HTTP 协议

基础 I

超文本传输协议（英语：**H**yper**T**ext **T**ransfer **P**rotocol，缩写：**HTTP**）是一种用于分布式、协作式和超媒体)信息系统的应用层协议。HTTP是万维网数据通信的基础。

HTTP 定义了 Web 组件（例如浏览器或命令行客户端）、Apache 或 Nginx 等服务器以及负载等代理软件之间的消息结构。本文旨在梳理我对于http 协议的知识，以备不时只需。

我会分章节对HTTP 协议内容进行整理介绍，包含我所了解的专业网站链接和遇到的问题。

协议概述

HTTP是一个客户端（用户）和服务端（网站）之间请求和应答的标准，通常使用TCP协议。通过使用网页浏览器、网络爬虫或者其它的工具例如: curl ,requests 库，客户端发起一个HTTP请求到服务器上指定端口（默认端口为80）。我们称这个客户端为用户代理程序（user agent）。应答的服务器上存储着一些资源，比如HTML文件和图像。我们称这个应答服务器为源服务器（origin server）。在用户代理和源服务器中间可能存在多个“中间层”，比如代理服务器、网关或者隧道（tunnel）。

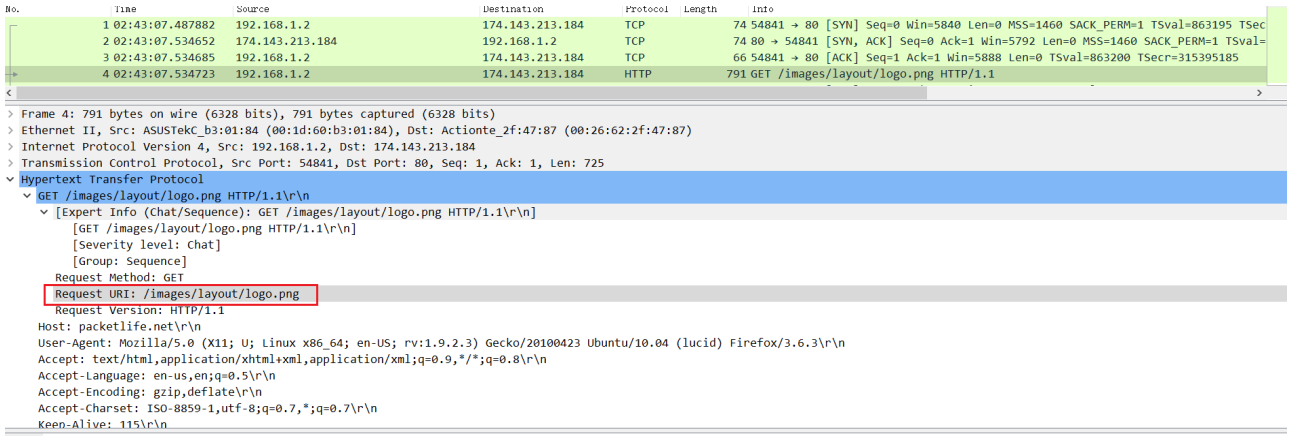
尽管TCP/IP协议是互联网上最流行的应用，但是在HTTP协议中并没有规定它必须使用或它支持的层。事实上HTTP可以在任何互联网协议或其他网络上实现。HTTP假定其下层协议提供可靠的传输。因此，任何能够提供这种保证的协议都可以被其使用，所以其在TCP/IP协议族使用TCP作为其传输层。

通常，由HTTP客户端发起一个请求，创建一个到服务器指定端口（默认是80端口）的TCP连接。HTTP服务器则在那个端口监听客户端的请求。一旦收到请求，服务器会向客户端返回一个状态，比如"HTTP/1.1 200 OK"，以及返回的内容，如请求的文件、错误消息、或者其它信息。

World Wide Web 万维网 (WWW)，通常称为 Web，是一种信息系统，可以通过 Internet 访问文档和其他 Web 资源。web历史

有三个构成web 非常重要的组件: URI、HTML、HTTP ，本文将围绕这三个组件进行梳理。

- URI 是统一资源标识符。将 URI 视为一个指针或指向。 URI 是一个简单的字符串，由三部分组成：协议、服务器和资源。参考[URL与URI的区别和联系。 URL，即Uniform Resource Locator的缩写，实际上是URI的一种形式，但在大多数情况下，它们可以互换使用。
- HTML 是超文本标记语言的缩写。它基于更通用的 SGML 或标准通用标记语言设计。 HTML 允许内容创建者提供结构、文本、图片和文档链接。在我们的上下文中，也是HTTP 协议传输的主要内容(载荷)。
- 如前所述，HTTP 是在 Web 上传输资源的最常见方式。它的核心功能是交换消息的请求/响应关系。 HTTP/1.1 版本中的 GET 消息示例如下图所示。



