了解 Resource Timing

By Jonathan Garbee (https://developers.google.com/web/resources/contributors/jonathangarbee)
Google Developer Expert for Web Technology

了解通过网络收集资源的阶段至关重要。这是解决加载问题的基础。

TL;DR

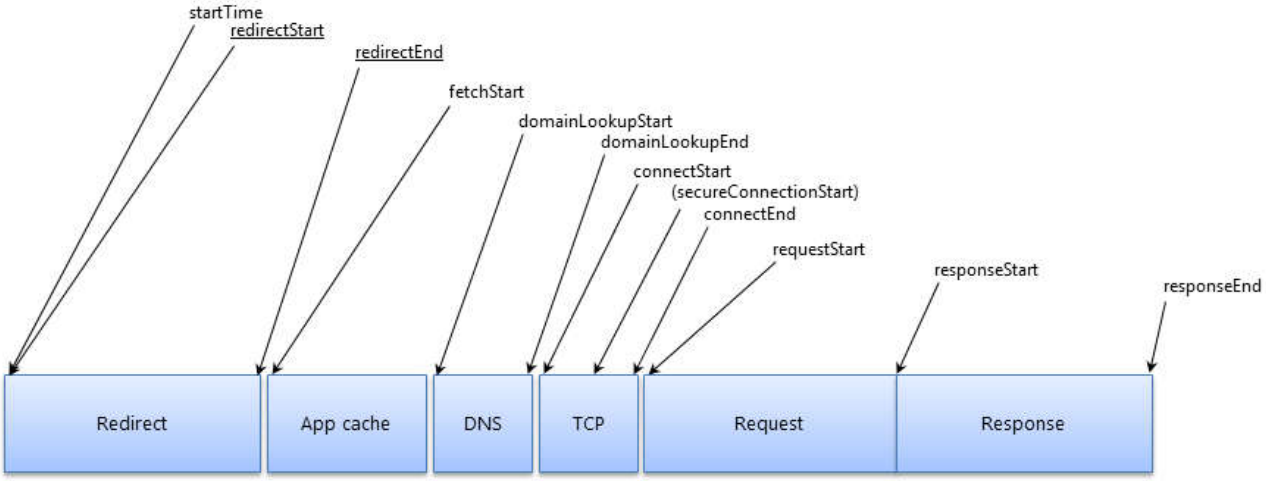
- 了解 Resource Timing 的阶段。
- 了解每个阶段向 Resource Timing API 提供的内容。
- 了解时间线图表中不同的性能问题指示器，例如一系列透明栏或者大片的绿块。

所有网络请求都被视为资源。通过网络对它们进行检索时，资源具有不同生命周期，以 Resource Timing 表示。Network 面板使用与应用开发者所用相同的 Resource Timing API (http://www.w3.org/TR/resource-timing)。

请注意：当使用具有跨源资源的 Resource Timing API 时，确保所有资源具有 CORS 标头。

Resource Timing API 提供了与接收各个资源的时间有关的大量详细信息。请求生命周期的主要阶段包括：

- 重定向
- 立即开始 `startTime`。
- 如果正在发生重定向，`redirectStart` 也会开始。
- 如果重定向在本阶段末发生，将采集 `redirectEnd`。
- 应用缓存
- 如果是应用缓存在实现请求，将采集 `fetchStart` 时间。
- DNS
- `domainLookupStart` 时间在 DNS 请求开始时采集。
- `domainLookupEnd` 时间在 DNS 请求结束时采集。
- TCP
- `connectStart` 在初始连接到服务器时采集。
- 如果正在使用 TLS 或 SSL，`secureConnectionStart` 将在握手（确保连接安全）开始时开始。
- `connectEnd` 将在到服务器的连接完成时采集。
- 请求
- `requestStart` 会在对某个资源的请求被发送到服务器后立即采集。
- 响应
- `responseStart` 是服务器初始响应请求的时间。
- `responseEnd` 是请求结束并且数据完成检索的时间。



在 DevTools 中查看

要查看 Network 面板中给定条目完整的耗时信息，您有三种选择。

1. 将鼠标悬停到 Timeline 列下的耗时图表上。这将呈现一个显示完整耗时数据的弹出窗口。
2. 点击任何条目并打开该条目的 Timing 标签。
3. 使用 Resource Timing API 从 JavaScript 检索原始数据。

此代码可以在 DevTools 的 Console 中运行。 它将使用 Network Timing API 检索所有资源。 然后，它将通过查找是否存在名称中包含“style.css”的条目对条目进行过滤。 如果找到，将返回相应条目。

```
performance.getEntriesByType('resource').filter(item => item.name.includes("style.css"))
```

```
[▼PerformanceResourceTiming
    connectEnd: 0
    connectStart: 0
    domainLookupEnd: 0
    domainLookupStart: 0
    duration: 24.014999999999986
    entryType: "resource"
    fetchStart: 697.33
    initiatorType: "link"
    name: "https://ssl.gstatic.com
```

