

## 39 | HTTP性能优化面面观（上）

2019-08-26 Chrono

《透视HTTP协议》

课程介绍 >



讲述：Chrono

时长 09:47 大小 11.21M



“透视 HTTP 协议”这个专栏已经陪伴了你近三个月的时间，在最后的这两讲里，我将把散落在前面各个章节的零散知识点整合起来，做一个总结，和你一起聊聊 HTTP 的性能优化。

由于 HTTPS（SSL/TLS）的优化已经在 [第 28 讲](#) 里介绍的比较详细了，所以这次就暂时略过不谈，你可以课后再找机会复习。

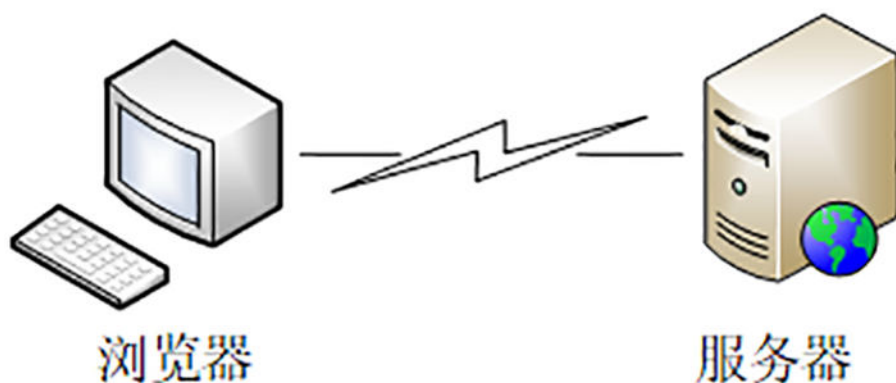
既然要做性能优化，那么，我们就需要知道：什么是性能？它都有哪些指标，又应该如何度量，进而采取哪些手段去优化？

领资料

“性能”其实是一个复杂的概念。不同的人、不同的应用场景都会对它有不同的定义。对于 HTTP 来说，它又是一个非常复杂的系统，里面有非常多的角色，所以很难用一两个简单的词就能把性能描述清楚。



还是从 HTTP 最基本的“请求 – 应答”模型来着手吧。在这个模型里有两个角色：客户端和服务端，还有中间的传输链路，考查性能就可以看这三个部分。



## HTTP 服务器

我们先来看看服务器，它一般运行在 Linux 操作系统上，用 Apache、Nginx 等 Web 服务器软件对外提供服务，所以，性能的含义就是它的服务能力，也就是尽可能多、尽可能快地处理用户的请求。

衡量服务器性能的主要指标有三个：**吞吐量**（requests per second）、**并发数**（concurrency）和**响应时间**（time per request）。

吞吐量就是我们常说的 RPS，每秒的请求次数，也有叫 TPS、QPS，它是服务器最基本的性能指标，RPS 越高就说明服务器的性能越好。

并发数反映的是服务器的负载能力，也就是服务器能够同时支持的客户端数量，当然也是越多越好，能够服务更多的用户。

响应时间反映的是服务器的处理能力，也就是快慢程度，响应时间越短，单位时间内服务器就能够给越多的用户提供服务，提高吞吐量和并发数。

除了上面的三个基本性能指标，服务器还要考虑 CPU、内存、硬盘和网卡等系统资源的占用程度，利用率过高或者过低都可能有问题。

在 HTTP 多年的发展过程中，已经出现了很多成熟的工具来测量这些服务器的性能指标，开源的、商业的、命令行的、图形化的都有。

领资料



在 Linux 上，最常用的性能测试工具可能就是 ab (Apache Bench) 了，比如，下面的命令指定了并发数 100，总共发送 10000 个请求：

```
1 ab -c 100 -n 10000 'http://www.xxx.com'
```

 复制代码

系统资源监控方面，Linux 自带的工具也非常多，常用的有 uptime、top、vmstat、netstat、sar 等等，可能你比我还要熟悉，我就列几个简单的例子吧：

```
1 top                #查看CPU和内存占用情况
2 vmstat 2           #每2秒检查一次系统状态
3 sar -n DEV 2       #看所有网卡的流量，定时2秒检查
```

