Web 的现状：网页性能提升指南

前端大全

互联网发展非常迅速，所以我们创造了Web平台。通常**我们会忽视连通性等问题，但用户们却不会视而不见。**一瞥万维网的现状，可以发现我们并没有用同情心、变通意识去构建它，更不要说性能了。

所以，今天的Web是什么状态呢？

在这个星球上的74亿人中，只有46%可以上网。平均网络速度上限为7Mb/s。更重要的是，有93%的互联网用户正在通过移动设备进行访问——若不适配移动设备将引起用户反感。通常情况下，数据比我们假设的更昂贵——可能需要1到13小时才能购买500MB的数据包（德国 vs. 巴西；更有趣的统计数据参见 Ben Schwarz 的 Beyond the Bubble: The Real World Performance）。

我们的网站也不是完美的——平均网站是原始Doom游戏的大小（约3 MB）（请注意，为了统计准确，应使用中位数，阅读 Ilya Grigorik 的优秀“平均页面”是一个神话，中档网站大小目前为1.4MB）。图像可以轻松占用1.7 MB的带宽，而JavaScript平均值也有400KB的体积。这不仅是Web平台的问题，原生应用程序可能更糟，还记得为了获取错误修复版本，而下载200MB安装包的情景吗？

技术人员经常会发现自己处于特权状态。随着最新的高端笔记本电脑、手机和快速有线互联网连接，很容易让我们忘记，这些并不是每个人都有的条件（实际上，真的很少）。

> 如果我们从特权和缺乏同情的角度来构建网络平台，那么将导致排他性的糟糕体验。

考虑到设计和开发的性能，我们怎样才能做得更好？

优化所有资源

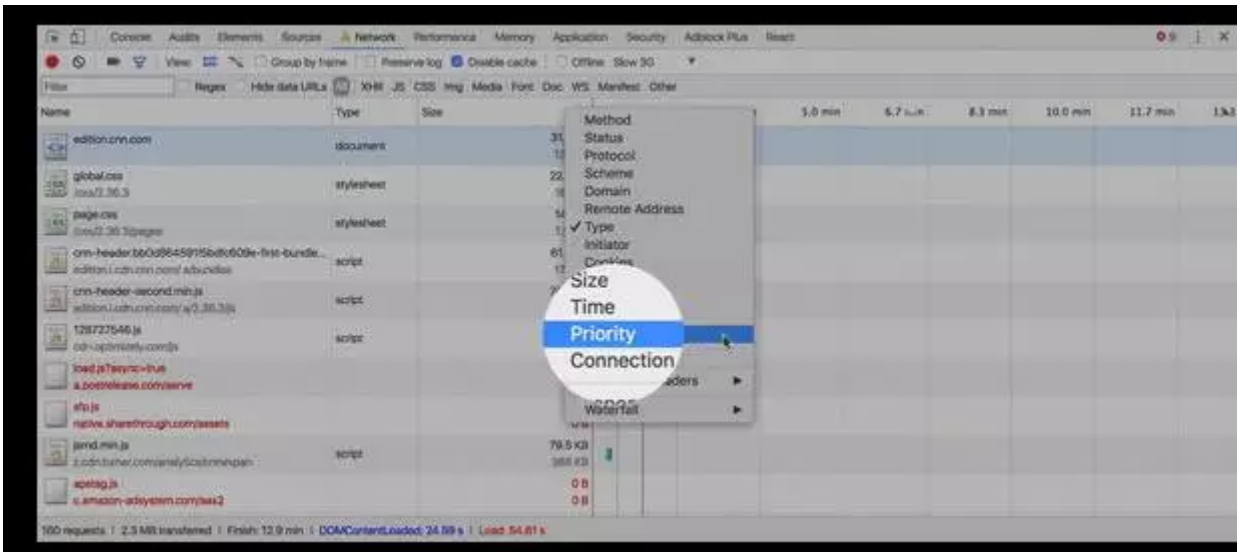
理解浏览器如何分析和处理资源，是显著提高性能的最强大但未充分利用的方式之一。事实证明，浏览器在嗅探资源方面非常出色，同时解析并确定其优先级。这里是**关键请求**的来源。

> 如果请求中包含用户视口中呈现内容所必需的资源，则该请求至关重要。

对于大多数网站，它将是HTML、必要的CSS、logo、网络字体，也可能是图片。在许多情况下，几十个其他不相关的资源（JavaScript、跟踪代码、广告等）影响了关键请求。幸运的是，我们能够通过仔细挑选重要资源并调整优先级来控制这种行为。

通过我们**可以手动强制调高资源的优先级，确保所需的内容按时呈现。**这种技术可以显著改善“交互时间”指标，从而使最佳的用户体验成为可能。

关键请求对许多人来说，似乎仍然是一个黑匣子，可共享资料的缺乏并不能改变现状。幸运的是，Ben Schwarz
发表了关于这个问题的非常全面并平易近人的文章——关键请求。另外，请参阅Addy的文章，在Chrome中的预加载、预取和优先级（Preload, Prefetch and Priorities in Chrome）。



[在Chrome开发人员工具中启用优先级]

要跟踪在请求优先级处理方面的情况，请使用Lighthouse性能工具和关键请求链审核工具，或查看Chrome开发人员工具中“网络”选项卡下的请求优先级。

通用性能清单

1. 积极地缓存
2. 启用压缩
3. 优化关键资源的优先级
4. 使用CDN（Content Delivery Networks）

图片优化

图片通常占网页传输的大部分有效载荷，因此图片优化可以带来最大的性能提升。有许多现有的策略和工具可以帮助我们删除额外的字节，但是首先应考虑的问题是：“图片对于我想传达的信息和效果至关重要吗？”。如果可以消除它，不仅可以节省带宽，而且还节省了请求。