我们将对HTTP 协议字段进行介绍，千万不要忘了每个头部字段后面紧跟的\r\n 也是CRLF 回车换行符。

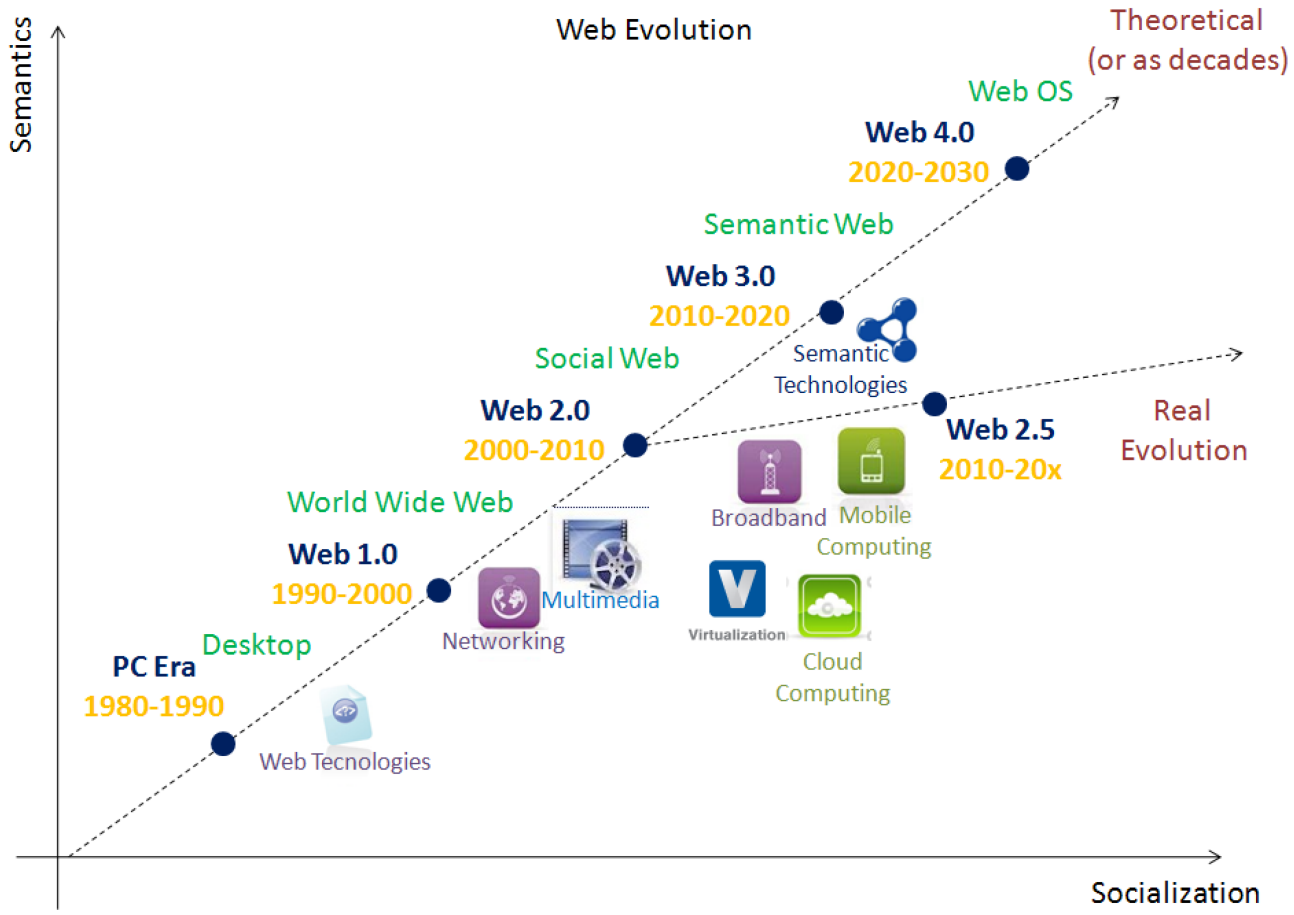
HTTP 版本历史

虽然 HTTP/2 已经完成了两年多，但目前的使用量正在增长，但不到 20%，其HTTP/1.1 一直处于主导地位。我们将在本系列的后面部分介绍特定于版本的细微差别，但整个web历史上的主要版本是：

- HTTP/0.9 - 1990
- HTTP/1.0 - 1996
- HTTP/1.1 - 1999
- HTTP/2 - 2015

  您可以查看 wiki 获取更多内容HTTP HTTP/2 是一个二进制协议，之前的HTTP 协议都是文本协议。二进制协议和文本协议您可以参考二进制协议与文本协议

web 发展历史



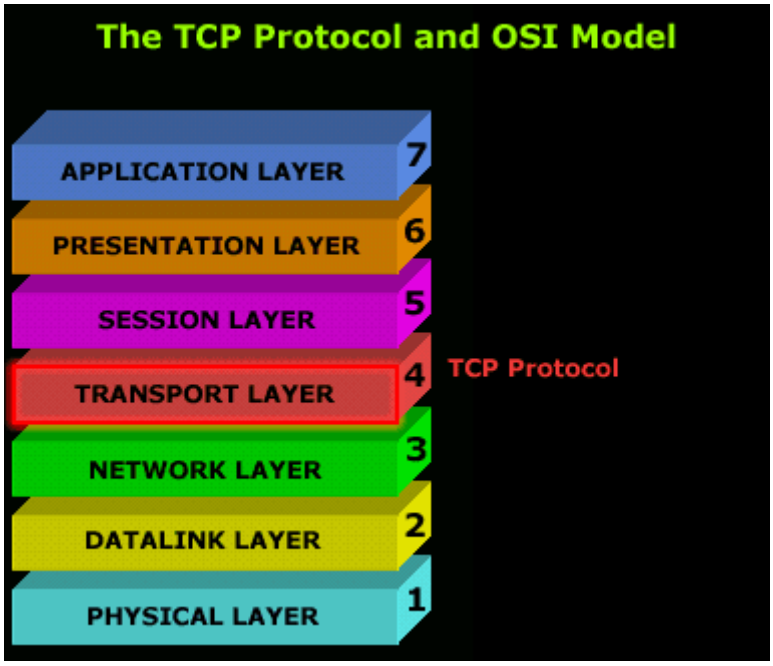
网页技术与标准

底层协议与分层概念 II

不论在生活中还是工作中我们都会遇到许多层级的概念，在开放的互联网世界中也是随处可见的，例如 OSI 模型、DNS 树状结构、以及很多路由协议的设计都存在这种概念。

OSI模型

前面我们大致了解了HTTP 协议，HTTP 协议定义了Web 资源的消息传输结构，但是这一切都需依赖底层的数据传输。



来源

众所周知，每个协议在 OSI 模型中都有自己的位置。 OSI 模型表示协议的复杂性和智能性。一般来说，OSI 模型越高，协议就越智能。该层的定位也反映了它们对 CPU 的密集程度，而 OSI 模型的较低层则完全相反，即 CPU 密集度较低且智能程度较低。

OSI 模型中的前两层，即物理层和数据链路层，被合并到 TCP/IP 模型的网络层中。将网络层视为河流和船舶、道路和汽车、铁路和火车，或者飞机和辽阔的天空。最底层数据传输的手段对 HTTP 来说并不重要。它不关心或需要知道网络层如何进行交互，所以无论是帧中继、ATM、sonet、dwdm、以太网等.....它都不关心。

网络层(IP层)