 复制代码

理解了这些性能指标，我们就知道了服务器的性能优化方向：**合理利用系统资源，提高服务器的吞吐量和并发数，降低响应时间。**

## HTTP 客户端

看完了服务器的性能指标，我们再来看看如何度量客户端的性能。

客户端是信息的消费者，一切数据都要通过网络从服务器获取，所以它最基本的性能指标就是“**延迟**” (latency) 。

之前在讲 HTTP/2 的时候就简单介绍过延迟。所谓的“延迟”其实就是“等待”，等待数据到达客户端时所花费的时间。但因为 HTTP 的传输链路很复杂，所以延迟的原因也就多种多样。

首先，我们必须谨记有一个“不可逾越”的障碍——光速，因为地理距离而导致的延迟是无法克服的，访问数千公里外的网站显然会有更大的延迟。

 领资料

其次，第二个因素是带宽，它又包括接入互联网时的电缆、WiFi、4G 和运营商内部网络、运营商之间网络的各种带宽，每一处都有可能成为数据传输的瓶颈，降低传输速度，增加延迟。



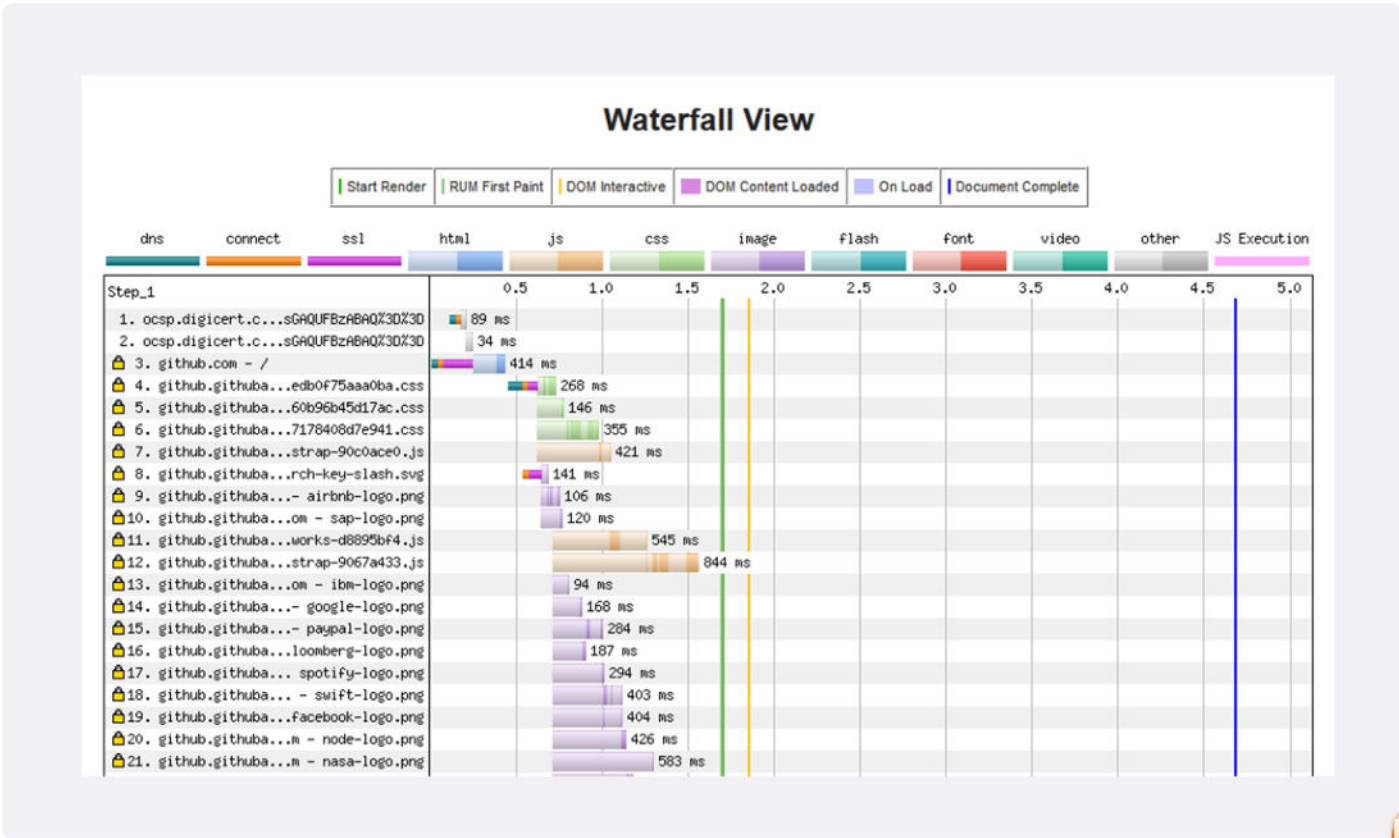
第三个因素是 DNS 查询，如果域名在本地没有缓存，就必须向 DNS 系统发起查询，引发一连串的网络通信成本，而在获取 IP 地址之前客户端只能等待，无法访问网站。

