阅读目录

引言

前阵子在w3ctech的走进名企 - 百度前端 FEX 专场上曾“夸下海口”说听完讲座后七天就可以打造自己的前端性能监控系统，既然说出去了也不能食言。从前一篇文章前端数据之美相信大家对前端数据有了一定的了解，下面就针对其中的性能数据及其监控进行详细阐述。

开始行动

本文中的性能主要指 web 页面加载性能，对性能还不了解？不用担心，接下来的“每一天”跟我一起进入前端性能的世界。

Day 1 为什么要监控性能?

> “If you cannot measure it, you cannot improve it” ———— William Thomson

这是一个最基本的问题，为什么要关注和监控前端性能？对于公司来说，性能在一定程度上与利益直接相关。国外有很多这方面的调研数据：

性能	收益
Google 延迟 **400ms**	搜索量下降 **0.59%**
Bing 延迟 **2s**	收入下降 **4.3%**
Yahoo 延迟 **400ms**	流量下降 **5-9%**
Mozilla 页面打开减少 **2.2s**	下载量提升 **15.4%**
Netflix 开启 Gzip	性能提升 13.25% 带宽减少**50%**

数据来源：http://www.slideshare.net/bitcurrent/impact-of-web-latency-on-conversion-rates http://stevesouders.com/docs/jsdayit-20110511.pptx

为什么性能会影响公司的收益呢？根本原因还是在于**性能影响了用户体验。**加载的延迟、操作的卡顿等都会影响用户的使用体验。尤其是移动端，用户对页面响应延迟和连接中断的容忍度很低。想象一下你拿着手机打开一个网页想看到某个信息却加载半天的心情，你很可能选择直接离开换一个网页。谷歌也将页面加载速度作为 SEO 的一个权重，页面加载速度对用户体验和 SEO 的影响的调研有很多。

尽管性能很重要，开发迭代过程中难免会有所忽视，**性能会伴随产品的迭代而有所衰减。**特别在移动端，网络一直是一个很大的瓶颈，而页面却越来越大，功能越来越复杂。并没有简单的几条黄金规则就可以搞定性能优化工作，我们需要一套性能监控系统持续监控、评估、预警页面性能状况、发现瓶颈，指导优化工作的进行。

Day 2 有什么可用的工具?