在某些情况下，可以通过不同的技术实现类似的结果。比如CSS就具有艺术方向的一系列属性，例如阴影、渐变、动画及形状，允许我们构造适当风格的DOM元素。

选择正确的格式

如果不能舍弃图片，确定哪种格式更合适就很重要了。首先要在矢量和光栅图形之间做出选择：

- **矢量图形**：分辨率独立，通常体积更小。非常适合logo、icon和简单的图形，包括基本形状（线，多边形，圆和点）。
- **光栅图形**：呈现更详细的信息，比较适合相片。

做出首个决定后，可以选择以下几种格式：JPEG、GIF、PNG–8、PNG–24，或最新的 WEBP 与 JPEG-XR 格式。有了这么多的选项，如何确保我们做出正确的选择？以下是找出最佳格式的基本方法：

- **JPEG**：颜色非常丰富的图片（例如照片）
- **PNG–8**：色彩相对单一的图片
- **PNG–24**：局部透明的图片
- **GIF**：动图

Photoshop可以通过各种设置，例如降低质量、降低噪音或色彩数量来优化以上每一种格式。确保设计师了解上述性能实践，并能够使用正确的方式优化相应格式的图片。如果您想了解更多如何处理图片，请阅读 Lara Hogan 的好文 Designing for Performance。

试用新格式

图像格式有几个较新的玩家，即WebP、JPEG 2000 和 JPEG-XR。它们都是由浏览器厂商开发的：Google 的 WebP，Apple 的 JPEG 2000 和 Microsoft 的 JPEG-XR。

WebP 是最受欢迎的竞争者，支持无损和有损压缩，这使得它非常灵活。**无损的 WebP 比 PNG 小 26%，比 JPG 小25-34%**。WebP 有着74%的浏览器支持，它可以安全地进行降级，最多可节省1/3的传输字节。JPG 和 PNG 可以在 Photoshop 和其他图像处理应用程序以及命令行界面（`brew install webp`）中转换为WebP。

如果你想探索其他格式之间的视觉差异，推荐 Github 上这个很赞的 Demo。

用工具和算法进行优化

即使**使用了高效的图像格式，也不应跳过后处理优化**。这一步很重要。

如果您选择了尺寸相对较小的 SVG，它们也是可以再次压缩的。SVGO 是一个命令行工具，可以通过剥离不必要的元数据来快速优化 SVG。另外，如果您喜欢Web界面或受操作系统的限制，请使用 Jake Archibald 的 SVGOMG。因为 SVG 是基于 XML 的格式，它也可以在服务器端进行 GZIP 压缩。

ImageOptim 是大多其他图像类型的最好选择。包括 pngcrush、pngquant、MozJPEG、Google Zopfli等，它在一个全面的开源包中捆绑了一大堆优秀的工具。ImageOptim 可以以 Mac OS 应用程序、命令行界面和 Sketch 插件形式，轻松地实现到现有的工作流程中。对于那些在 Linux 或 Windows 上的场景，大多数 ImageOptim 的 CLI 都可以在您的平台上使用。

如果您倾向于尝试新兴的编码器，Google 今年早些时候发布了 Guetzli——源自 WebP 和 Zopfli 的开源算法。**Guetzli 可以比任何其他可用的压缩方法生成35%更小体积的 JPEG**。唯一的缺点是：处理时间慢（CPU 每处理百万像素需要1分钟）。

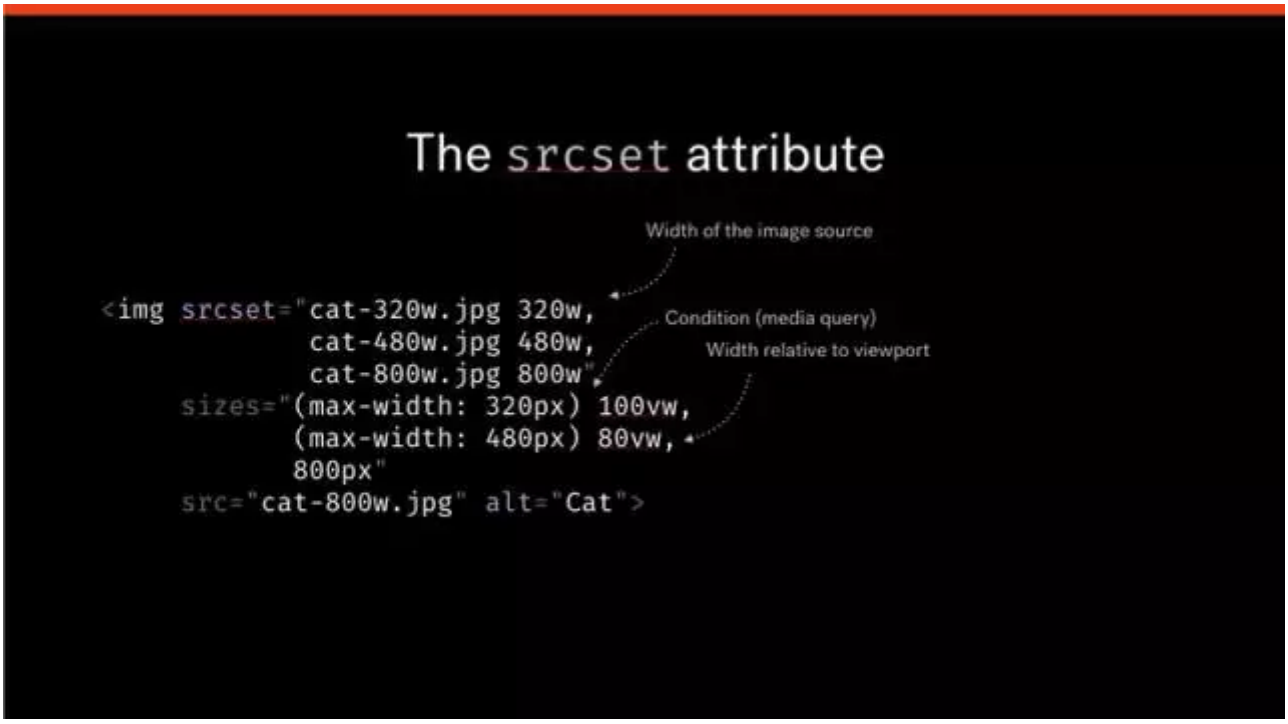
选择工具时，请确保它们生成所需的结果并适应团队的工作流程。理想情况是，将优化过程自动化，这样就不会产生漏掉的情况。

