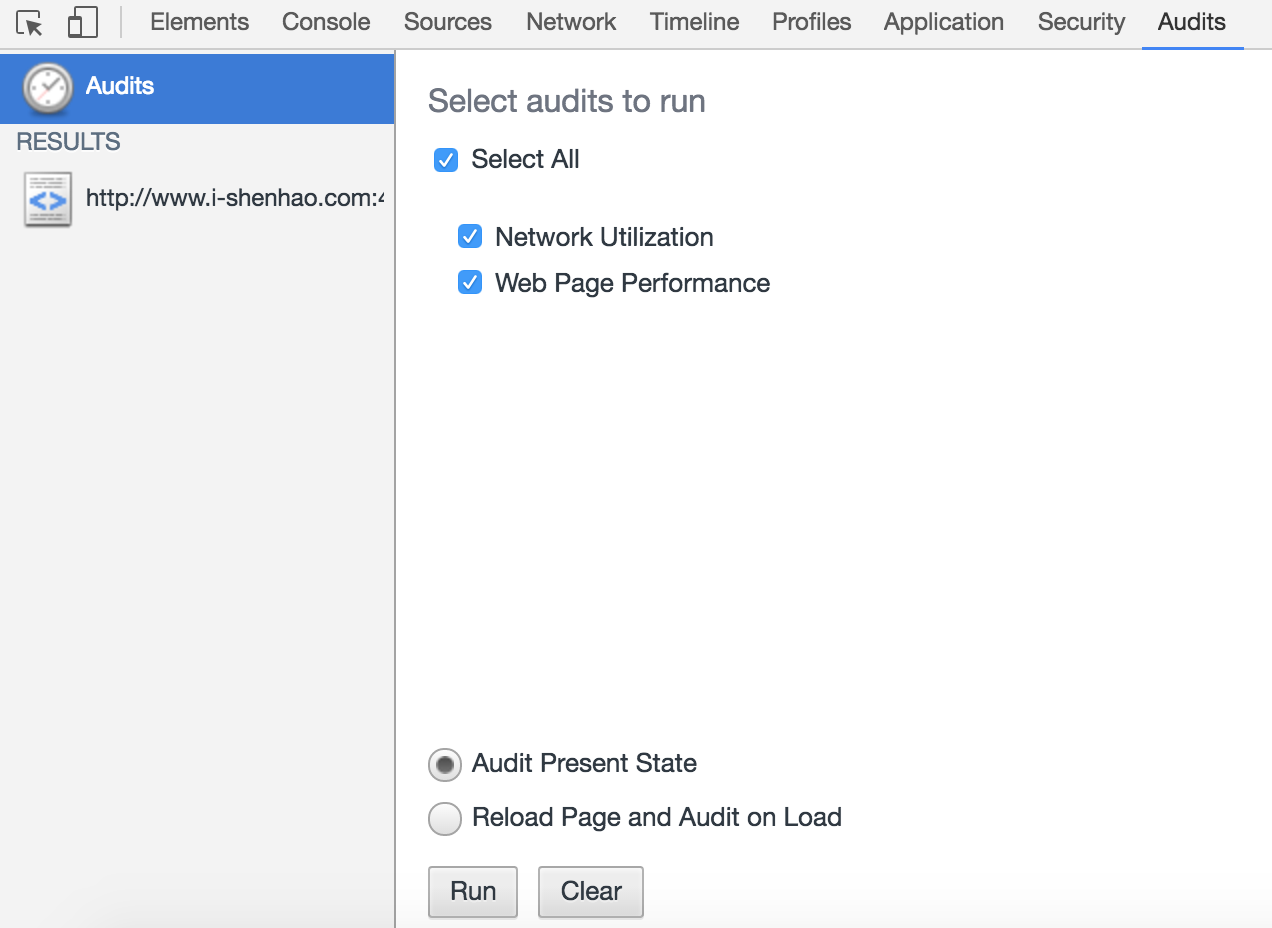