网络层（OSI 模型）或互联网协议（IP）层（TCP/IP 模型）是下一层。网络层通过路由协议（英语：Routing protocol 是一种指定数据包转送方式的网络协议）传输数据。现在，从消息传递的角度来看，HTTP 仍然不关心这个，但你作为管理员或服务器的配置需求需要关心相关信息，例如源和目标信息。

互联网它不是一个中央控制的有机体，它是由数千家公司建造的，但是对于全球大型ISP的一些了解，头部运营商相较于小型运营商具有很大的话语权。ISP拥有成千上万的设备数据传输策略纷繁复杂，但是它的实际工作方式是“尽力而为”。

传输层(TCP 层)

传输控制协议，或者我们从现在开始提到的 TCP，是当今网络世界上最重要和最知名的协议之一。它用于全球各种类型的网络，使数以百万计的数据传输能够到达目的地并充当桥梁，将主机相互连接并允许它们使用各种程序来交换数据。

TCP 由 RFC 793 定义，并于 1981 年底被引入世界。创建这种协议的动机是早在 80 年代初，计算机通信系统在军事、教育和正常的办公环境。因此，需要创建一种机制，在各种介质上进行稳健、可靠和完整的数据传输，而不会造成巨大损失。

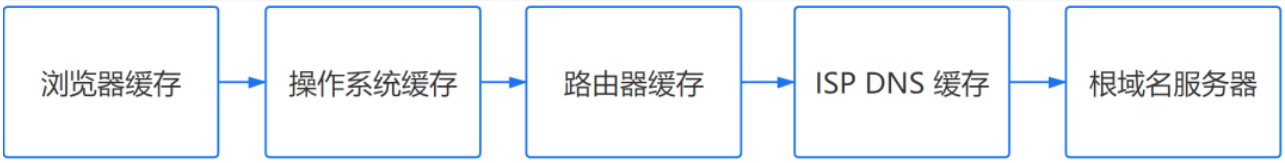
但是对于HTTP 协议来说我们关注的东西也不是太多，例如:源目地址和端口，有一种场景您可能在云环境中见过，就是四层负载软件通过对TCP 协议中OPTION 扩展字段进行填充传递客户端地址的场景。

域名系统(DNS)

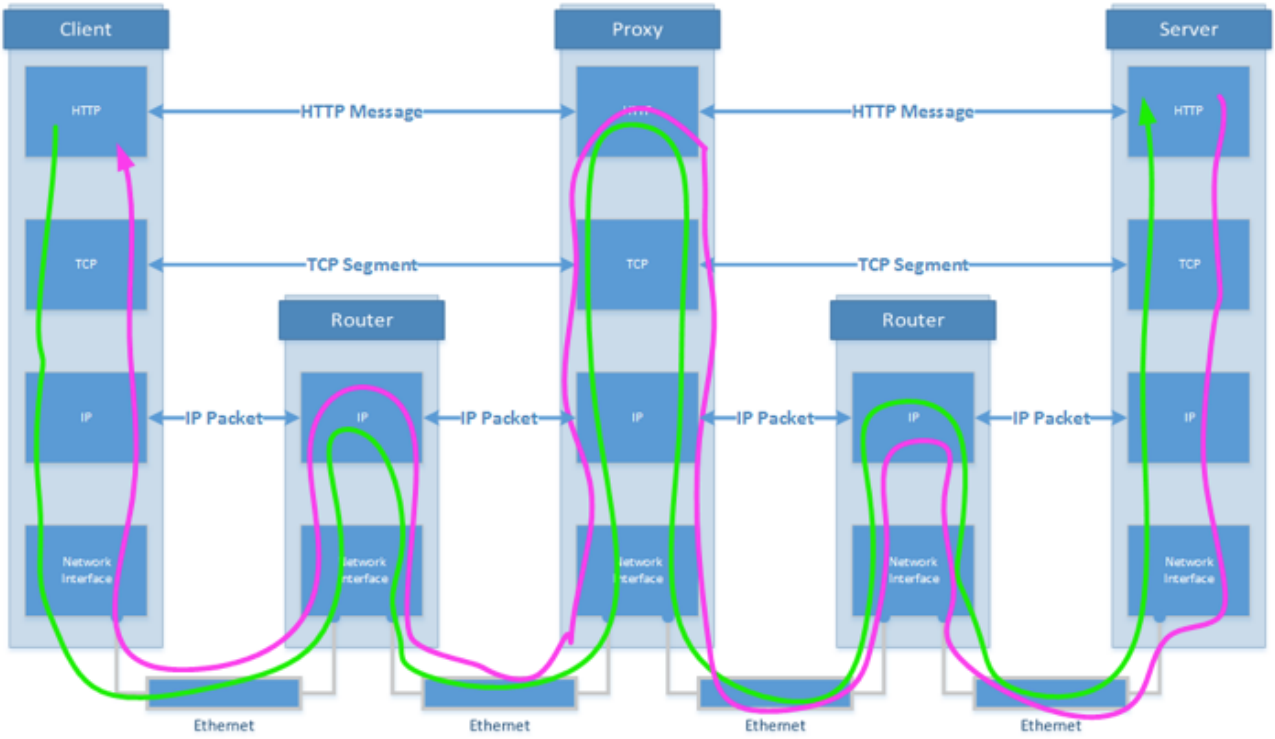
DNS 不是一个层，而是与 HTTP 相结合的关键应用程序来协助 Web 工作。DNS 旨在帮助人们从简单易记的字符到IP地址进行转换，对于IPv6 来说记住地址变成了噩梦。 DNS 或域名系统的存在是为了将名称映射到数字。 DNS 与它所依赖的 IP 基础设施一样，是在域的分层架构中由单独控制或集中控制分布式系统组合而成，DNS 系统根据组织机构的不同，有各种各样的部署架构和管理方式，也是一个值得花费时间仔细了解的好东西。

组合在一起

当您在浏览器的定位器栏中键入https://home.cnblogs.com/u/zy09/时，第一个任务不是向我们的服务器发送 IP 数据包。首先，它对该名称进行 DNS 查找，检查您浏览器缓存、系统的缓存，然后查找您服务器上配置的本地 DNS 服务器。届时，LDNS 服务器将递归您的请求进行查找并检索该答案，然后将其返回到您的系统，该答案将被缓存一段时间（希望在指定的生存时间或更短的时间内）。