响应式图片

十年前，我们使用一种分辨率，就可以为所有人服务，但时代变化太快，现今的响应式 Web 已非往日可比。这也是为什么我们必须要特别留心，去精心优化视觉资源，确保它们适应各种视口设备。幸运的是，感谢 Responsive Images 社区小组，我们可以完美使用 `picture`元素和 `srcset` 属性（二者都有85%+支持率）。

srcset 属性

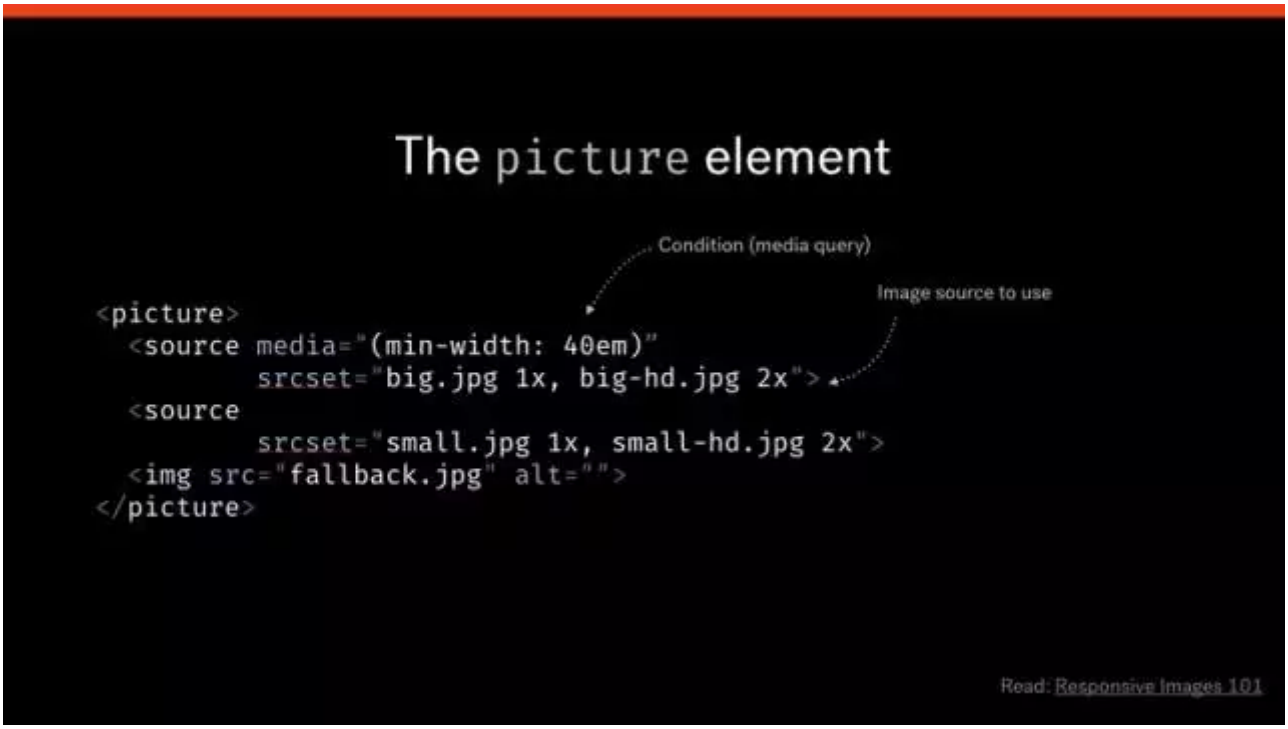
`srcset`在分辨率切换方案中效果最佳——即当我们需要根据用户的屏幕密度和大小显示图像时。基于`srcset`和`size`属性中的一组预定义规则，浏览器将选择最佳图片，相应地提供给视口。这项技术可以带来很大的带宽和请求节省，特别是对于移动用户。



[srcset 使用示例]

picture 元素

`picture`元素和`media`属性旨在使艺术设计变得容易。通过为不同情形提供不同图片（通过媒体查询进行测试），无论什么分辨率，我们都能始终将图像中最重要的元素保持在焦点。



