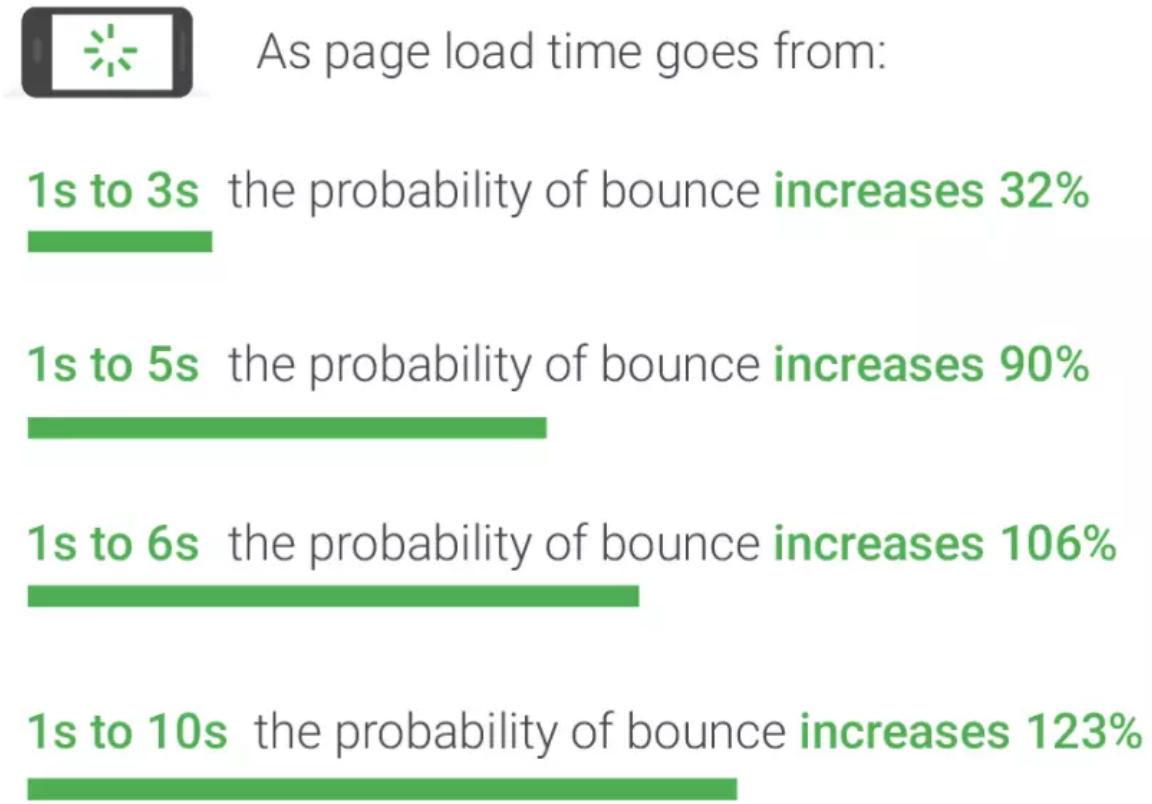
> 通常来说，工程师熟悉某个系统会只要搞清楚其数据结构和数据流转过程基本足够了，但从商业的角度来看，前后端工程师都应该关注的另一个重要维度是系统性能，纯技术角度来看性能可以认为是系统的响应速度（实际上还可以认为是执行效率等），而从用户角度来看性能就是使用体验。

网页性能为什么重要？

页面性能差的直接后果是用户需要等待，而等待，尤其是不确定要多长时间的等待会给用户带来焦虑，为了尽早的结束这种焦虑，除非访问网页是刚需，用户通常会选择直接关闭页面。从实际数据来看，性能差是页面高跳出率的重要原因之一。

为了搞清楚页面性能对业务目标的影响，诸如 Yahoo、Google、Amazon 等科技公司都投入了不少资源去研究和优化，比如下面是 ThinkWithGoogle 运用神经网络分析 2017 年 1100 万广告落地页[加载速度和页面跳出率关系所得到的结论](#)：



结论显而易见：**越快越好，少即是多**。

实际上，最近几年来大型互联网公司在页面性能优化研发中产出了不少工具和文档，方便工程师给网站做性能分析和优化。

其中文档方面比较经典的当属三本书：

- High Performance Websites，早期在 Yahoo 性能团队工作的 Steve Souders 所著；
- Even Faster Websites，同上，在上册的基础上，列出了很多行之有效的细节优化手段；
- High Performance Browser Networks，Google 性能工程师 Ilya Grigorik 所著，可以免费在线阅读，当然如果想支持读者，可以选择购买；