那时，您的客户端系统将尝试和服务器去建立会话，并发送您的HTTP 请求。 HTTP 流量的简单流程如下图所示



总结

HTTP 协议的传输依赖底层网络，但是却并不关注它们，网络的建设也是一项昂贵且复杂的伙计，尤其搬运那些几百斤的设备，真的痛苦。如果您对浏览器如何工作很有兴趣推荐您阅读 MDN https://web.dev/howbrowserswork/ 。

术语 III

现在我们开始了解HTTP 协议，首先我们需要对于HTTP 协议相关术语进行了解。

- 万维网——或 WWW，或简称为“Web”——全球互连的计算机系统资源集合。
- 资源 ——由 URI 标识的对象或服务。网页是一种资源，但网页的图像、脚本和样式表也可以是资源。
- 网页——通过 URI 在 Web 上可访问的文档。
- 网站 ——网页的集合。
- Web客户端—— 通过生成、接收和处理 HTTP 消息来请求网站资源的软件应用程序。 Web 客户端始终是发起者。
- Web—— 服务器 - 通过接收、处理和生成 HTTP 消息为网站资源提供服务的软件应用程序。 Web—— 服务器不会向客户端发起流量。例如 Apache、NGINX、IIS.
- URI—— 统一资源标识符- 正如名称中所明确指出的，URI 是一个标识符，它可以表示资源名称或位置，或两者兼而有之。
- Message 消息——这里是HTTP 基本的通信单元
- Header标头——这是 HTTP 消息中的控制部分
- Entity 实体 —— 这是 HTTP 消息的正文
- User Agent ——用户代理 -代理用户请求。

- Proxy 代理 ——充当中间人的方式。服务器到客户端，客户端到服务器，代理必须能处理HTTP 消息——我们将在下一篇文章中深入探讨代理。
- Cache缓存—— 这是可以存在于服务器、任意数量的中间代理、浏览器上的 Web 资源存储。或以上所有。缓存的目标是减少网络上的带宽消耗，减少服务器上的计算资源利用率，并减少客户端上的页面加载延迟。
- Cookie ——最初用于添加管理状态（因为 HTTP 本身是无状态协议），cookie 是 Web 客户端按照 Web 服务器的指示存储的一小段数据。
- RFC——协议规范，HTTP 协议规范https://www.rfc-editor.org/rfc/rfc9110.html。

基础消息格式

Requests 请求格式

```
request-line
headers
CRLF (carriage return / line feed)
message body (optional)
```

示例:

```
GET /images/layout/logo.png HTTP/1.1
Host: packetlife.net
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.3)
Gecko/20100423 Ubuntu/10.04 (lucid) Firefox/3.6.3
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie:
__gads=ID=be651be1986ac2a7:T=1269487862:S=ALNI_MZdKlZ_XhLKVjt8GOzrM
TlwEKOuAQ;
__utma=116878343.438472518.1269487857.1275669543.1275673440.259;
__utmz=116878343.1275503662.251.26.utmcsr=google|utmccn=
(organic)|utmcmd=organic|utmctr=path%20mtu%20discovery%20packetlife
; sessionid=c9f137b8fff0639404c94d55032ca85e
```

CRLF（存在于 tcpdump 捕获中，但此处未显示）表示标头的结尾，并且没有正文。请注意，在请求行中，您会看到客户端的请求方法、URI 和 http 协议版本。

Responses 响应格式

```
status-line
headers
CRLF (carriage return / line feed)
message body (optional)
```

示例:

```
HTTP/1.1 200 OK

Server: nginx/0.8.34

Date: Fri, 04 Jun 2010 18:43:08 GMT

Content-Type: image/png

Content-Length: 22612

Last-Modified: Wed, 30 Sep 2009 02:12:49 GMT

Connection: keep-alive

Keep-Alive: timeout=20

Expires: Sat, 04 Jun 2011 18:43:08 GMT

Cache-Control: max-age=31536000

Cache-Control: public

Accept-Ranges: bytes


body 内容.........
```

与 HTTP 请求的请求行一样，服务器的协议在 HTTP 响应状态行中说明。还说明了响应代码，在本例中为 200 ok。您会注意到，在这种情况下，请求和响应之间唯一相似的标头是 Connection 标头。

HTTP 请求方法

URI 是客户端想要与之交互的资源。请求方法提供了客户端希望与资源交互的“方式”。请求方法有很多，但我将重点介绍最流行的几种方法。

- GET - 客户端使用此方法从服务器检索资源。
- HEAD - 与 GET 方法类似，但仅检索将响应的头部信息不包含body 内容 这对于监控和故障排除很有用。
- POST - 主要用于通过服务器上的进程处理程序创建或更新对象，将数据从客户端上传到服务器。出于安全考虑，通常对谁可以执行此操作、如何执行以及允许的更新大小有限制。
- PUT - 通常用于替换更新资源的内容。
- DELETE - 此方法删除资源。
- PATCH - 用于修改但不替换资源的内容。

HTTP Headers

有可以应用于请求和响应的通用标头，然后根据是请求还是响应消息，还有特定的标头。请注意，支持的 HTTP/1.0 和 HTTP/1.1 标头之间存在差异，本系列的最后介绍 HTTP/2。前文中的示例不完整，并不表示实际情况。 RFC 2616 记录了所有 HTTP/1.1 定义的标头。

General Headers

这些标头可以出现在请求或响应中。从概念上讲，它们处理更广泛的客户端/服务器会话，例如计时、缓存和连接管理。上面请求和响应消息中的 Connection 标头是一个通用连接管理标头的示例。

Request Headers

这些标头仅用于请求，用于通知服务器（和代理）有关首选响应行为（可接受的编码）服务器的约束（内容或主机定义的范围）有条件的请求（资源修改时间戳）和客户端配置文件（用户代理、授权）。上面请求消息中的主机标头是一个将服务器限制为该身份的请求标头示例。