[picture 元素使用示例]
请务必阅读 Jason Grigsby 的 Responsive Images 101指南，以便对这两种方法进行彻底的阐述。

使用图片 CDN 进行分发

视觉优化的最后一步是分发。所有资源都可以从使用 内容分发网络 中受益，但还有一些针对图片优化的特定工具，例如 Cloudinary 和 imgx。使用这些服务的好处远远超过了减少服务器上的流量，并显着降低了响应延迟。

CDN可以很好的解决重图片网站的复杂度，包括响应式服务与图片优化。虽然产品不同（价格也是如此），但是大多数方案都是根据设备和浏览器，调整大小、裁剪来确定哪种格式最适合用户。甚至更多——它们可以压缩、检测像素密度、水印、识别面部，并允许后置处理能力。借助这些强大的功能，和将参数附加到URL的能力，以用户为中心的图片服务变得十分容易。

图像性能清单

1. 选择正确的图片格式
2. 尽可能使用矢量图形
3. 如果变化不明显，则降低图片质量
4. 使用新格式图片
5. 使用工具与算法优化
6. 学习`srcset`和`picture`
7. 使用图片 CDN

Web 字体优化

自定义字体是一项非常强大的设计工具。但是能力伴随着很多责任。**现有68%的网站在使用 Web字体，这种类型的资源是性能杀手之一**（平均轻松可达100KB，取决于变体和字体的数量）。

即使体积不是最大的问题，**不可见文本闪动**（FOIT）也算是。当Web字体加载中或加载失败时，会发生FOIT，这会让空白页面，从而导致内容无法访问。首先仔细检查我们是否需要Web字体可能是值得的。如果真是这样，有一些策略可以帮助我们减轻对业务的负面影响。

选择正确的格式

有4种网络字体格式：EOT、TTF、WOFF 和最近的 WOFF2。TTF 和 WOFF 被广泛使用，拥有超过90%的浏览器支持率。根据支持情况，**最有可能安全地使用WOFF2**，并在旧版浏览器降级使用 WOFF。使用WOFF2的优点是，一套定制的预处理和压缩算法（如Brotli），并有大约30%的文件大小减少和改进的解析能力。

在`@font-face`中定义网页字体的来源时，请使用`format()`提示来指定应使用哪种格式。

如果您使用 Google Fonts 或 Typekit 来提供字体，这两种工具都实施了一些策略来优化其性能。Typekit 现在可以异步地为所有套件提供服务，防止 FOIT 以及允许其JavaScript套件代码的10天延长缓存期限（而不是默认10分钟）。Google Fonts 可以根据用户设备自动提供最小的文件。

审核字体范围

无论是否自主托管，字体数量、字体体积和样式，都将显著影响您的性能预算。

理想情况下，我们只需要一种包括常规和粗体的字体。如果您不确定如何选择字体范围，请参考 Lara Hogan 的 Weighing Aesthetics and Performance。

使用Unicode范围子集

Unicode范围子集允许将大字体分割成较小的集合。这是一个相对先进的策略，特别是在处理亚洲语言的时候，可能会带来显着的节省（你知道中文字体有平均数为 20,000 个字形吗？）。第一步是将字体限制为必要的语言集，例如拉丁语，希腊语或西里尔语。如果仅使用Web字体做Logo类使用，则应使用Unicode范围描述符，来选择特定字符。

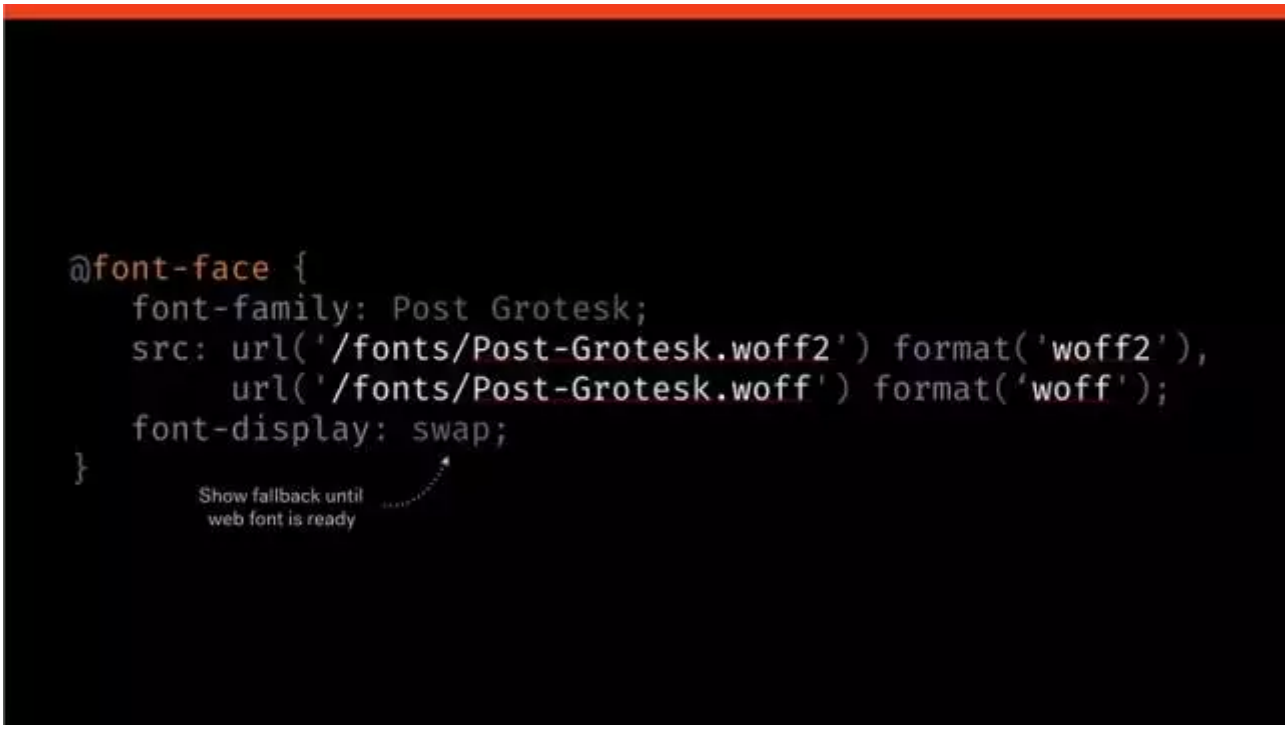
Filament Group 发布了一个开源命令行工具，可以根据文件或URL生成必要字形列表的 glyph hanger。或者，基于 Web 的 Font Squirrel Web Font Generator 提供高级子集和优化选项。如果在字体选择器界面中内置了使用Google Fonts 或 Typekit 选择语言子集，则使基本子集更容易。