Queuing

如果某个请求正在排队，则指示：

- 请求已被渲染引擎推迟，因为该请求的优先级被视为低于关键资源（例如脚本/样式）的优先级。 图像经常发生这种情况。
- 请求已被暂停，以等待将要释放的不可用 TCP 套接字。
- 请求已被暂停，因为在 HTTP 1 上，浏览器仅允许每个源拥有六个 TCP 连接 (https://crbug.com/12066)。
- 生成磁盘缓存条目所用的时间（通常非常迅速）

Stalled/Blocking

请求等待发送所用的时间。 可以是等待 Queueing 中介绍的任何一个原因。 此外，此时间包含代理协商所用的任何时间。

Proxy Negotiation

与代理服务器连接协商所用的时间。

DNS Lookup

执行 DNS 查询所用的时间。 页面上的每一个新域都需要完整的往返才能执行 DNS 查询。

Initial Connection / Connecting

建立连接所用的时间，包括 TCP 握手/重试和协商 SSL 的时间。

SSL

完成 SSL 握手所用的时间。

Request Sent / Sending

发出网络请求所用的时间。 通常不到一毫秒。

Waiting (TTFB)

等待初始响应所用的时间，也称为至第一字节的时间。 此时间将捕捉到服务器往返的延迟时间，以及等待服务器传送响应所用的时间。

Content Download / Downloading

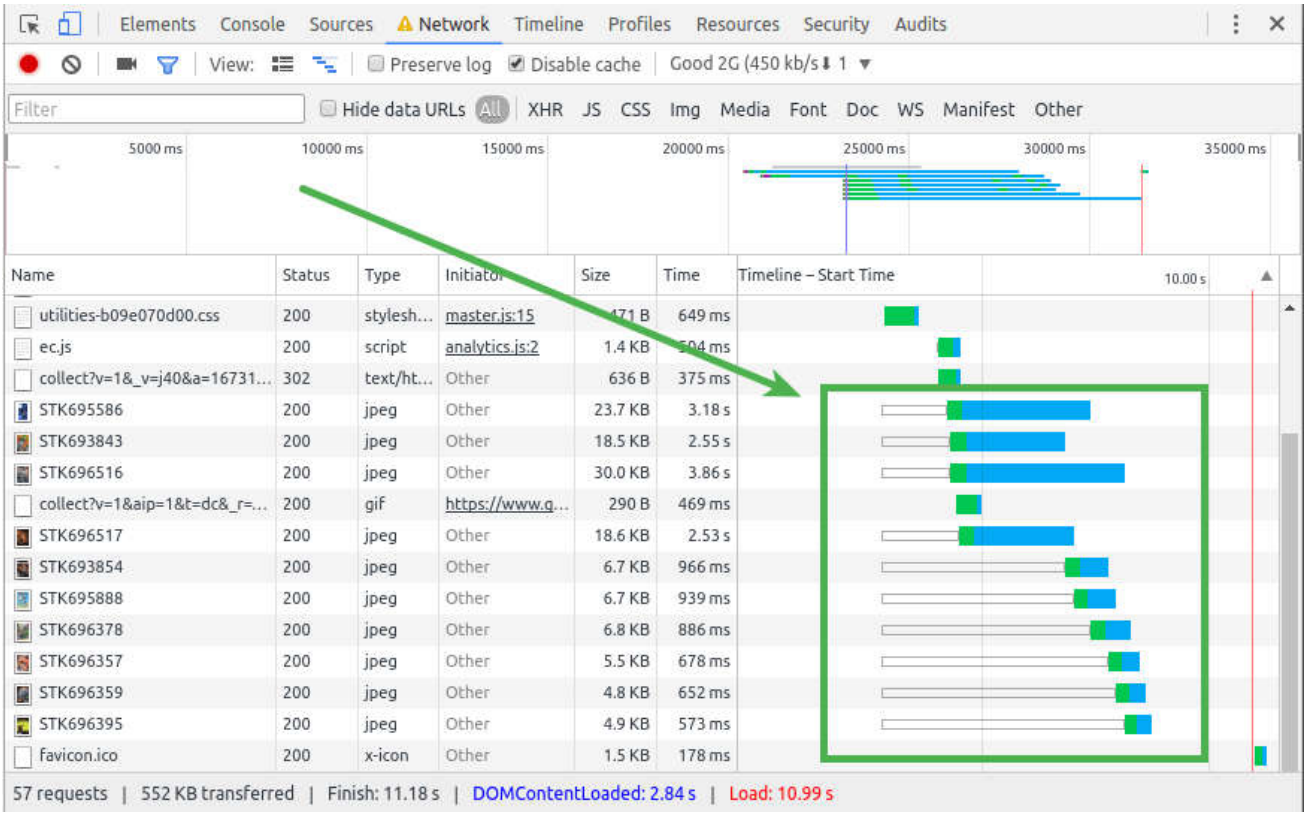
接收响应数据所用的时间。

诊断网络问题

通过 Network 面板可以发现大量可能的问题。查找这些问题需要很好地了解客户端与服务器如何通信，以及协议施加的限制。

已被加入队列或已被停止的系列

最常见问题是一系列已被加入队列或已被停止的条目。这表明正在从单个网域检索太多的资源。在 HTTP 1.0/1.1 连接上，Chrome 会将每个主机强制设置为最多六个 TCP 连接。如果您一次请求十二个条目，前六个将开始，而后六个将被加入队列。最初的一半完成后，队列中的第一个条目将开始其请求流程。

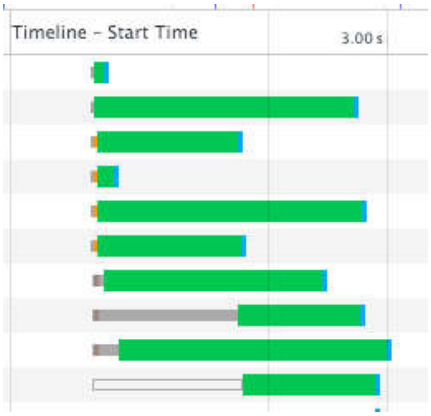