Response Headers

与请求一样，响应标头仅用于响应，并用于安全（身份验证问题）缓存（计时和验证）信息共享（识别）和重定向。上面响应消息中的服务器标头是一个示例标识服务器的响应标头。

Entity Headers

实体标头的存在不是为了提供请求或响应消息传递上下文，而是为了提供有关消息正文或有效负载的特定标识。例如，上面响应消息中的 Content-Type 标头指示客户端响应的有效负载只是图片，应该以图片方式呈现。

MIME 媒体类型（通常称为 **Multipurpose Internet Mail Extensions** 或 **MIME** 类型）是一种标准，用来表示文档、文件或字节流的性质和格式。它在IETF RFC 6838中进行了定义和标准化。

关于 MIME 类型的注意事项 - Web 客户端/服务器在很大程度上是“愚蠢的”，因为它们不会根据分析猜测内容类型，它们通过 Content-Type 标头遵循消息中的说明。

HTTP Response Status Codes

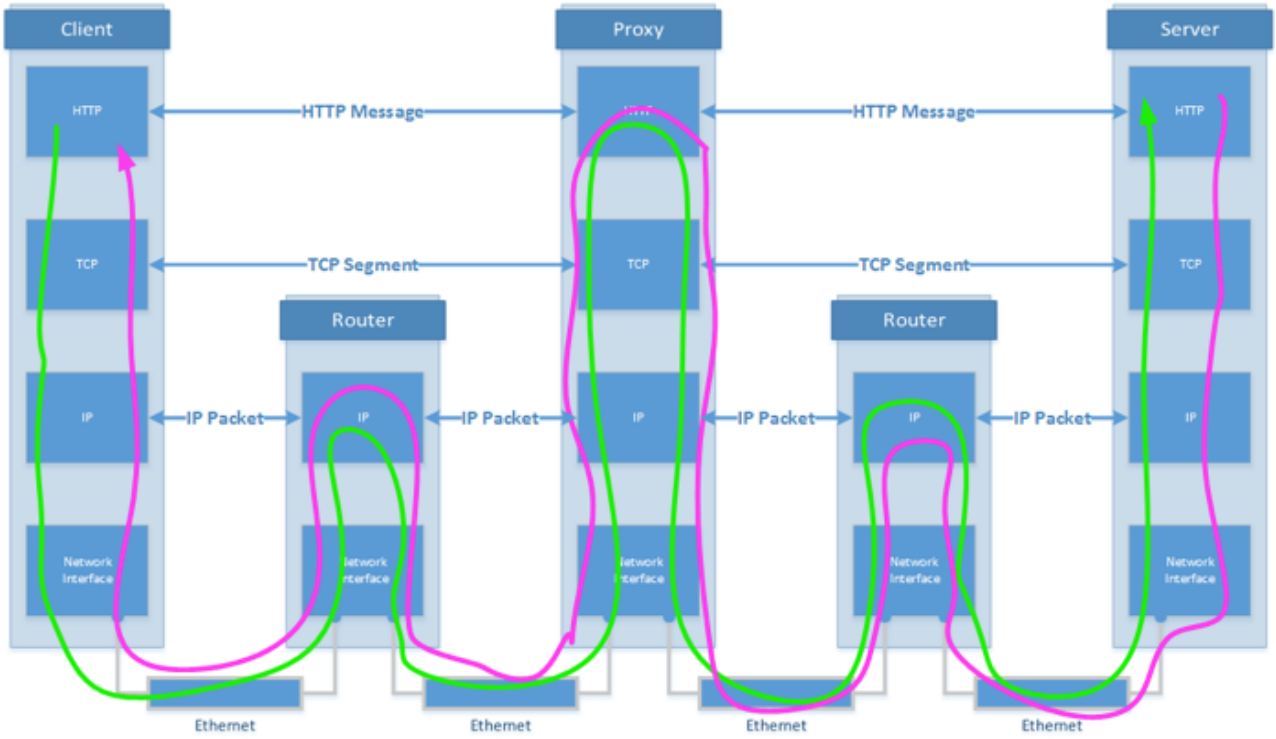
状态码: HTTP/1.1 中识别了 5 个类别和 41 个状态代码。

- Informational - 1xx - 为 HTTP/1.1 添加此类别。用于通知客户端已收到请求并且初始请求（可能是 POST 数据）可以继续,例如 websocket 的101
- Success - 2xx - 用于通知客户端请求已成功处理。
- Redirection - 3xx - 已收到请求，但需要以不同的方式处理资源。
- Client Error - 4xx - 客户端处理出现问题（资源错误、身份验证错误等）
- Server Error -5xx - 服务器端出现问题。

应用程序监视器更关注 5xx 错误。安全从业者关注 4xx/5xx 错误，但是如果出现业务层面的问题那就需要关注 2xx/3xx 消息。

客户端、服务端和代理

本章我们将介绍客户端服务端和代理三个方面的内容，以及他们和http 协议的关系。



每一段旅程都有起点和终点，但是只要和你一起那么再远的路都是美好的回忆，HTTP 协议也是如此。

```
[c:\~]$ telnet 127.0.0.1 8000


Connecting to 127.0.0.1:8000...
Connection established.
To escape to local shell, press 'Ctrl+Alt+]'.
GET /index/ HTTP/1.1

HTTP/1.1 200 OK
Date: Mon, 05 Sep 2022 10:35:42 GMT
Server: WSGIServer/0.2 CPython/3.7.4
Content-Type: text/html; charset=utf-8
X-Frame-Options: DENY
Vary: Cookie
Content-Length: 4037
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Set-Cookie:  csrftoken=MLdF8wjYiDzrSCwC4DVJTzHzy1zv7nZXpBVPYshLJ0FMM0LWaTd4lZ
 Path=/; SameSite=Lax
```

这里我用django 构建的web 服务进行测试。

在许多客户端-服务器系统中，大多数的逻辑处理都在服务端。但是目前行业发展逻辑处理的功能开始加速向 HTTP 客户端的老大 - 浏览器偏移。这些年来浏览器的发展简直令人震惊。您还记得播放视频时必须安装的大量插件吗？现在不需要了！浏览器处理名称解析、压缩、缓存、cookie、请求路由、渲染、安全等工作。浏览器是一个非常强大和丰富的应用程序平台，远远超出了本文的范围，但它仍然是一个 HTTP 客户端。