建立字体加载策略

字体是阻塞渲染的——因为浏览器必须首先构建 DOM 和 CSSOM；在使用与现有节点相匹配的CSS选择器之前，浏览器并不会下载Web字体。这种行为会明显延迟文本呈现，通常会导致前面提到的不可见文本闪动（FOIT）。在较慢的网络和移动设备上，FOIT会更加显着。

实施字体加载策略，可防止用户无法访问您的内容。通常，选择**无样式文本闪动**（FOUT）是最简单和最有效的解决方案。

`font-display`是提供非 JavaScript 依赖解决方案的新 CSS 属性。不幸的是，它仅有部分支持（Chrome 和 Opera），目前正在 Firefox 和 WebKit 中开发。尽管如此，它可以并且应该与其他字体加载机制结合使用。



[font-display 属性实践]
幸运的是，Typekit 的 Web Font Loader 和 Bram Stein 的 Font Face Observer 可以帮助管理字体加载行为。此外，网页字体性能专家 Zach Leatherman 发布了字体加载策略综合指南，这将有助于为您的项目选择正确的方法。

字体性能清单

1. 选择正确的格式
2. 审核字体范围
3. 使用Unicode范围子集
4. 建立字体加载策略

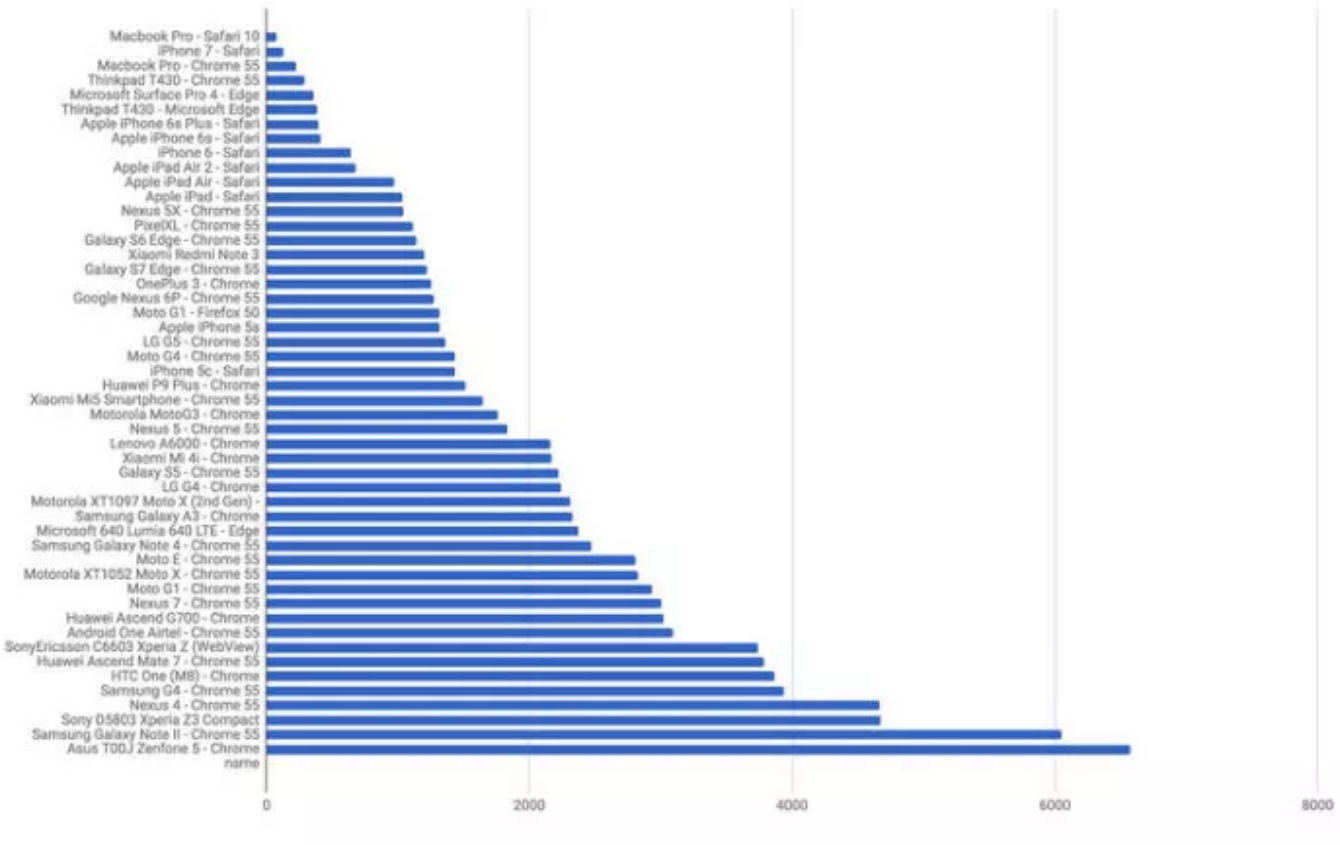
JavaScript 优化

目前，JavaScript 的平均大小为446 KB，已经使其成为第二大的资源类型（第一为图片）。

> 我们可能没有意识到，我们所爱的JavaScript隐藏着更加严峻的性能瓶颈。

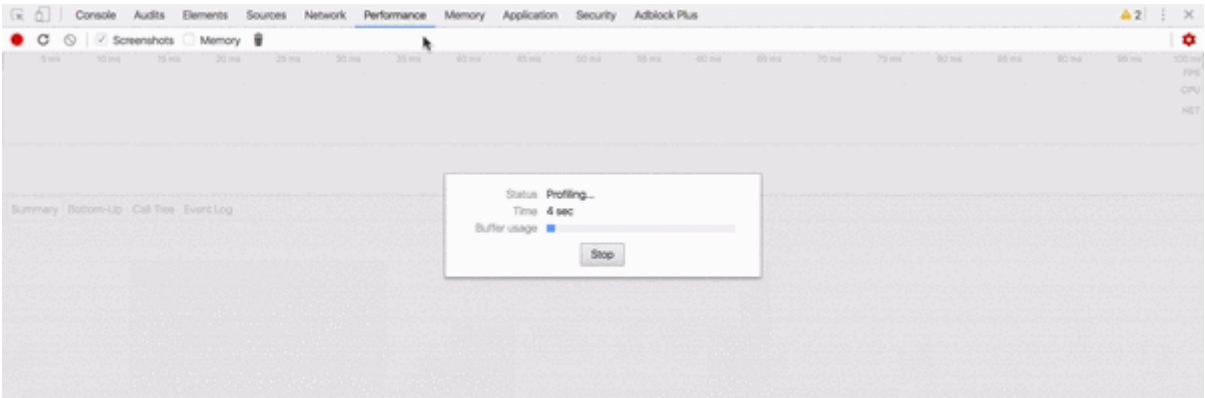
监控JavaScript的流量

优化交付只是解决网页膨胀的第一步。JavaScript 下载后，必须由浏览器进行解析、编译和运行。快速浏览一些流行的网站，显而易见的是，gzip 压缩的 JS 在**解压之后至少变大三倍**。事实上，我们正在发送一大堆代码。



1MB JavaScript 在不同设备上的解析时间。图片由 Addy Osmani 和他的 JavaScript Start-up Performance 文章提供。