第四个因素是 TCP 握手，你应该对它比较熟悉了吧，必须要经过 SYN、SYN/ACK、ACK 三个包之后才能建立连接，它带来的延迟由光速和带宽共同决定。

建立 TCP 连接之后，就是正常的数据收发了，后面还有解析 HTML、执行 JavaScript、排版渲染等等，这些也会耗费一些时间。不过它们已经不属于 HTTP 了，所以不在今天的讨论范围之内。

之前讲 HTTPS 时介绍过一个专门的网站“[SSL Labs](#)”，而对于 HTTP 性能优化，也有一个专门的测试网站“[WebPageTest](#)”。它的特点是在世界各地建立了很多的测试点，可以任意选择地理位置、机型、操作系统和浏览器发起测试，非常方便，用法也很简单。

网站测试的最终结果是一个直观的“瀑布图”（Waterfall Chart），清晰地列出了页面中所有资源加载的先后顺序和时间消耗，比如下图就是对 GitHub 首页的一次测试。

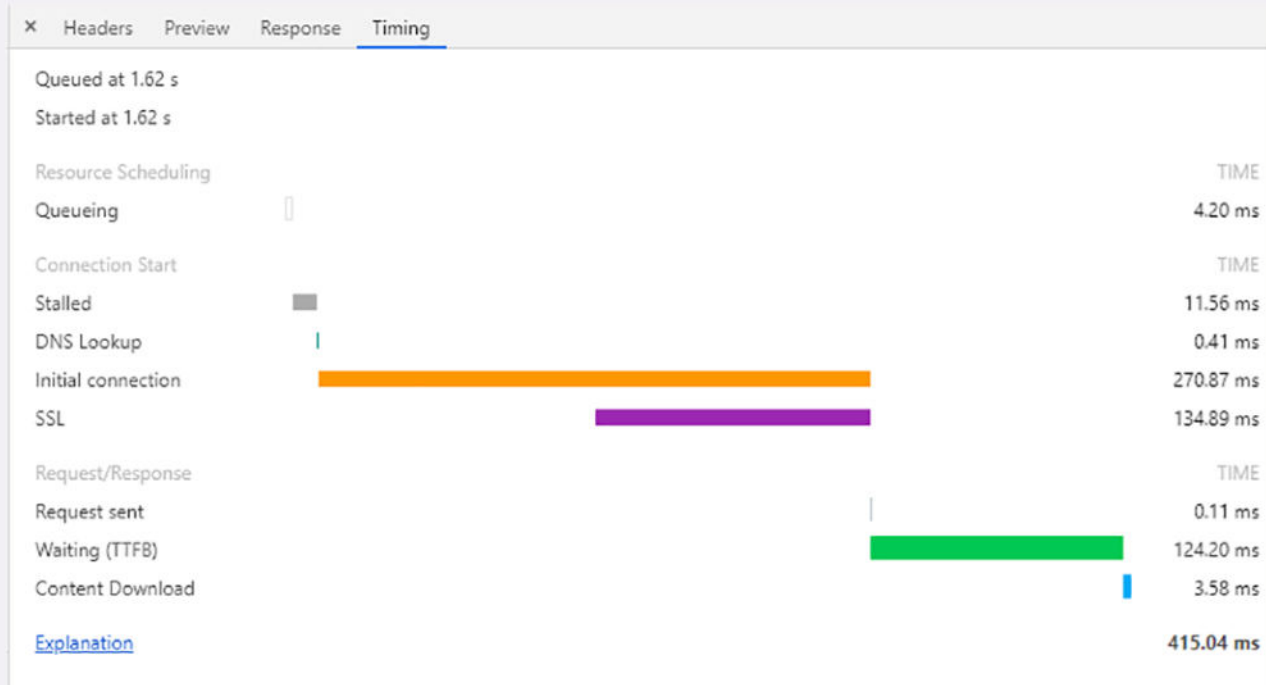


Chrome 等浏览器自带的开发者工具也可以很好地观察客户端延迟指标，面板左边有每个 URI 具体消耗的时间，面板的右边也是类似的瀑布图。

点击某个 URI，在 Timing 页里会显示出一个小型的“瀑布图”，是这个资源消耗时间的详细分解，延迟的原因都列的清清楚楚，比如下面的这张图：

领资料





图里面的这些指标都是什么含义呢？我给你解释一下：

- 因为有“队头阻塞”，浏览器对每个域名最多开 6 个并发连接（HTTP/1.1），当页面里链接很多的时候就必须排队等待（Queued、Queueing），这里它就等待了 1.62 秒，然后才被浏览器正式处理；
- 浏览器要预先分配资源，调度连接，花费了 11.56 毫秒（Stalled）；
- 连接前必须要解析域名，这里因为有本地缓存，所以只消耗了 0.41 毫秒（DNS Lookup）；
- 与网站服务器建立连接的成本很高，总共花费了 270.87 毫秒，其中有 134.89 毫秒用于 TLS 握手，那么 TCP 握手的时间就是 135.98 毫秒（Initial connection、SSL）；
- 实际发送数据非常快，只用了 0.11 毫秒（Request sent）；
- 之后就是等待服务器的响应，专有名词叫 TTFB（Time To First Byte），也就是“首字节响应时间”，里面包括了服务器的处理时间和网络传输时间，花了 124.2 毫秒；
- 接收数据也是非常快的，用了 3.58 毫秒（Content Download）。