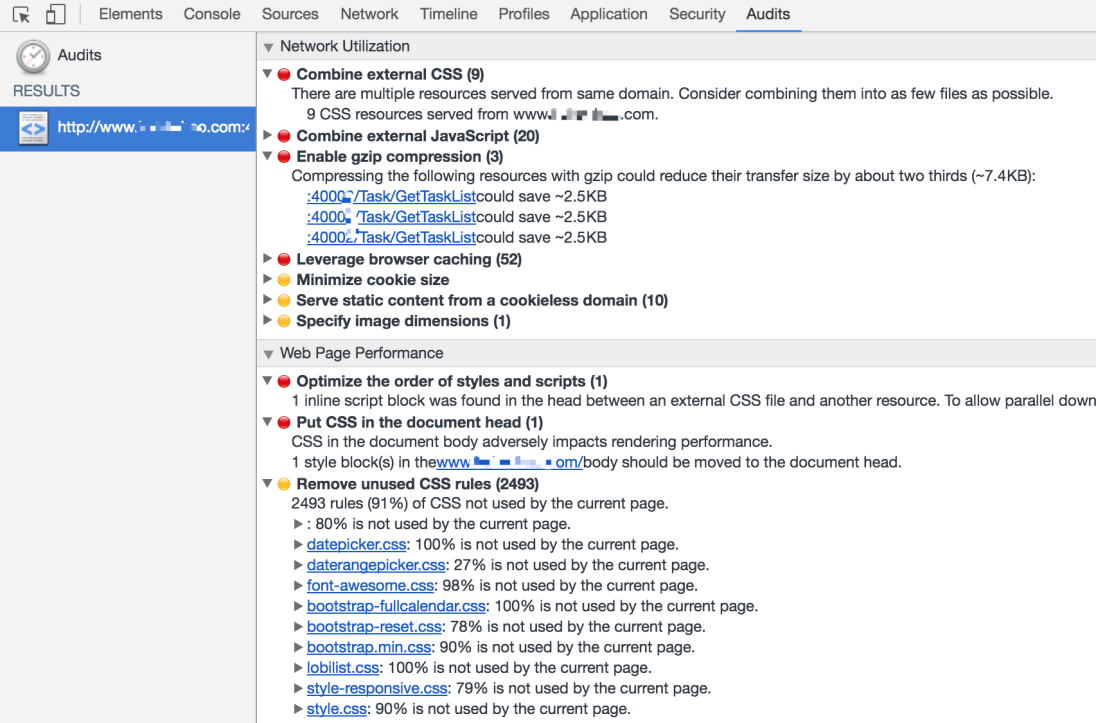
## 1.谷歌59 版Audits面板简介