分析解析和编译时间，对于理解应用程序是否准备好进行交互至关重要。这些耗时根据用户设备的硬件能力而异。**解析和编译会很容易在低端手机上高出2-5倍**。Addy的研究证实，在常规手机上，一个应用程序将需要16秒才能达到交互式状态，而在桌面电脑上为8秒。分析这些指标至关重要，幸运的是，我们可以通过 Chrome DevTools 来完成。



[在 Chrome 开发工具中查看解析和编译过程]
请务必阅读 Addy Osmani 对 JavaScript 启动性能的详细说明。

摆脱不必要的依赖

现代软件包管理器的工作方式，可以轻而易举地掩盖依赖关系的数量和大小。webpack-bundle-analyzer 和 Bundle Buddy 是很好的可视化工具，帮助识别出代码重复、最大性能问题和过时的、不必要的依赖。

图 webpack bundle analyzer 实践（译者注：原gif太大，只能用外链了）

通过 VS Code 和 Atom 中的`Import Cost`扩展，我们可以使导入依赖成本更加明显。

图 VS Code `Import Code`扩展

实现代码分割

只要有可能，**我们就应只提供用户体验所必需的资源**。向用户发送一个完整的 bundle.js 文件，包括他们可能永远看不到的交互效果处理代码，并不太理想（假设在访问着陆页时，去下载处理整个应用程序的 JavaScript）。同样，我们不应普遍提供针对特定浏览器或用户代理的代码。

Webpack，最受欢迎的模块打包器之一，天生具备代码分割支持。最简单的代码分割可以按页面实现（如用于着陆页的`home.js`，联系人页面的`contact.js`等），Webpack 还提供了一些高级策略，如动态导入及延迟加载，值得一看。

考虑框架选择

JavaScript 前端框架日新月异。根据2016年的 JavaScript 调查，React 是最受欢迎的选择。仔细审视架构选择，可能会发现，您可以使用更为轻量级的替代方案，例如 Preact（请注意，Preact 并不是一个完整的 React 重新实现，只是一个高性能，功能更轻的虚拟 DOM 库）。类似地，我们可以将较大的库更换成更小的版本——`moment.js`换成`date-fns`（或者在特定情况，删除`moment.js`中未使用的 locales）。