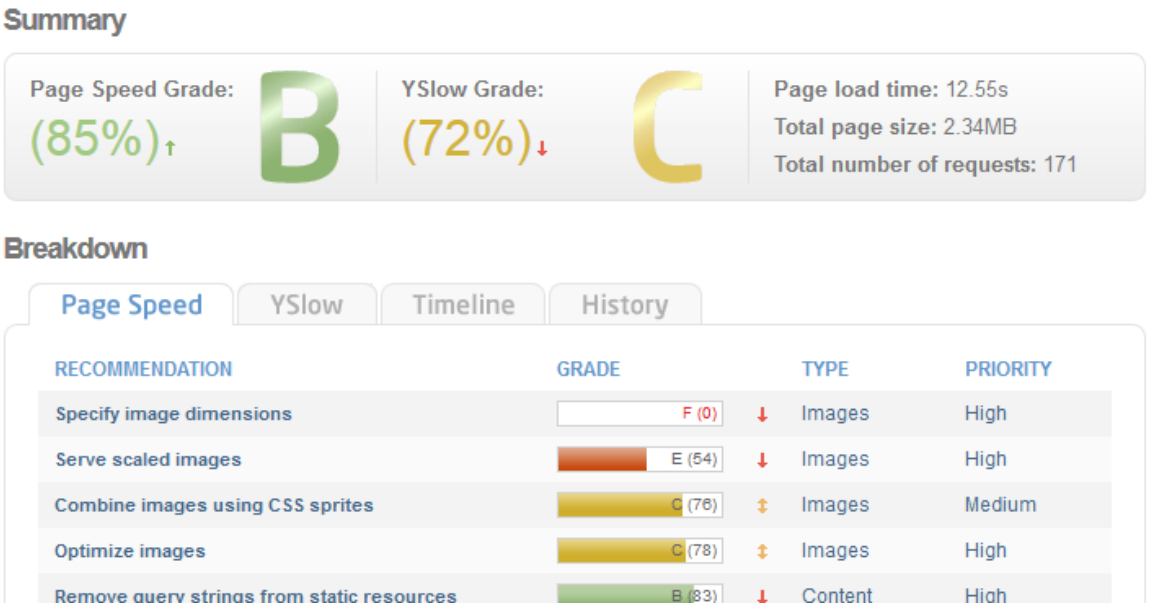
> 工欲善其事必先利其器

页面性能的评估与监控有很多成熟优秀的工具，合理利用已有工具能达到事半功倍的效果。下面简单介绍几个常用的工具：

回到顶部

Page Speed

Page Speed 是谷歌开发的分析和优化网页的工具，可以作为浏览器插件使用。工具基于一系列优化规则对网站进行检测，对于未通过的规则会给出详细的建议。与此类似的工具还有 Yslow 等，推荐使用gtmetrix网站同时查看多个分析工具的结果，如下图所示：



回到顶部

WebPagetest

WebPageTest 是一款非常优秀的网页前端性能测试工具,已开源。可以使用在线版，也可以自己搭建。国内也有利用 WebPagetest 搭建的性能测试平台，推荐使用阿里测 (以下示例使用阿里测进行测试)。

使用 WebPagetest，你可以详细掌握网站加载过程中的瀑布流、性能得分、元素分布、视图分析等数据。其中比较直观的视图分析功能可以直接看到页面加载各个阶段的截屏:

注: 整个测试结果请点击此处

上图直观地展现了浏览类网站两个重要的时间点：白屏时间和首屏时间，即用户多久能在页面中看到内容，以及多久首屏渲染完成(包含图片等元素加载完成)。这两个时间点直接决定了用户需要等待多久才能看到自己想看到的信息。谷歌优化建议中也提到减少非首屏使用的 css 及 JS,尽快让首屏呈现。

回到顶部

PhantomJS

PhantomJS轻松地将监控带入了自动化的行列。Phantom JS 是一个服务器端的 JavaScript API 的 WebKit，基于它可以轻松实现 web 自动化测试。PhantomJS 需要一定编程工作，但也更灵活。官方文档中已经有一个完整的获取网页加载 har 文件的示例，具体说明可以查看此文档，国内也有不少关于此工具的介绍。另外新浪@猿吃馍香开发的类似工具**berserkJS**也挺不错，还贴心的提供了**首屏统计**的功能，具体文章可以查看此处。

Day 3 开始线上真实用户性能监控

> 取其所长，避其所短

到此肯定有同学问，既然有这么多优秀的工具，为什么要监控线上用户真实访问性能呢?

我们发现，工具模拟测试会在一定程度上与**真实情况偏离**，有时无法反映性能的波动情况。另外除了白屏首屏之类的基础指标，产品线同样关注**产品相关的指标**，例如广告可见、搜索可用、签到可用等，这些功能直接与页面 JS 加载相关,通过工具较难模拟。

为了持续监控不同网络环境下用户访问情况与页面各功能可用状况，我们选择在页面中植入 JS 来监控线上真实用户访问性能，同时利用已有的分析工具作为辅助，形成一套完整多元的数据监控体系，为产品线的评估与优化提供可靠的数据。

关于不同监控方式的简单对比可以查看下表：

类型	优点	缺点	示例
非侵入式	指标齐全、客户端主动监测、竞品监控	无法知道性能影响用户数、采样少容易失真、无法监控复杂应用与细分功能	Pagespeed、PhantomJS、UAQ
侵入式	真实海量用户数据、能监控复杂应用与业务功能、用户点击与区域渲染	需插入脚本统计、网络指标不全、无法监控竞品	DP 、Google 统计

Day 4 如何采集性能数据?

> 监控用户的痛点

线上监控哪些指标呢？如何更好地反映用户感知?