服务器可以提供静态内容，如脚本、样式表、图像等静态资源，但是目前我们常见的更多的网站都是动态交互式的网站。目前Web 系统已经变得极为庞大，应用程序服务、缓存服务、

静态资源服务、数据库服务等各种功能，但是HTTP 信息的传递的方式任然不变。

此外，使用django 这样的框架，您可以使用几行代码提供 HTTP 响应消息：

```
from django.contrib import admin
from django.urls import path
from login import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/',views.index),
    path('hello/',views.hello),

]


from  django.http import HttpResponse,request,HttpRequest
def hello(request):
    return HttpResponse('first website!')
```

除了内容管理，服务器还可以提供其他服务。一些和客户端相同的功能如压缩和缓存，以及一些附加服务，如身份验证和授权、会话处理、用户隔离等。

Proxies 代理

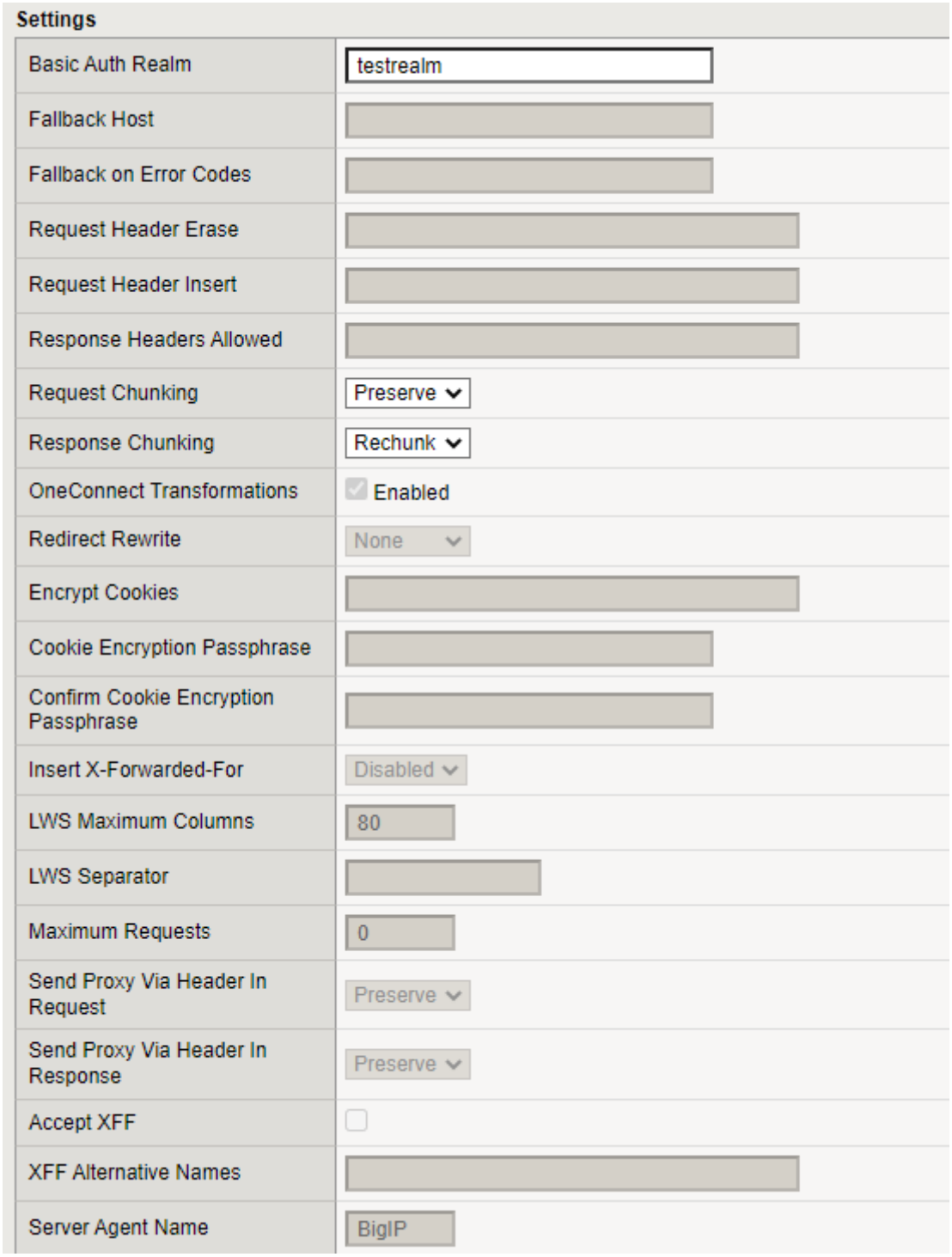
在网络和网络流量中，代理是代表其他设备的设备或服务器。它位于两个实体之间并执行服务。代理是位于客户端和服务器之间的硬件或软件解决方案，对请求和响应做一些事情。

- 缓存 - 当带宽严重受限时，代理缓存是一个常见解决方案，代理缓存将在代理本地进行检索和处理，针对请求相同资源的访问节省很多带宽。缓存可以用在正向代理和反向代理服务中。我们将在本系列后面更深入地讨论缓存。
- 转换 - 分别管理客户端和服务器可用功能。例如，维护 HTTP/1.0 客户端和 HTTP/1.1 服务器端。
- 客户端匿名 - 在服务器端您可能希望跟踪客户端的来源IP进行统计分析，在服务器部署在代理之后可能会影响您的统计，那么可以使用在 TCP 选项中插入源 IP。或者在HTTP 报文头字段插入X-Forwarded-For 标头。
- 请求/响应 检查/过滤 - 正向代理和反向代理都可以对数据进行过滤检查，上网行为管理系统作为正向代理服务可以过滤审计我们访问的网站和发送的数据，对于加密的网站一般给出的解决方案是统一在客户端推送受信任的CA 证书，访问的任何加密网站都用这个CA 临时签发网站证书。反向代理代理服务端，过滤用户到后端的请求，进行各项检查。

代理设备 **HTTP** 基本策略

代理设备、负载设备、WAF 各种各样的网络安全设备在处理 HTTP 协议数据时可以执行各种各样的操作，以及许多功能的实现也需要对应设备能够处理解析HTTP 协议，否则就无法完成对应的工作。