从这张图你可以看到，一次 HTTP“请求 - 响应”的过程中延迟的时间是非常惊人的，总时间 415.04 毫秒里占了差不多 99%。

领资料

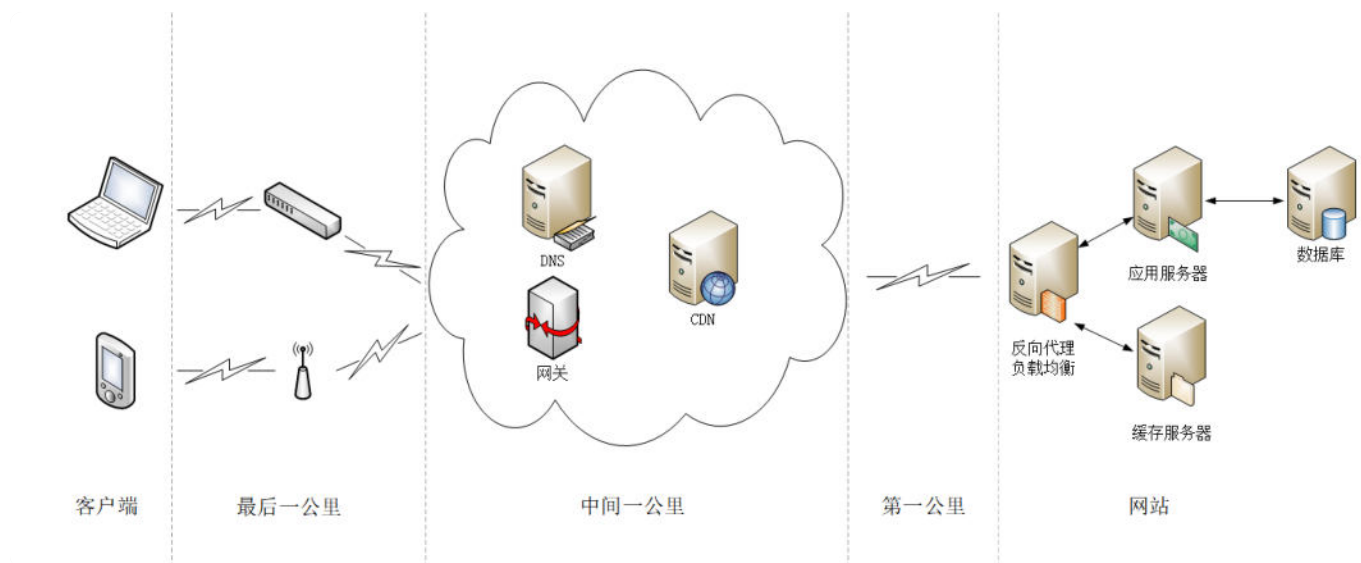


所以，客户端 HTTP 性能优化的关键就是：**降低延迟**。

## HTTP 传输链路

以 HTTP 基本的“请求 – 应答”模型为出发点，刚才我们得到了 HTTP 性能优化的一些指标，现在，我们来把视角放大到“真实的世界”，看看客户端和服务端之间的传输链路，它也是影响 HTTP 性能的关键。

还记得 [第 8 讲](#) 里的互联网示意图吗？我把它略微改了一下，划分出了几个区域，这就是所谓的“**第一公里**”“**中间一公里**”和“**最后一公里**”（在英语原文中是 mile，英里）。



“第一公里”是指网站的出口，也就是服务器接入互联网的传输线路，它的带宽直接决定了网站对外的服务能力，也就是吞吐量等指标。显然，优化性能应该在这“第一公里”加大投入，尽量购买大带宽，接入更多的运营商网络。

“中间一公里”就是由许多小网络组成的实际的互联网，其实它远不止“一公里”，而是非常非常庞大和复杂的网络，地理距离、网络互通都严重影响了传输速度。好在这里面有一个 HTTP 的“好帮手”——CDN，它可以帮助网站跨越“千山万水”，让这段距离看起来真的就好像只有“一公里”。

“最后一公里”是用户访问互联网的入口，对于固网用户就是光纤、网线，对于移动用户就是 WiFi、基站。以前它是客户端性能的主要瓶颈，延迟大带宽小，但随着近几年 4G 和高速宽带的普及，“最后一公里”的情况已经好了很多，不再是制约性能的主要因素了。

领资料