对于用户来说他感觉到的为什么页面打不开、为什么按钮点击不了、为什么图片显示这么慢。而对于工程师来说，可能关注的是 DNS 查询、TCP 连接、服务响应等浏览器加载过程指标。我们根据用户的痛点，将浏览器加载过程抽取出四个关键指标，即白屏时间、首屏时间、用户可操作、总下载时间(定义可见上篇文章)。这些指标是如何统计的呢?

回到顶部

确定统计起点

我们需要在用户输入 URL 或者点击链接的时候就开始统计，因为这样才能衡量用户的等待时间。如果你的用户高端浏览器占比很高，那么可以直接使用Navigation Timing接口来获取统计起点以及加载过程中的各个阶段耗时。另外也可以通过 cookie 记录时间戳的方式来统计，需要注意的是 Cookie 方式只能统计到站内跳转的数据。

回到顶部

统计白屏时间

白屏时间是用户首次看到内容的时间，也叫做首次渲染时间，chrome 高版本有 firstPaintTime 接口来获取这个耗时，但大部分浏览器并不支持，必须想其他办法来监测。仔细观察 WebPagetest 视图分析发现，白屏时间出现在**头部外链资源加载完**附近，因为浏览器只有加载并解析完头部资源才会真正渲染页面。基于此我们可以通过获取头部资源加载完的时刻来近似统计白屏时间。尽管并不精确，但却考虑了影响白屏的主要因素：首字节时间和头部资源加载时间。

如何统计头部资源加载呢？我们发现头部内嵌的 JS 通常需等待前面的 JS\CSS 加载完才会执行，是不是可以在浏览器 head 内底部加一句 JS 统计头部资源加载结束点呢？可以通过一个简单的示例进行测试：