下面以负载设备的配置举例了解我们可以对 HTTP 协议数据做哪些操作。



Settings

SETTING	DEFAULT	DESCRIPTION
Basic Auth Realm	None	领域是源服务器上资源的保护范围。将领域视为分区，每个分区都可以拥有自己的身份验证方案。当需要身份验证时，服务器应该以 401 状态和 WWW-Authenticate 标头响应。如果将此字段设置为 testrealm，则返回给客户端的标头值为 Basic realm=“testrealm”。在客户端，当收到此响应时，将触发浏览器弹出窗口并提供一条消息，例如请输入您的用户名和密码：尽管有些浏览器没有在此消息中提供领域。

SETTING	DEFAULT	DESCRIPTION
Fallback Host	None	配置文件中的两个fallback 备用字段对于处理服务器资源池可用性告警很有用。 Fallback on Error Codes 字段允许您在以空格分隔的列表中设置多个状态码，当服务器以任何这些代码响应时，该列表将重定向到您的 Fallback Host。这有助于隐藏后端问题，获得更好的用户体验以及避免任何底层服务器信息泄漏。HTTP 请求重定向到指定的回退主机，HTTP 回复状态码为 302 Found。。
Fallback on Error Codes	None	指示来自服务器响应的 HTTP 错误代码，该代码应触发重定向到回退主机。如果您要指定多个代码，请用空格分隔代码，例如 500 501 502。您还可以指定一系列错误代码，例如：505-515。
Request Header Erase	None	系统从客户端请求中删除的 HTTP 请求标头的名称。每个池只能删除一个标头，除非您使用 自定义 策略。
Request Header Insert	None	请求标头插入是系统作为标头插入到 HTTP 请求中的字符串。如果表头已经存在，系统不会替换它。对于多个标头插入，可以使用脚本进行处理
Response Headers Allowed	None	可以指定系统允许的 HTTP 响应中的标头。如果您指定多个标题，请用空格分隔标题。例如，如果您输入字符串 Content-Type Set-Cookie Location，则系统将允许使用标头 Content-Type、Set-Cookie 和 Location。
Request Chunking	Sustain	处理HTTP 分块请求的方式，Transfer-Encoding: chunked 是一个值得详细分析的参数 - Preserve: 系统处理分块的内容并将请求原封不动地发送到服务器。 - Selective: 系统将 HTTP 内容解块，处理数据，重新添加块头，然后将分块的请求或响应发送到服务器。请注意，对于分块内容，此模式与 Rechunk 模式相同。 - Unchunk: 对于分块内容，指定系统将响应取消分块，处理 HTTP 内容，并将响应作为未分块传递。不支持 Connection 标头的 Keep-Alive 值，因此系统将标头的值设置为关闭。如果响应未分块，则系统会处理 HTTP 内容并将响应按原样传递。 - Rechunk: 系统将 HTTP 内容解块，处理数据，重新添加块头，然后将分块的请求或响应发送到服务器。 对于未分块的内容: - Preserve: 系统处理 HTTP 内容并将请求原封不动地发送给服务器。 - Selective: 系统处理 HTTP 内容并将请求原封不动地发送到服务器。 - Rechunk: 系统处理 HTTP 内容，将传输编码和块头添加到响应中，然后将分块请求发送到服务器。

SETTING	DEFAULT	DESCRIPTION
Response Chunking	Sustain	响应分块的处理 - Unchunk: 对于分块内容，指定系统将响应分块，处理 HTTP 内容，并将响应作为未分块传递。不支持 Connection 标头的 Keep-Alive 值，因此系统将标头的值设置为关闭。如果响应未分块，则系统会处理 HTTP 内容并将响应按原样传递。 - Rechunk: 指定系统将请求或响应取消分块，处理 HTTP 内容，重新添加块尾标头，然后将请求或响应作为分块传递。任何块扩展都会丢失。如果请求或响应未分块，系统会在出口处添加传输编码和分块标头。 - Sustain: 指定系统保留请求或响应分块，除非有修改正文的命令。如果请求或响应被分块，请取消对 HTTP 内容的分块，处理数据，并在出口处重新添加分块标头。块扩展将丢失。当响应被分块时，它可以在出口到客户端时重新分块。较旧的参数 Selective -分块传输编码修改 HTTP 消息的主体并将其作为一系列块传输。响应分块设置指定 BIG-IP 系统如何处理由服务器分块的 HTTP 内容。每种模式下的行为取决于服务器是发送分块响应还是非分块响应。 - Preserve: 系统处理分块的内容并将响应原封不动地发送给客户端。 - Selective: 系统将 HTTP 内容解块，处理数据，重新添加块头，然后将分块的请求或响应发送给客户端。请注意，对于分块内容，此模式与 Rechunk 模式相同。 - Unchunk: 系统去除HTTP传输编码头，去除chunk header，处理HTTP内容，然后将未分块的响应发送给客户端。系统在发送完所有数据后关闭连接。 - Rechunk: 系统将 HTTP 内容解块，处理数据，重新添加块头，然后将分块的请求或响应发送给客户端。 对于未分块的内容： - Preserve: 系统处理 HTTP 内容并将响应原封不动地发送给客户端。 - Selective: 系统处理 HTTP 内容并将响应原样发送给客户端。 - Unchunk: 系统处理 HTTP 内容并将响应原封不动地发送给客户端。 - Rechunk: 系统处理 HTTP 内容，将传输编码和块头添加到响应中，然后将分块的响应发送给客户端。
OneConnect Transformations	Enabled	启用后，系统执行 HTTP 标头转换，以允许 HTTP/1.0 连接在连接的服务器端转换为 HTTP/1.1 请求。这允许这些连接保持打开状态以供重用，否则它们不会执行连接复用。在 HTTP 配置文件中启用 OneConnect 转换设置后，系统会将 HTTP/1.0 客户端请求中的 Connection: close 标头转换为服务器端的 X-Cnection: close 标头。这允许系统发出包含 Connection: close 标头的客户端请求，例如 HTTP/1.0 请求，符合连接重用条件。仅当您为虚拟服务器配置 OneConnect 配置文件时，此设置才适用。但是目前更多场景都是基于HTTP/1.1