除了这“三公里”，我个人认为还有一个“第零公里”，就是网站内部的 Web 服务系统。它其实也是一个小型的网络（当然也可能会非常大），中间的数据处理、传输会导致延迟，增加服务器的响应时间，也是一个不可忽视的优化点。

在上面整个互联网传输链路中，末端的“最后一公里”我们是无法控制的，所以我们只能在“第零公里”“第一公里”和“中间一公里”这几个部分下功夫，增加带宽降低延迟，优化传输速度。

## 小结

1. 性能优化是一个复杂的概念，在 HTTP 里可以分解为服务器性能优化、客户端性能优化和传输链路优化；
2. 服务器有三个主要的性能指标：吞吐量、并发数和响应时间，此外还需要考虑资源利用率；
3. 客户端的基本性能指标是延迟，影响因素有地理距离、带宽、DNS 查询、TCP 握手等；
4. 从服务器到客户端的传输链路可以分为三个部分，我们能够优化的是前两个部分，也就是“第一公里”和“中间一公里”；
5. 有很多工具可以测量这些指标，服务器端有 ab、top、sar 等，客户端可以使用测试网站，浏览器的开发者工具。

## 课下作业

1. 你有 HTTP 性能优化的经验吗？常用的有哪些方法？
2. 你是怎么理解客户端的“延迟”的？应该怎样降低延迟？

欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。



## 课外小贴士

01 “HTTP 性能优化”是“Web 性能优化”的一部

领资料



分，后者涉及的范围更广，除了 HTTP 协议，还包含 HTML、CSS、JavaScript 等方面的优化。例如为了优化页面渲染顺序，CSS 应该放在 HTML 顶部，而 JavaScript 应该放在 HTML 的底部。