对当前网页进行网络利用情况、网页性能方面的诊断，并给出一些优化建议。比如列出所有没有用到的CSS文件等。



选中**Network Utilization**、**Web Page Performance**，点击**Run**按钮，将会对当前页面进行网络利用率和网页的性能优化作出诊断，并给出相应的优化建议。



网页性能部分

**1、优化样式和脚本的顺序：Optimize the order of styles and scripts (4)**

* 1. The following external CSS files were included after an external JavaScript file in the document head. To ensure CSS files are downloaded in parallel, always include external CSS before external JavaScript.

**2、把CSS放到head中：Put CSS in the document head (3)**

**CSS in the document body adversely impacts rendering performance.**

**3、删除没用的CSS：Remove unused CSS rules (44)**

* 1. 44 rules (19%) of CSS not used by the current page.

**4、Use normal CSS property names instead of vendor-prefixed ones (3)**

**2谷歌60 Audits面板简介**

Audits 面板基于谷歌开发的 Lighthouse，它提供了一整套的测试来评估网页的质量，并且是一个开源的项目。

### ****如何使用 Lighthouse****

Audits 标签是 DevTool 内置选项卡的最后一个标签。想要使用它的话，需要安装 Chrome 60 的开发版或者 Canary 版。

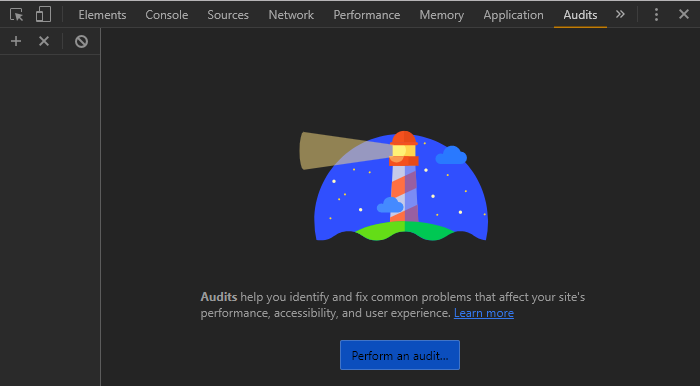
你可以按照以下步骤来审查页面：

按 F12 来打开DevTool。

点击 **Audits** 标签

点击 **Perform an audit**

点击 **Run audit**。Lighthouse 会启动 DevTool 来模拟一个移动设备，在页面上运行一些测试，然后将结果显示在 Audit 面板中。

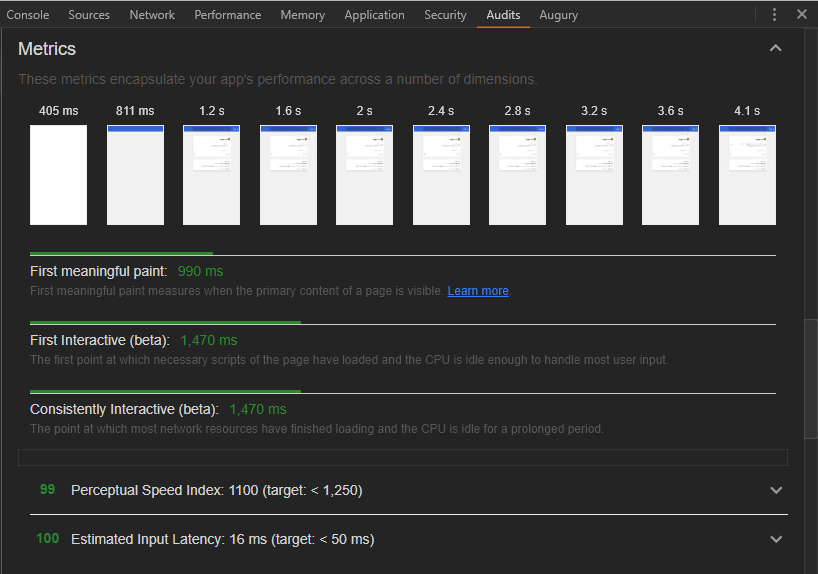


Lighthouse 从4个方面来分析页面： **性能、可访问性、最佳实践和 PWA**。

#### ****性能****

Web 性能是指网页在浏览器上下载和渲染的速度。web 性能优化是提高 web 性能的一个技术领域。

事实证明，网站的速度能够影响访客的留存率、忠诚度和用户满意度，特别是对于网速较慢的用户和移动设备上的用户。



性能类别的第一部分是**指标**。这些指标从多个维度对应用程序的性能进行评估

性能的指标

正如你所看到的，页面加载有包括3个重点：

**首次有效绘制**-衡量的是用户看到网页的主要内容所需的时间。

**首次交互** - 指的是页面加载完成必要的脚本，并且 CPU 足以应对大多数用户输入时的所需的时间。

**持续交互**-指的是页面中的大多数网络资源完成加载，并且 CPU 在很长一段时间都很空闲的所需的时间。

性能的下一个部分是**可优化项**。 例如你可以通过压缩资源图像和代码来让程序运行的更快，这些都是可优化的地方。

**如何优化我们的网站：**

大家可以对照着去优化自己的网站或者相关项目。

**0.提高服务器的响应速度**

砸钱的东西，但却最根本；搞好这一项，甚比下面任何一项。

**1.优化样式表和脚本的排列顺序**

正确地排列外部样式表与外部和内嵌脚本的顺序，可增加下载时同时加载的数据量，并提高浏览器显示网页的速度。

将样式表放在顶部，将脚本放在底部

**2.使用浏览器缓存**

在 HTTP 标头中为静态资源设置有效期或最长存在时间（Google建议最短为一周，最好能达一年左右，⊙﹏⊙b汗），可指示浏览器从本地磁盘中加载以前下载的资源而不是从网络中加载。