要为传统的 HTTP 1 流量解决此问题，您需要实现域分片 (https://www.maxcdn.com/one/visual-glossary/domain-sharding-2/)。也就是在您的应用上设置多个子域，以便提供资源。然后，在子域之间平均分配正在提供的资源。

HTTP 1 连接的修复结果**不会**应用到 HTTP 2 连接上。事实上，前者的结果会影响后者。 如果您部署了 HTTP 2，请不要对您的资源进行域分片，因为它与 HTTP 2 的操作方式相反。在 HTTP 2 中，到服务器的单个 TCP 连接作为多路复用连接。这消除了 HTTP 1 中的六个连接限制，并且可以通过单个连接同时传输多个资源。

至第一字节的漫长时间

又称：大片绿色



等待时间长表示至第一字节的时间 (TTFB) 漫长。建议将此值控制在 200 毫秒以下 (https://developers.google.com/speed/docs/insights/Server)。长 TTFB 会揭示两个主要问题之一。

请执行以下任一操作：

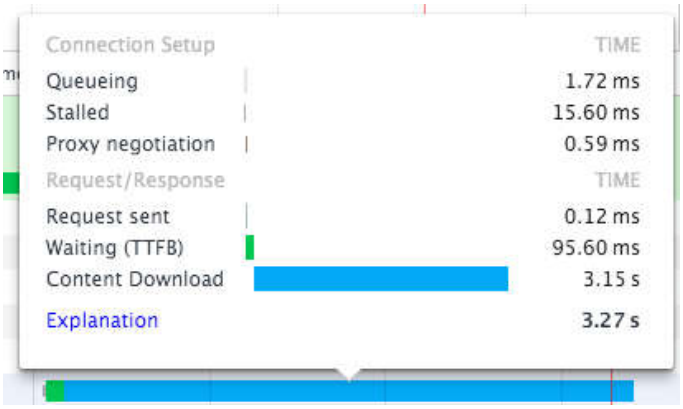
1. 客户端与服务器之间的网络条件较差，或者 2.服务器应用的响应慢

要解决长 TTFB，首先请尽可能缩减网络。理想的情况是将应用托管在本地，然后查看 TTFB 是否仍然很长。如果仍然很长，则需要优化应用的响应速度。可以是优化数据库查询、为特定部分的内容实现缓存，或者修改您的网络服务器配置。很多原因都可能导致后端缓慢。您需要调查您的软件并找出未满足您的性能预算的内容。

如果本地托管后 TTFB 仍然漫长，那么问题出在您的客户端与服务器之间的网络上。很多事情都可以阻止网络遍历。客户端与服务器之间有许多点，每个点都有其自己的连接限制并可能引发问题。测试时间是否缩短的最简单方法是将您的应用置于其他主机上，并查看 TTFB 是否有所改善。

达到吞吐量能力

又称：大片蓝色



如果您看到 Content Download 阶段花费了大量时间，则提高服务器响应或串联不会有任何帮助。首要的解决办法是减少发送的字节数。