而工具则非常多，尤其是 2015 年开始爆发的各种应用性能管理（APM，如 New Relic）工具，对工程师来说，比较经典易用的有：

- WebPageTest，可以认为是网页性能分析工具中的 Vim 了，纯开源项目，在全球都有节点，分析思路基本与 Yahoo 性能黄金法则相贴合；
- LightHouse，已经集成在 Chrome 开发者工具中，能够从现代 WEB 应用比较重要的几个维度给出分析结果，比如加载速度、PWA、可用性、SEO 等，工具易用性、可得性都高于 WebPageTest，个人强烈建议；

网页性能该怎么衡量？

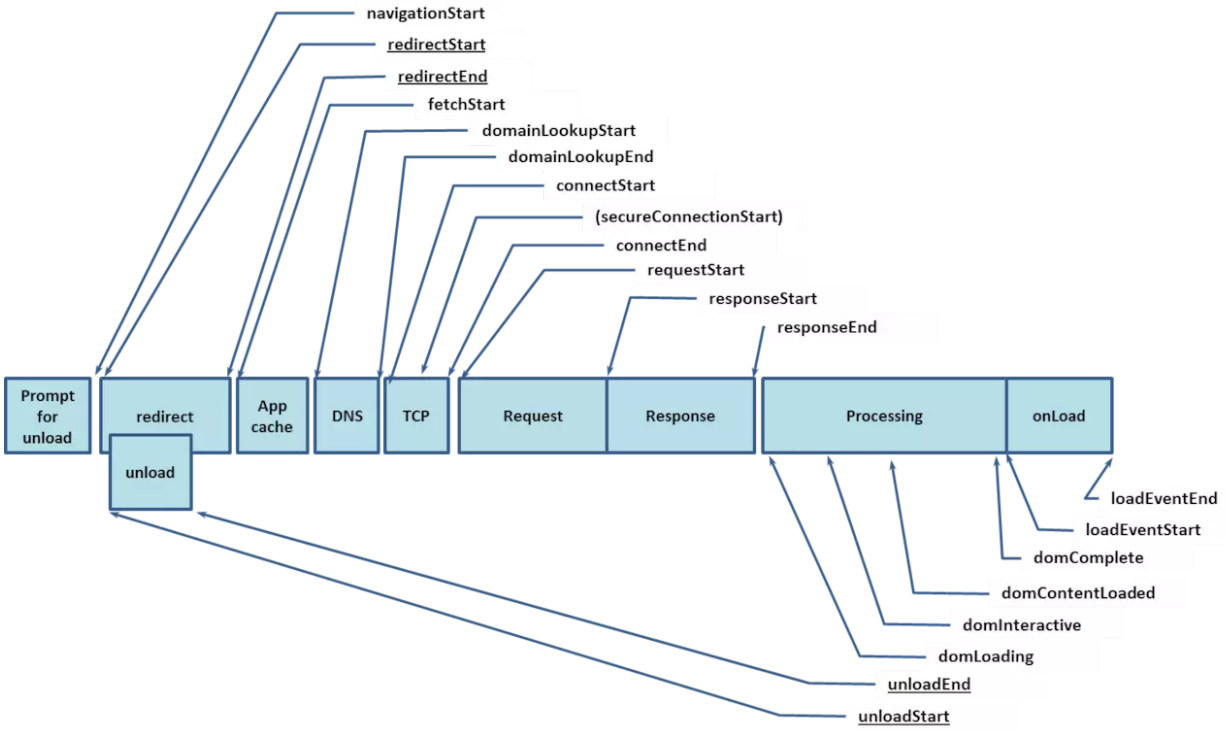
要清晰、准确的衡量网页性能，我们先要定义页面性能，如何定义页面性能？

1. 从后端角度看，可以是**首字节时间**，即页面发起请求到浏览器收到第一个响应字节的时间，英文 Time to First Byte；
2. 从浏览器角度看，可以是页面所依赖的全部静态资源加载完成所需要的时间，即常说的**完全加载时间**；
3. 从用户角度看，可以是敲回车键开始到看到页面开始渲染的过程所需的时间，即常说的**首次渲染时间**，此概念还可以细分，比如 WebPageTest 和 LightHouse 都有的 FirstMeaningfulPaint；

