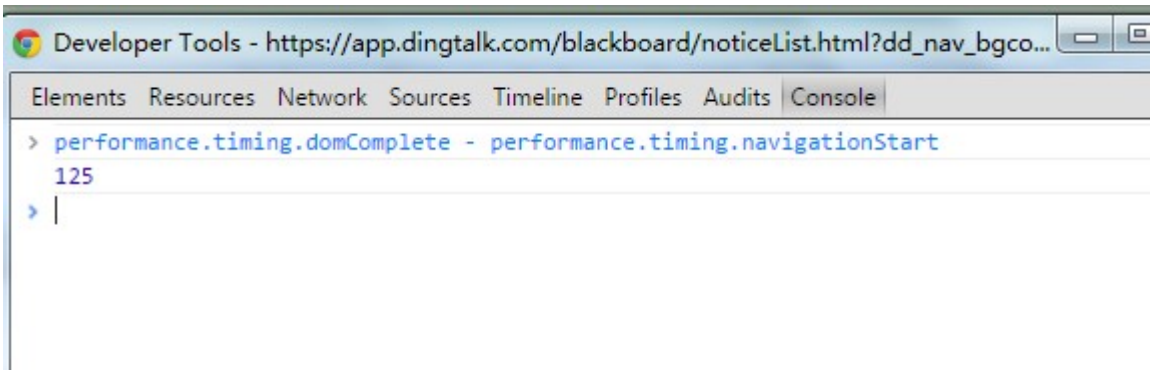
例如iphone5下，这个地图的图标，明显就可以看到是用iconfont来实现的效果。

当然，这个功能也仅仅是一种模拟，通过控制屏幕分辨率、UserAgent等来模拟设备请求，在实际的设备上，又该怎么查看呢？

还是Chrome，我们在地址栏中输入chrome://inspect （注意：Android版本需要是4.4+，并且应用中的WebView必须进行相应的调试声明配置）



在这里点击inspect，则可进入我们熟悉的F12控制台界面，只不过debug的对象变成了我们在手机应用中的网页。



输入performance.timing.domComplete - performance.timing.navigationStart，就可以打印出网页加载的时间，domComplete表示所有的处理都已完成并且所有的附属资源都已经下载完毕。navigationStart表示开始加载新页面。两者相减，就代表这个网页完成渲染所需要的时间了。

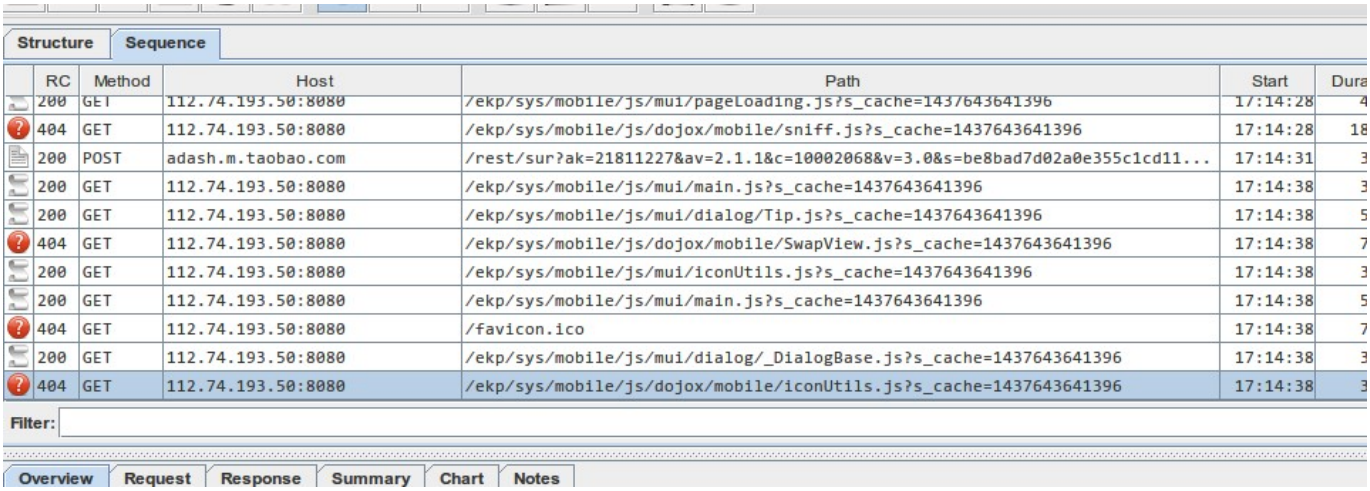
同样地，我们可以在Elements tab中，debug网页，查看各个资源的使用，在Network中，看看加载了哪些资源，是否都做过了压缩。

然而，这种方式，还是有一定缺陷。 1. 如果打开网页经过了跳转，那么我们只能在这里看到最后一个url页面的加载情况。 2. 第一次打开页面的时候，在Network中，默认是不显示请求的详情的，当我们选择了preserve log upon navigation之后才会捕获，因此首次进入页面的加载情况，我们就很难获得了。

那么有没有一种方法，让我们能够更方便地去查看首次访问时，各种资源的加载使用情况呢？

Charles

Charles Proxy，可以说是H5测试的一个神器。 它的作用是在PC端开启一个代理服务器，手机连到这个代理服务器上之后，所有的http请求就都可以在这里看得清清楚楚。经过配置证书选项，https的请求也可以正常查看。



从图中，我们明显可以看到，有一些404的异常请求，这些都将对我们H5应用的性能造成影响。 如果我们发现有一些资源的Duration比较大，那这些可能是服务端响应太慢，自然也可以作为我们优化的依据。