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="UTF-8"/>
    <script>
      var start_time = +new Date; //测试时间起点，实际统计起点为 DNS 查询
    </script>
    <!-- 3s 后这个 js 才会返回 -->
    <script src="script.php"></script>
    <script>
      var end_time = +new Date; //时间终点
      var headtime = end_time - start_time; //头部资源加载时间
      console.log(headtime);
    </script>
    </head>
    <body>
    <p>在头部资源加载完之前页面将是白屏</p>
    <p>script.php 被模拟设置 3s 后返回，head 底部内嵌 JS 等待前面 js 返回后才执行</p>
    <p>script.php 替换成一个执行长时间循环的 js 效果也一样</p>
    </body>
</html>
```

经测试发现，统计的头部加载时间正好跟头部资源下载时间相近，而且换成一个执行时间很长的 JS 也会等到 JS 执行完才统计。说明此方法是可行的(具体原因可查看浏览器渲染原理及 JS 单线程相关介绍)。

回到顶部

统计首屏时间

首屏时间的统计比较复杂，因为涉及图片等多种元素及异步渲染等方式。观察加载视图可发现，影响首屏的主要因素的图片的加载。通过统计首屏内图片的加载时间便可以获取首屏渲染完成的时间。统计流程如下：

`首屏位置调用 API 开始统计 -> 绑定首屏内所有图片的 load 事件 -> 页面加载完后判断图片是否在首屏内，找出加载最慢的一张 -> 首屏时间`

这是同步加载情况下的简单统计逻辑，另外需要注意的几点：

- 页面存在 iframe 的情况下也需要判断加载时间
- gif 图片在 IE 上可能重复触发 load 事件需排除