- 02 更高级的服务器性能测试工具有 LoadRunner、JMeter 等，很多云服务商也会提供专业的测试平台。
- 03 在 Chrome 开发者工具的瀑布图里可以看到有两条蓝色和红色的竖线。蓝线表示的是“DOM Ready”，也就是说浏览器已经解析完 HTML 文档的 DOM 结构；红线表示的是“Load Complete”，即已经下载完页面包含的所有资源（JS、CSS、图片等）。
- 04 还记得几年前的“光进铜退”吗？以前都是用电话线里的“铜”上网，用的是 ADSL，网速只有 10M 左右，现在都变成了“光纤入户”，网速通常都是 100M 起步。



# 透视 HTTP 协议

深入理解 HTTP 协议本质与应用

罗剑锋

奇虎360技术专家


Nginx/OpenResty 开源项目贡献者




新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

分享给需要的人，Ta订阅超级会员，你将得 **50** 元

Ta单独购买本课程，你将得 **20** 元

 生成海报并分享

 赞 7  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 38 | WebSocket：沙盒里的TCP

下一篇 40 | HTTP性能优化面面观（下）

领资料



# JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



## 精选留言 (17)

写留言



许童童

2019-08-26

你有 HTTP 性能优化的经验吗？常用的有哪些方法？

本人生产环境中会用到的：tcp fast open，DNS，HTTP缓存，DNS-prefetch

你是怎么理解客户端的“延迟”的？应该怎样降低延迟？

就是客户端与服务器一次请求响应的往返时间，降低延迟的话用DNS缓存，TCP连接复用，使用CDN，应该可以降低延迟。

作者回复: good

共 2 条评论 >

15



安排

2019-08-26

老师，再请教一个知识，ATM,帧中继这种是属于局域网技术吗？那中间一公里用到了哪些网络技术呢？底层还是以太网这种协议吗？中间一公里是不是有特殊的协议？

作者回复:

1.ATM,帧中继是比较底层的技术了，不是太了解，抱歉。

领资料



2.“中间一公里”是一种抽象化的说法，其实就是指整个互联网，也就是我们画网络图时的那朵“云”，是由许多小网络组成的，每个小网络内部可能会用不同的协议，但对外都是用ip协议连接起来的。

共 4 条评论 >

👍 4



**Jaykey**

2020-08-03

把课程中学习到的技术用到了生产上，给Nginx添加了缓存控制的头，保证了每一次上线之后资源能更新到用户的设备上，作为前端驱动运维的方式

作者回复: 学以致用，nice。



👍 3



**Wakeup**

2019-11-08

老师 我对比了下瀑布图 感觉http1的content download的时间比http2的要短，请问下为什么然后在一个连接上，并发请求很多资源，服务器的资源就不会耗尽了吗？为什么多个h2请求中间如果有h1，并发会被h1阻断

作者回复:

1.单从少数几次下载是看不出http/2的优势的，http/2的优势在多路复用、多并发，充分利用带宽。

2.因为http/2只用一个tcp连接，并发的请求只是虚拟的“流”，所以服务器端的资源消耗就少，而http/1会对一个网站创建很多连接。

3.http/2和http/1用的是彼此独立的连接，不会互相影响，不知道你说的是什么现象，我觉得可能是很多链接中混杂了http/2和http/1。



👍 3



**徐海浪**

2019-08-29

1. 你有 HTTP 性能优化的经验吗？常用的有哪些方法？

优化业务逻辑、启用缓存减少交互次数、开启压缩、按需传输(图片的裁剪)减少传输体积；后端服务器的弹性负载，确保后端服务运行正常。

另：客户端性能过剩，把一部分服务器的计算交给客户端来完成？

2. 你是怎么理解客户端的“延迟”的？应该怎样降低延迟？

客户端与服务端的交互的环节过多、环节耗时过长就会出现延迟。

服务端使用高版本的HTTP协议，在耗时长环节，用钱和空间换时间。

领资料



作者回复: 欢迎分享经验。



👍 3



安排

2019-08-26

光进铜退这里的铜是说的最后一公里吗？即使是铜的时代那中间一公里也是用的光纤吧？这叫主干网？

作者回复: 是的，主干网都是用光缆，容量大。

共 2 条评论 >

👍 3



旅途

2020-08-19

目前解决的现场问题 瓶颈在数据库 还没到http传输层面 并且这个是内网服务 访问的范围在市内 这种的话还属于老师您说的三公里模型吗

作者回复: 内网服务就相当于第零公里的问题了，需要公司内部去优化，通常不属于http协议的范围了。



👍 2



Joker

2020-04-15

在浏览器，资源文件获取还是挺耗时间的，尽量把静态资源资源文件合并，css文件，js文件这些能合并就合并，图片能拼就拼，能用CDN，绝不用服务器。

作者回复: 一定要注意资源合并是http/1的优化方式，在http/2里绝对不要用。



👍 1



渴望做梦

2019-09-02

老师，为什么因为队头阻塞所以浏览器允许只能并发6个请求呢？

作者回复: 队头阻塞与最多6个并发连接没有关系。

最多6个并发连接是rfc标准的规定，为了防止客户端并发太多连接，耗尽服务器的资源。

领资料



队头阻塞是因为http的请求应答模式，多个请求必须顺序排队，所以队头会阻塞整个队列。

共 3 条评论 >

👍 1



闫飞

2019-08-26

中间一公里的说法很容易引起混淆，尤其是对熟悉通信网复杂性的工程师而已更加费解。

作者回复: 这是CDN领域里常用的一种说法，这里借用了一下，我个人觉得还是很形象的，简化了网络模型。



👍 1



功夫熊猫

2021-10-31

感觉还有好多比如数据库的读写，线程（协程）的并发通信以及锁的粒度也可能影响速度，还有静态资源在浏览器的渲染也挺费时间的。

作者回复: http优化涉及的环节很多，有的是与http协议无关的，要是都说出来可能不太现实，欢迎大家补充。



👍



joel

2021-10-22

老师，请教一个关于最多6个并发连接的问题。

假如现在有10个资源，一个tcp 最多支持6个请求，那当第7个请求的时候，能不能在重新开一个tcp 链接，然后继续请求资源。还是说只能在现有的tcp 链接上等待。

作者回复: 最多6个并发连接是浏览器的限制，如果是我们自己写客户端，可以完全不受限制，开多少都可以。



👍

领资料



胡戎

2021-05-28

“总时间 415.04 毫秒里占了差不多 99%。”

我一直在怀疑jmeter等测试工具测试的响应时间指标是否准确，如webgis系统。

老师有没有什么方法可以准确的测出用户从请求到渲染排版结束正常显示的时间，特别是多用户。





作者回复: 这个属于前端的问题了, 不是http协议, 我也不是很擅长前端, 抱歉帮不上忙, 不过看Chrome的开发者面板应该还凑合吧。



果汁

2020-05-21

老师我想问一下那个并发6个请求的问题, 应该不只是6个XHR请求吧, 其中也包括请求js、css文件这些请求

作者回复: http请求连接数与内容无关, 任何请求都会限制在最多6个连接里。



钱

2020-04-02

性能优化的思路是相通的, 不过具体到每一个细分的领域具体措施又是不一样的。作为业务研发除代码层、系统设置层、软件系统架构层的性能优化, 感觉其他的涉足的都比较少, 感觉无能为力, 比如: 带宽不够、机器不好这些公司级别的人物才能推动。

水管理论, 找“细管”, 然后换“粗管”, 如果换不了基本没辙了。

作者回复: 学的很快啊, 都到最后这几节了。

共 2 条评论 >



子杨

2020-02-23

老师好, 想请问对于 HTTP 的首字节优化可以从哪些地方入手呢? 根据 Chrome 的分析来看, 从响应开始, 首字节占据了很大一部分时间。

作者回复: 缓存是关键, 比如外部的cdn, 内部的redis、memcache, 剩下的就是内部系统优化了。



领资料



业余草

2019-09-03

ab测试, 就是破坏rfc的标准。这个标准服务器端是没法限制的, 只能靠客户端的自觉性。

作者回复: 是的, 不过因为是测试, 所以就不用太在意了。