性能优化工作处在不同阶段，或者业务场景不同，上面不同定义视角的适用性是不同的，也有把上面 3 种衡量方法加权求和得到综合的性能指数。

要想真正开始做优化，需要搞清楚从发起请求到浏览器渲染页面并呈现给用户的过程中有哪些关键环节，好在现代浏览器提供的 Navigation Timing API 已经把这个过

程标准化，方便我们做性能指标的计算，如下图：



举例来说上面提到的首字节时间和完全加载时间可以用如下公式计算：

```
首字节时间 = responseStart - navigationStart
完全加载时间 = loadEventEnd - navigationStart
```

至于首次渲染时间，准确的计算方式需要结合录屏，篇幅原因，这里不做展开。

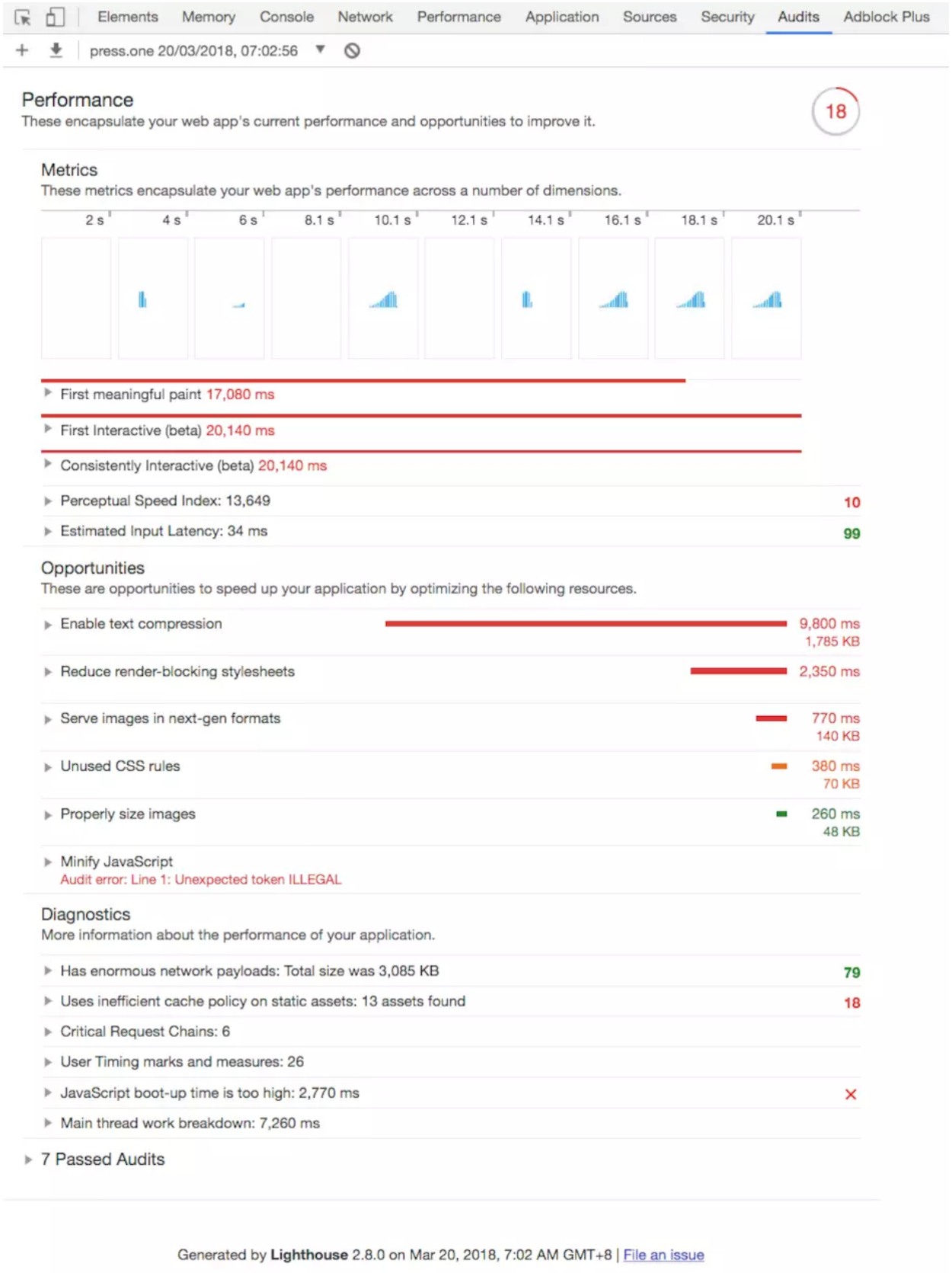
PressOne 首页性能 CaseStudy

PressOne 是基于区块链的内容分发公链，而 https://press.one 则是项目入口，目前功能还比较简单，主要包括账户创建、用户登录、三方账号绑定、用户主页、内容签名等功能。首页是任何网站的门户，确保其访问速度和体验的重要性不言而喻，而 press.one 给笔者的初体验除了新奇还包括慢，新用户加载页面平均需要 6s 以上，即使内容渲染之前有加载中提示，还是有明显的等待感。

虽然 press.one 是基于 angular 开发的单页应用，适用于传统页面的大部分性能优化方法同样适用，下面结合 LightHouse 对 press.one 首页做简单的性能分析，并列出行之有效的优化行动清单。

LightHouse 可以独立安装使用，也可以在 Google Chrome 中使用，因为集成到了开发者工具的 Audits 面板中，使用方法比较简单，建议直接阅读文档。

下面是使用 Google Chrome 做性能诊断的结果：



百分制的性能指数结果是 18 分，3G 网络下的完全可交互时间长达 20S，通常到 10S 用户基本都以为网站坏掉了，可见优化的空间是巨大的。

LightHouse 列出的优化手段（Opportunities）和可能的原因诊断（Diagnostics）都比较直观：

- Enable text compression，启用文本压缩，针对 JS、CSS 等静态资源是非常有效的优化手段，通常可节省 60% 以上，实施成本低，收益巨大，如果加上适当的缓存，可以对重复访问用户更加友好；