- 异步渲染的情况下应在异步获取数据插入之后再计算首屏
- css 重要背景图片可以通过 JS 请求图片 url 来统计(浏览器不会重复加载)
- 没有图片则以统计 JS 执行时间为首屏，即认为文字出现时间

回到顶部

统计用户可操作和总下载

用户可操作默认可以统计**domready**时间，因为通常会在这时候绑定事件操作。对于使用了模块化异步加载的 JS 可以在代码中去主动标记重要 JS 的加载时间，这也是产品指标的统计方式。

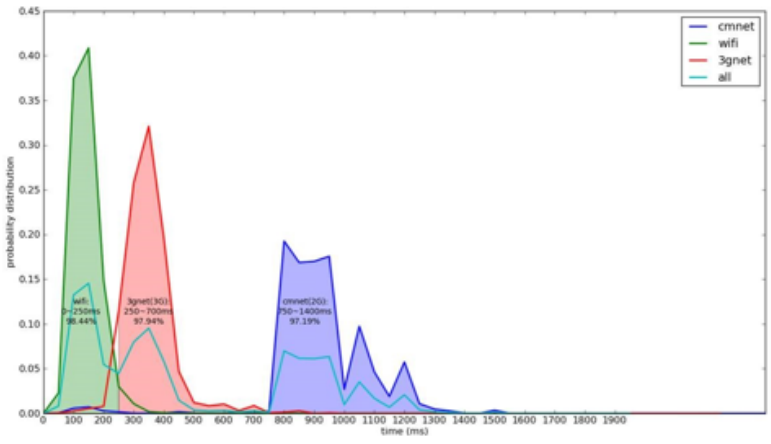
总下载时间默认可以统计**onload**时间，这样可以统计同步加载的资源全部加载完的耗时。如果页面中存在很多异步渲染，可以将异步渲染全部完成的时间作为总下载时间。

回到顶部

网络指标

网络类型判断

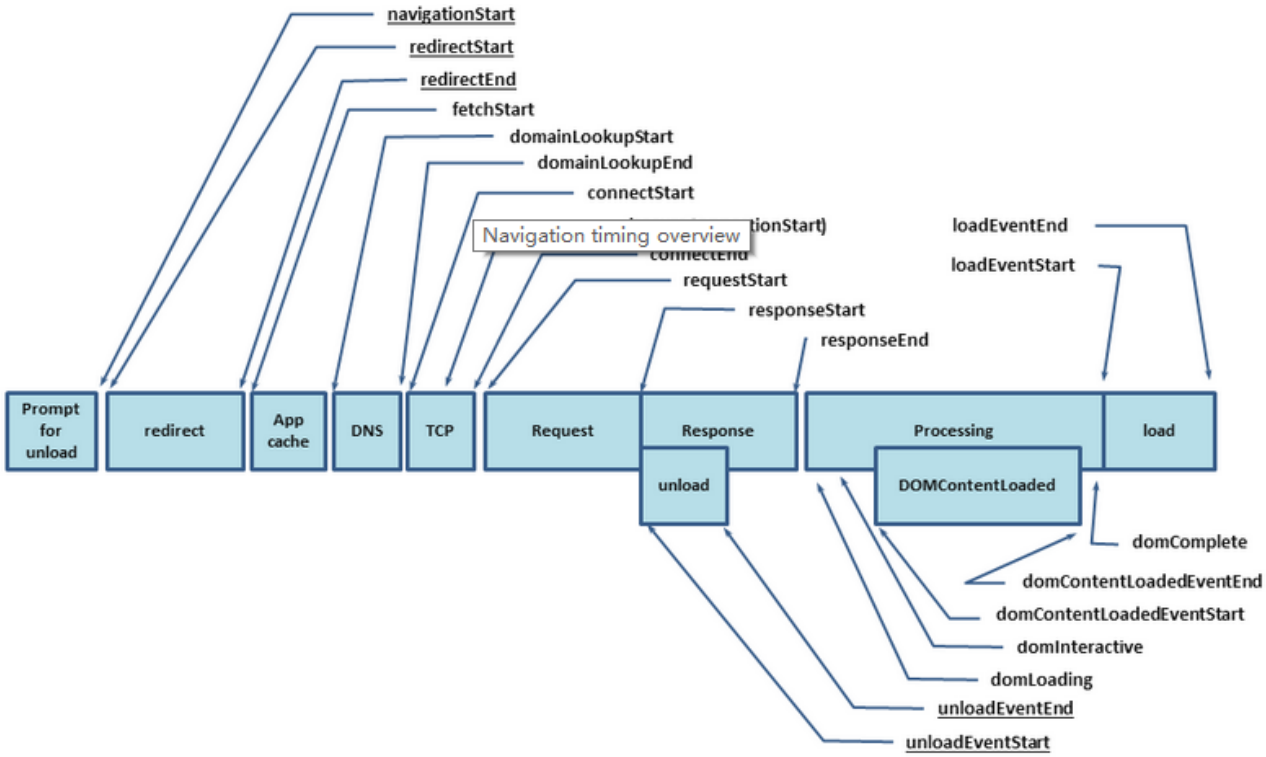
对于移动端来说，网络是页面加载速度最大的影响因素，需要根据不同的网络来采取相应的优化措施，例如对于 2G 用户采用简版等。但 web 上没有接口获取用户的网络类型。为了获取用户网络类型，可以通过测速的方式来判断不同 IP 段对应的网络。测速例如比较经典的有 facebook 的方案。经过测速后的分析，用户的加载速率有明显的分布区间，如下图所示：



各个分布区间正好对应不同的网络类型，经过与客户端的辅助测试，成功率可以在 95%以上。有了这个 IP 库对应的速率数据，就可以在分析用户数据时根据 IP 来判断用户网络类型。

网络耗时统计

网络耗时数据可以借助前面提到 Navigation Timing 接口获取，与之类似的还有Resource Timing,可以获取页面所有静态资源的加载耗时。通过此接口可以轻松获取 DNS、TCP、首字节、html 传输等耗时，Navigation Timing 的接口示意图如下所示：



以上重点介绍了数据采集部分，这也是系统中最关键的一部分，只有保证数据能真实反映用户感知，才能对症下药提升用户体验。数据采集完之后我们可以在页面加载完之后统一上报，如示例：