**3.使目标网页重定向可缓存**

许多网页都会将用户重定向至不同的网址，例如：从 www.example.com 重定向至 m.example.com。使这些重定向信息可缓存到用户的浏览器中，可加快访问者多次访问同一网站时的网页加载速度。

**4.内嵌小型 CSS**

将小型样式表内嵌到主 HTML 网页中，可减少在下载其他资源时的往返时间 (RTT) 和延迟时间。

**5.内嵌小型 JavaScript**

将小型 JavaScript 文件内嵌到主 HTML 网页中，可减少在下载其他资源时的往返时间 (RTT) 和延迟时间。

**6.压缩 CSS**

压缩 CSS 代码可以节省大量的数据字节空间，并提高下载、解析和执行的速度。

**7.压缩 HTML**

压缩 HTML 代码（包括其中所含的任何内嵌 JavaScript 和 CSS）可节省大量的数据字节空间，并提高下载、解析和执行的速度。

**8.压缩 JavaScript**

压缩 JavaScript 代码可以节省大量的数据字节空间，并提高下载、解析和执行的速度。

**9.启用 Keep-Alive**

有效地降低TCP握手的次数，减少httpd进程数，降低内存的使用

**10.启用压缩**

使用 gzip 或 deflate 压缩资源，可减少通过网络发送的字节数。

**11.将图片组合为 CSS 贴图定位（精灵图）**

尽量减少使用 CSS 贴图定位合并图片后的文件数量，可减少在下载其他资源时的往返时间和延迟时间、节省请求开销，并减少网页下载的总字节数。

**12.将查询字符串从静态资源中删除**

在 HTTP 标头中为静态资源启用公共缓存，可让浏览器从附近的代理服务器中下载资源，而不必从远程原始服务器中下载。

**13.尽量减少请求的数据量**

尽量减少 Cookie 和请求标头的大小，可确保将 HTTP 请求放入单个数据包中。

**14.尽量减少重定向**

尽量减少从一个网址到另一个网址的 HTTP 重定向次数，以减少额外的往返时间 (RTT) 和用户等待的时间。

**15.指定图片大小**

为所有的图片指定宽度和高度，可消除进行不必要重排与重绘的需求，从而加快图片的显示速度。

**16.提供压缩后的图片**

适当地调整图片的大小，可节省大量的数据字节空间。

**17.由同一网址提供资源**

通过唯一的网址发布资源，这样才能避免下载重复的字节以及产生额外的往返时间 。

**18.请指定一个“Vary: Accept-Encoding”标头**

指示代理服务器缓存资源的两个版本：压缩版与未压缩版。这样有助于避免公共代理无法正确检测 Content-Encoding 标头的问题。

**19.请指定缓存验证工具**

通过指定缓存验证工具（Last-Modified 或 ETag 标头），您可以确保系统能够有效地确定缓存资源的有效性。

**20.避免出现错误的请求**

删除“已损坏的链接”或会导致 404/410 错误的请求，可避免发出无效的请求。

**21.首选异步资源**

错开资源的加载时间可减小网页加载时出现资源堵塞的概率。

**22.避免在 CSS 中使用 @import**

在外部样式表中使用 CSS @import 会增加网页加载时的延迟。

**23.暂缓 JavaScript 解析**

通过最大程度地减少显示网页所需的 JavaScript 数据量，并暂缓解析不需要的 JavaScript（等到需要执行时再进行解析），您可以提高网页的初始加载速度。

**24.优化图片**

适当地设置图片的格式并进行压缩，可节省大量的数据字节空间。

**25.指定字符集**

尽早为您的 HTML 文档指定字符集，可让浏览器立即开始执行脚本。

**26.避免在meta 标签中指定字符集**

**27.合并CSS、JavaScript文件**

减少HTTP请求的数量

十个在线网页测试工具：http://www.daqianduan.com/3962.html