在开始一个新项目之前，有必要确定什么样的功能是必需的，并为您的需求和目标选择最具性能的框架。有时这可能意味着选择写更多的原生 JavaScript。

JavaScript 性能清单

1. 监控 JavaScript 流量
2. 摆脱不必要的依赖
3. 实现代码分割
4. 考虑框架选择

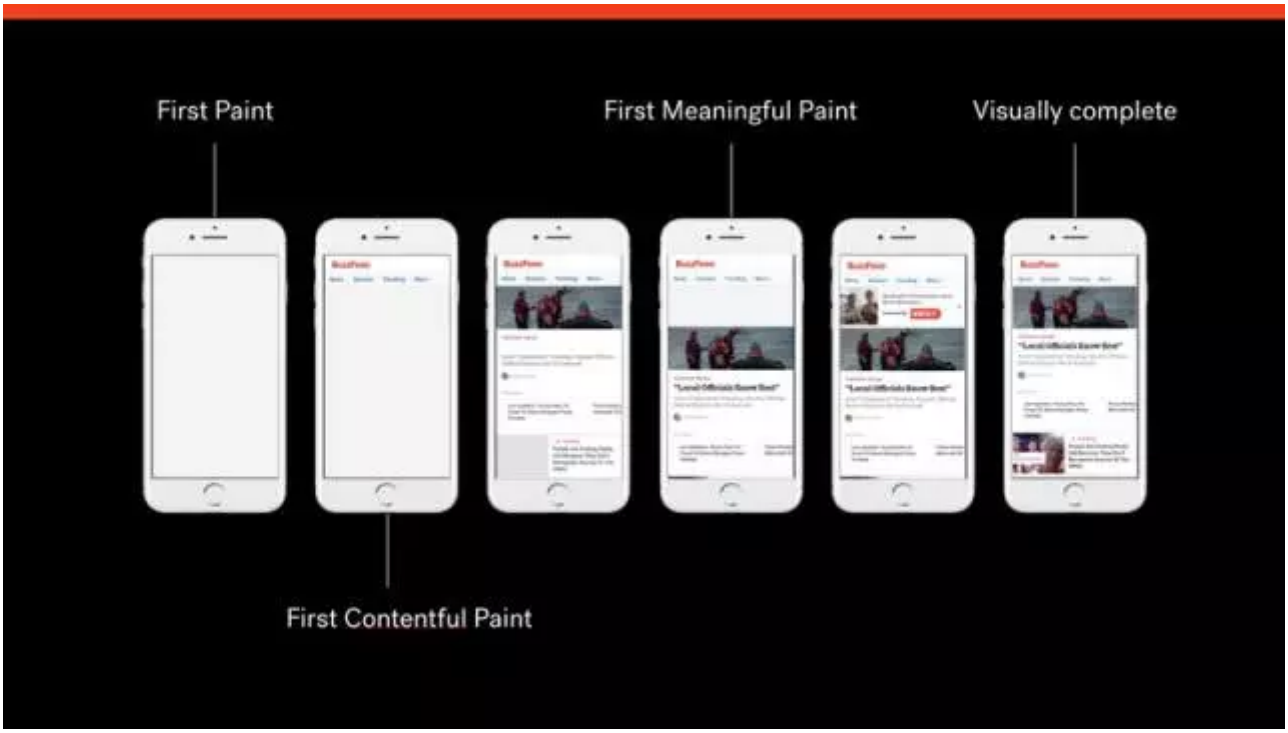
性能追踪，前进之路

我们已经讨论了一些策略，在大多数情况下会对我们正在建立的产品用户体验产生积极的变化。性能可能是一个棘手的问题，有必要长期地跟踪我们调整的结果。

以用户为中心的性能指标

卓越的性能指标，旨在尽可能接近描绘用户体验。以往的`onLoad`，`onContentLoaded`或`SpeedIndex`对「用户多快能与页面交互」给出的信息非常少。当聚焦到传输资源时，量化地感知性能十分困难。好在，有一些时间可以全面地描述内容的可视性和互动性。

这些指标是首次渲染（First Paint），首次有意义渲染（First Meaningful Paint），视觉完整（Visually Complete）和可交互时间（Time to Interactive）。



- **首次渲染**：浏览器从白色屏幕到第一次视觉呈现的变化。
- **首次有意义渲染**：文字，图像和主要内容都已可见。
- **视觉完整**：视口中的所有内容都可见。
- **可交互时间**：视口中的所有内容都是可见的，可以与之进行交互（JavaScript 主线程停止活动）。

这些时间直接对应于用户的实际体验，因此可以作为重点进行追踪。如果可能，将它们记录全部，否则选择一两个来更好地监控性能。其他指标也需要留意，特别是我们发送的字节数（优化和解压缩）。