```
http://xxx.baidu.com/tj.gif?
dns=100&ct=210&st=300&tt=703&c_dnslookup=0&c_connecting=0&c_waiting=296&c_receiving=403&c_fetch_dns=0&c_nav_dns=75&
c_nav_fetch=75&drt=1423&drt_end=1436&lt=3410&c_nfpt=619&nav_type=0&redirect_count=0&_screen=1366*768|1366*728&produ
ct_id=10&page_id=200&_t=1399822334414
```

Day 5 如何分析性能数据?

> 让数据会说话

而数据分析过程，如前一篇文章所述，可以从多个维度去分析数据。大数据处理需要借助 hadoop、Hive 等方式，而对于普通站点则任意一种后端语言处理即可。

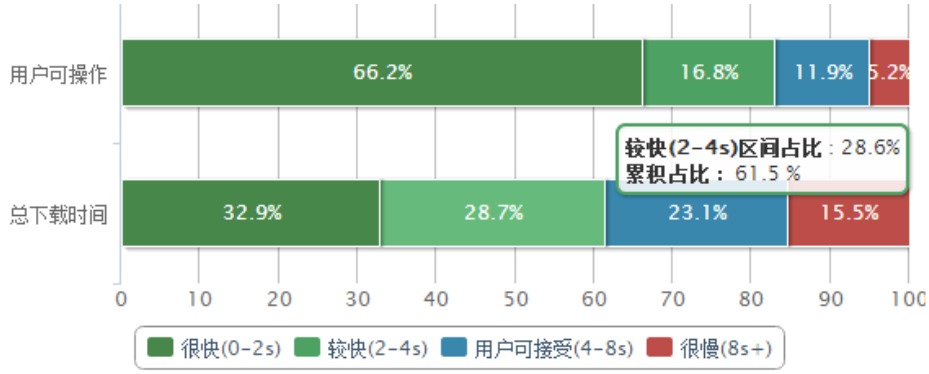
均值与分布

均值与分布是数据处理中最常见的两种方式。因为它能直观的表示指标的趋势与分布状况，方便进行评估、瓶颈发现与告警。处理过程中应去除异常值，例如明显超过阈值的脏数据等。

耗时的评估中，有很多这方面的研究数据。例如有人提出三个基本的时间范围：

- **0.1秒** : 0.1 秒是用户感知的最小粒度，在这个时间范围内完成的操作被认为是流畅没有延迟的
- **1.0秒** : 1.0 秒内完成的响应认为不会干扰用户的思维流。尽管用户能感觉到延迟，但 0.1 秒 -1.0 秒内完成的操作并不需要给出明显 loading 提示
- **10秒** : 达到 10 秒用户将无法保持注意力，很可能选择离开做其他事情

我们根据业界的一些调研，结合不同指标的特点，制定了指标的分布评估区间。如下图所示：



评估区间的制定方便我们了解当前性能状况，同时对性能趋势波动做出反应。

多维分析

为了方便挖掘性能可能的瓶颈，需要从多维的角度对数据进行分析。例如移动端最重要的维度就是网络，数据处理上除了总体数据，还需要根据网络类型对数据进行分析。常见的维度还有系统、浏览器、地域运营商等。我们还可以根据自身产品的特点来确定一些维度，例如页面长度分布、简版炫版等。

需要注意的是维度并不是越多越好，需要根据产品的特点及终端来确定。**维度是为了方便查找性能瓶颈。**

小插曲：之前从微博中看到有人评价说想做监控但是公司**没有日志服务器**。并不需要单独的日志服务器,只要能把统计的这个请求访问日志保存下来即可。如果网站自己的独立服务器都没有还有解决办法，在百度开发者中心新建一个应用，写一个简单的 Web 服务将接收到的统计数据解析存到百度云免费的数据库中，然后每天再用 Mysql 处理下当天的数据即可，对于普通站点的抽样性能数据应该没问题。请叫我雷锋。

Day 6 如何利用监控数据解决问题?

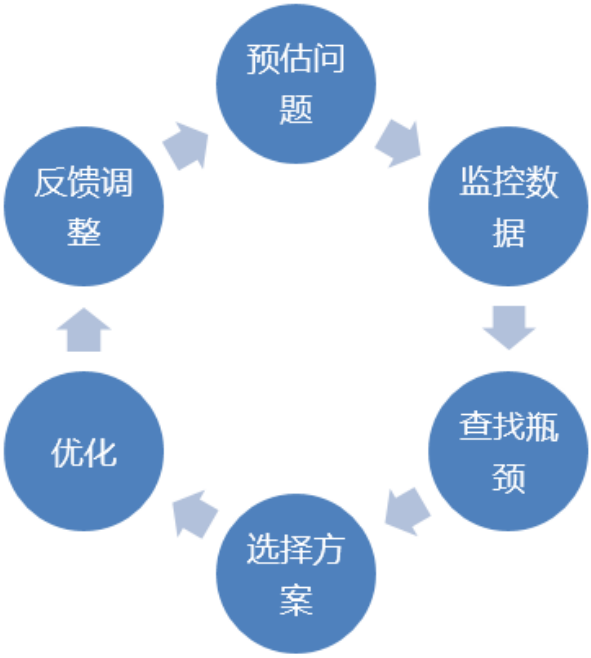
> 发现瓶颈，对症下药

对于图表制作，比较出名的有Highcharts，百度开发的Echarts也很不错。不管使用什么工具，最关键的一点就是让报表能**突出重点，直观明了。**

制作报表前多问几个如何让人直观看到目前状况和可能存在的问题，哪些地方可以加强，哪些可以去掉，使用是否习惯等。

有了能反映用户感知的真实世界、并且细分到各个业务功能，有详细的网络等辅助数据，我们在解决前端性能上便更加得心应手。监控系统已经对线上访问状况有了连续的反馈，根据现有评估与瓶颈选择对应方案进行优化，最后根据反馈进行调整，相信性能优化不再是个难题。

如何选择优化方案呢？这又是一个比较大的话题了，好在已经有很多经验可以借鉴。附录中就整理了部分**性能的学习资料**，可以根据需要阅读学习。



Day 7 总结

通过以上“几天”的努力，我们可以搭建一个小而美的前端性能监控系统。但这仅仅是开始，前端数据有很多挖掘的价值。性能优化也是一门需要认真学习的课程，为了打造流畅的使用体验，为了让用户更加满意，赶紧搭建起自己的前端数据平台吧!

该文写在w3ctech的走进名企 - 百度前端 FEX 专场之后，分享时的 PPT 在这里，视频在这里。