- Reduce render-blocking stylesheets，减少阻塞渲染的样式，需要把首屏渲染的样式从整体样式中剥离出来优先加载，实施成本偏高，收益中等；
- Unused CSS rules 是针对 CSS 文件的优化，减少传输实际上不需要的代码，我仔细拔了下代码，项目引入的 fontawesome 貌似确实没用到，实际上这个库却不小，实施成本小，收益中等；
- Serve images in next-gen formats，使用更好的压缩算法压缩图片，实施成本低，收益大；
- Proper size images，请求的图片和使用尺寸对应，避免缩放，造成浪费，实施成本低，收益看情况；

如果上面的优化手段都落地实施，重新评估性能指数大概率可以达到及格线。

结合瀑布流分析，我们也不难发现更进一步的优化点，关键资源加载链条（Critical Request Chain）太长了，如下图：

″ style='height:308.109px;width:652px;float:none;border:0px none rgb(0, 0, 0);'/>

要开始首次渲染（对应如下瀑布流图右侧蓝色的竖线）除主文档外，我们还需要额外下载 5 个 JS 文件，1 个 CSS 文件：

″ style='height:287.734px;width:652px;float:none;border:0px none rgb(0, 0, 0);'/>

关键资源加载链条太长的问题怎么优化呢？

- 合并资源请求，比如 elliptic 和 keythereum 的三方依赖可以直接合并，更激进的可以和 polyfill 也合并；
- 合理使用懒加载，首次渲染不需要的资源做适当的拆分，SPA 和传统页面都可以实现，不让非关键资源阻塞页面首次渲染；
- 合理使用 CDN，因为通常 CDN 是地理位置离用户更近的节点，可以大大节省网络传输的 RTT；

关键资源加载链条的问题解决之后，重新评估的性能指数大概率能达到 80 分。

那么接下来呢？**理论上看起来性能优化是无止境的，实际上任何一个领域花 20% 的时间能达到 80% 的目的（比如把页面首次渲染时间降到 3s 以内），然后可以收手去解决更重要的问题**，当然时间允许的话，有追求的工程师会不断的问自己，这是我能做到的极致么？

继续往下优化则要从页面渲染的角度去考虑，毕竟我们已经尽可能快的把渲染页面所需的各种资源交给了浏览器，怎么让它更快、更流畅的渲染出可交互的页面是接下来需要重点考虑的。而渲染速度跟 DOM 节点的组织、JS 的组织、JS 和 DOM 交互的优化都有关系，篇幅原因，可以单独写文章介绍。

以上，希望对你有用。