设置性能预算

所有这些上报数字可能会很快变得混乱和不易理解。没有可操作的目标和对象，很容易迷失我们最初的目的。几年前，Tim Kadlec 写过关于性能预算的概念。

遗憾的是，并没有一个万能的神奇公式。性能预算通常归结为竞争分析和产品目标，而这是每个业务所各异的。

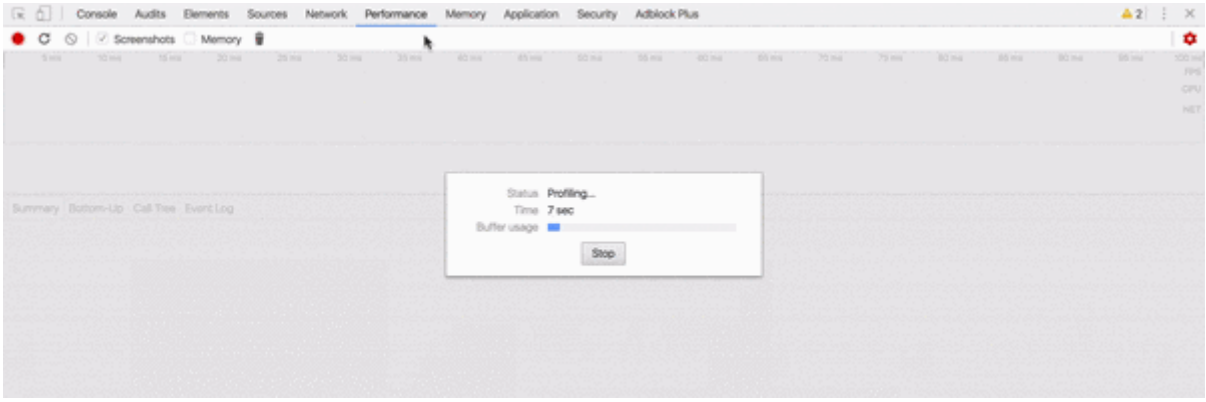
设定预算时，重要的是要达到明显的差异，通常是至少改善20%。实践和迭代您的预算，利用 Lara Hogan 的方法新设计与性能预算作为参考。

试用性能预算计算器或Chrome扩展浏览器卡路里，以帮助创建预算。

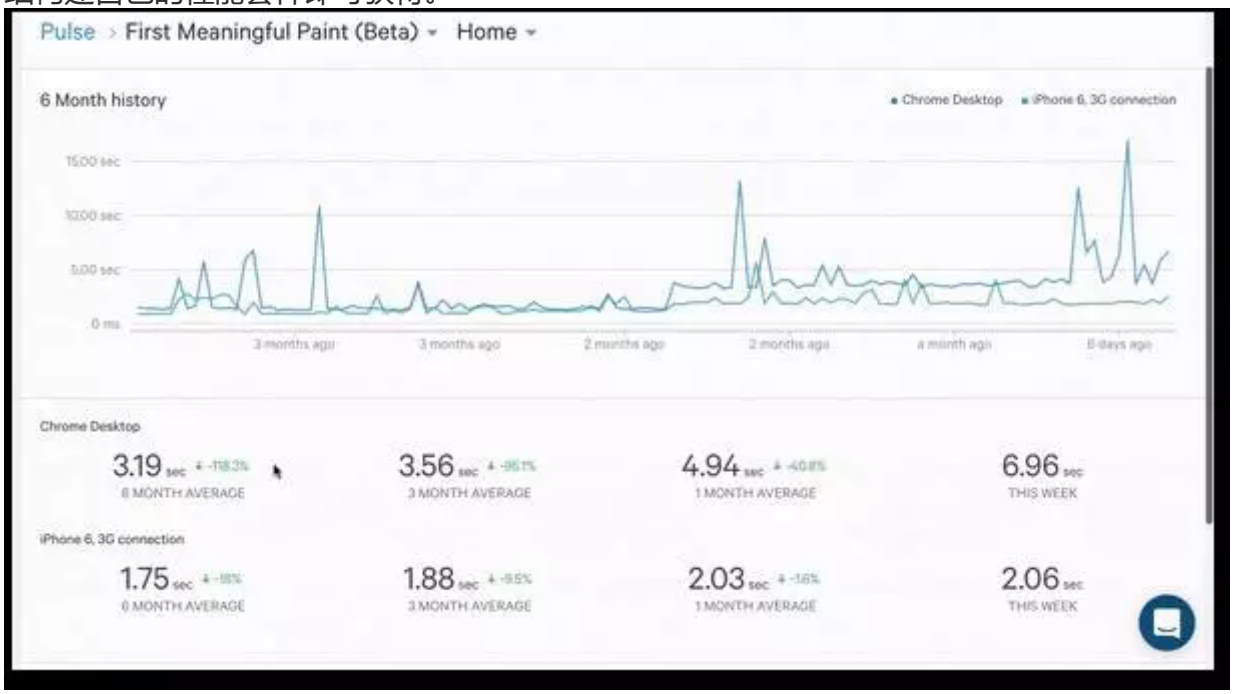
持续监控

监控性能不应该是手动的。市面上有很多强大的工具，还可以提供全面的报告。

Google Lighthouse 是一个可以审核性能、可访问性、渐进式网络应用程序等的开源项目。您可以在命令行中或直接在 Chrome Developer Tools 中使用Lighthouse。



[Lighthouse 运行一次性能审查]
对于持续的追踪，选择选择 Calibre，它可以提供性能预算、设备仿真、分布式监控和许多其他功能，无需我们仔细构建自己的性能套件即可获得。



[Calibre 报表]

无论您在追踪什么，请确保使整个团队或组织能够透明地访问数据。

> 性能是一项分担责任，远远超过开发人员团队——我们都应对所创建的用户体验负责，不管是什么角色或职级。

倡导速度和建立协作流程，以便在产品决策或设计早期阶段，尽早暴露可能遇到的瓶颈，是非常重要的。

建立性能意识和同情心

关心性能不仅仅是一个业务目标（但是如果您需要通过销售统计数据来进行销售，那么可以通过PWA统计）。这是关于基本的同情和用户体验放在第一位。

> 作为技术专家，我们的责任是，不要让用户的注意力和时间放在等待页面上，而已可以更开心地花费在其他地方。我们的目标是建立意识到时间和人们关注的工具。

提倡性能意识应该是每个人的目标。让我们抱着性能和同情心，为大家建立一个更好、更有意义的未来吧。