测试标准

在钉钉的测试中，形成了一套标准。

指标项	遵循的原则	优先级	检查项	说明
内存	内存无泄漏	P0	主功能页面反复打开，功能的重复调用，内存无泄漏	可以使用sysdump,也可以用我们开发的perfeasy工具

指标项	遵循的原则	优先级	检查项	说明
				进行观察
		P1	主功能页面，持续操作，退出后，内存占用不超过总内存的5%	perfeasy
CPU	减少无端的CPU使用	P1	灭屏，静置2分钟，在5分钟内CPU使用平均1%	adb连接后, 使用top命令
			主干功能正常操作CPU占用不超过60%, 持续5秒	perfeasy
电量	避免无端电量消耗	P0	灭屏状态下，无线程持续运行	一般来说, 静置cpu正常, 这一项就没有问题
			灭屏，window.setTimeout()方法停止	一般来说, 静置cpu正常, 这一项就没有问题
			灭屏，window.setInterval()方法停止	同上
			ajax超时时间设置为5000ms以内	同上
			ajax无retry逻辑	结合代码
				结合代码

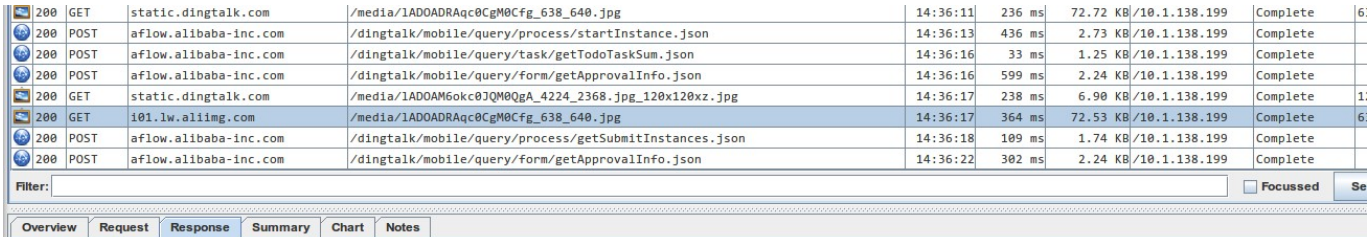
指标项	遵循的原则	优先级	检查项	说明
资源	资源的正确使用	P0	是否存在资源的重复拉取	charles
		P1	H5页面首屏总大小不超过200K	charles, chrome
		P1	抓包检查(js/css/html)代码去除了空格/注释，JS文件变量名变成a/b等代替	charles, chrome
		P1	H5引用的单张图片小于60K	charles, chrome 如果影响了显示质量, 可酌情调整
流畅度	确保给到用户流畅的展示体验	P1	流畅的实时动画展示，avgFPS>=45	perfeasy
时延	确保给到用户流畅的切换体	P0	wifi网络下，首次进入页面onload时间<1000ms	Chrome

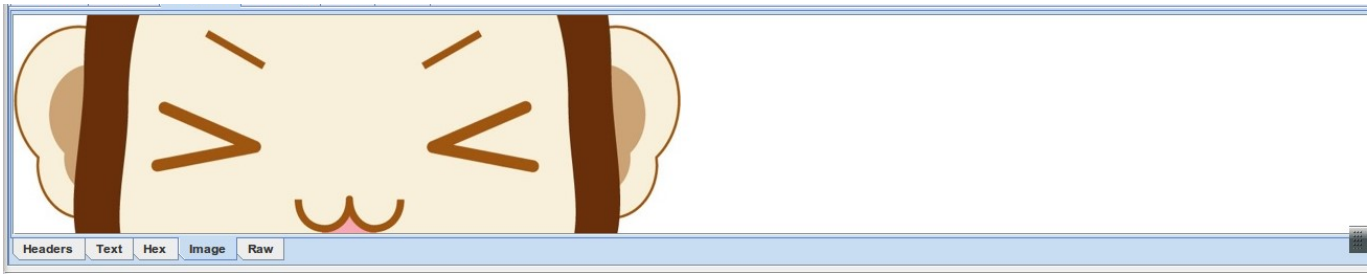
指标项	遵循的原则	优先级	检查项	说明
时延	验 确保给到用户流畅的切换体验	P0	wifi网络下，首次进入页面onload时间<1000ms	Chrome
			wifi网络下，非首次进入页面onload时间<500ms	Chrome
			3G正常网络， 首次进入页面onload时间<2000ms	chrome, 树莓派模拟3G
			3G正常网络， 非首次进入页面onload时间<1000ms	chrome, 树莓派模拟3G

经典案例

图片未优化

通过charles可以方便地进行测试。 从请求监控的情况看，有一张图片超过了60KB，宽度640px，但是在应用中，实际显示的是一张小缩略图，是通过代码控制让它显示成小图的，因此修改方案很简单，将所有头像的图片均改为获取120px宽度的即可。





按需加载

- 钉钉的教学页面
- 多个slide页面, 每个页面有2-3个图片, 其中有一个是大图显示
- 图片是客户提供的, 最大的图片大约是300KB以上
- 第一次进入页面后, 所有slide的图片均进行加载
- 3G网络下, loading的图标大约持续6000ms后才会消失
- 优化方案
- 尽可能优化图片, 但是压缩后发现噪点明显增加, 影响了显示效果
- 修改加载方案, 第一次进入后, 只加载本页的图片
- loading时间降低至1秒左右