SETTING	DEFAULT	DESCRIPTION
Redirect Rewrite	None	客户端请求可能会从 HTTPS 协议重定向到 HTTP 协议，这是一个非安全通道。如果要确保请求保持在安全通道上，可以将 系统配置为重写重定向，以便将其重定向回 HTTPS 协议。要使 系统能够重写 HTTP 重定向，您可以使用 Rewrite Redirections 设置来指定您希望系统在重写期间处理 URI 的方式。可以使用以下选项: - None: 指定系统不重写任何 HTTP 重定向响应中的 URI。 - All: 指定系统重写所有 HTTP 重定向响应中的 URI。 - Matching: 指定系统重写与请求 URI 匹配的任何 HTTP 重定向响应中的 URI。 - Nodes: 指定如果 URI 包含节点 IP 地址而不是主机名，则系统将其更改为虚拟服务器地址。 请注意，这里重写的不是内容。仅更新重定向中的 Location 标头。
Encrypt Cookies	None	这是一个安全功能，使你能够在服务器响应中获取一个未加密的cookie，用192位AES密码进行加密，并在向客户端发送响应前对其进行b64编码。可以在一个空格分隔的列表中指定多个cookie。这也可以在脚本中针对所有的cookie进行，而不需要叫出具体的cookie名称。
Cookie Encryption Passphrase	None	输入 cookie 加密的密码。
Confirm Cookie Encryption Passphrase	None	重新输入您在 Cookie 加密密码短语框中指定的密码。
Insert X-Forwarded-For	Disabled	当使用允许客户端使用现有服务器端连接的连接池时，您可以将带有客户端 IP 地址的 X-Forwarded For 标头插入到请求中。当您将系统配置为插入此标头时，目标服务器可以将请求识别为来自发起连接的客户端以外的客户端。
LWS Maximum Columns	80	在 HTTP 请求中插入 HTTP 标头时，指定任何给定行的最大列宽。
LWS Separator	\r\n	指定当标题超过您在 LWS 最大列设置中指定的最大宽度时系统插入的线性空白 (LWS) 分隔符。
Maximum Requests	0	指定系统基于每个连接接受的 HTTP 请求数。默认情况下，系统不限制每个连接的请求数。连接复用
Send Proxy Via Header in Request	Preserve	指定是删除、保留还是附加包含在客户端请求中的Via 头到源 Web 服务器。

SETTING	DEFAULT	DESCRIPTION
Send Proxy Via Header in Response	Preserve	指定是删除、保留还是附加包含在对客户端的源 Web 服务器响应中的 Via 标头。
Accept XFF	Disabled	根据请求的 X-Forwarded-For (XFF) 标头（如果存在）启用或禁用信任客户端 IP 和来自客户端 IP 地址的统计信息。
XFF Alternative Names	None	指定受信任的替代 XFF 标头，而不是默认的 X-Forwarded-For 标头；这没有指定插入的 XFF 标头的名称。如果您指定多个替代 XFF 标头，请用空格分隔替代 XFF 标头，例如 client1 proxyserver 10.10.10.10。
Server Agent Name	Proxy	指定在系统生成的响应中用作服务器名称的字符串。如果设置为空字符串，则不会在系统生成的 HTTP 响应中插入服务器标头。注意：此设置不会修改来自后端服务器的 HTTP 响应中的服务器标头。

HTTP 安全合规策略

根据HTTP RFC 规范我们对HTTP 协议进行很多合规性检查和配置。

SETTING	DEFAULT	DESCRIPTION
Enforce RFC Compliance	Disabled	执行基本的 RFC 合规性检查，如 HTTP 协议的最新 RFC 中所述。如果客户端请求未通过这些检查，则重置连接。
Allow Space Header Name	Disabled	指定是否允许在 HTTP 请求或响应中的标头名称和分隔符冒号之间的 HTTP 标头中使用空格。启用后，系统会忽略标头名称和冒号之间的空格。
Allow Truncated Redirect	Proxy Mode (Reverse): Disabled Proxy Mode (Explicit): Disabled Proxy Mode (Transparent): Enabled	这是一个简单的切换，允许您选择如何在没有行尾 CRLF 的情况下处理重定向。默认情况下，重定向将被静默删除。

SETTING	DEFAULT	DESCRIPTION
Maximum Header Size	Proxy Mode (Reverse): 32,768 bytes Proxy Mode (Explicit): 32,768 bytes Proxy Mode (Transparent): 16384 bytes	HTTP 规范没有规定最大标头大小，在这种情况下，最大标头大小不是单个标头的大小，而是请求行和所有标头组合的大小，并且各种服务器设置的最大值不同。 BIG-IP 默认是 32k，但 Apache 是 8k，根据版本不同，nginx 是 4k-8k，IIS 是 8k-16k，Tomcat 是 8k-48k。因此，如果您在特定虚拟机上面向这些服务器中的任何一个，而不是潜在的 Tomcat，您可以显着降低配置文件默认值，而不会影响服务器。但是，存在行为差异。大多数服务器会返回 4xx 状态码，我看到提供了 400、404 和 413。你不喜欢标准吗？ BIG-IP 将简单地重置 TCP 连接。您可以在 iRules 中限制单个标头的大小，这已在为在应用程序服务器补丁之前提供 0day 缓解而编写的各种规则中完成。指定系统允许合并的所有 HTTP 请求或响应标头的最大大小（以字节为单位）。（对于 HTTP 请求，它包括请求行。）如果 HTTP 请求或响应中的组合标头长度（以字节为单位）超过此值，系统将停止解析标头并重置 TCP 连接。
Oversize Client Headers	Reject	指定客户端超过最大标头大小值时的传递行为。此设置仅在代理模式设置为透明时可用。
Oversize Server Headers	Reject	指定服务器超出最大标头大小值时的传递行为。此设置仅在代理模式设置为透明时可用。
Maximum Header Count	Proxy Mode (Reverse): 64 headers Proxy Mode (Explicit): 64 headers Proxy Mode (Transparent): 32 headers	指定系统支持的最大标头数。
Excess Client Headers	Pass Through	指定客户端超过最大标头计数值时的传递行为。此设置仅在代理模式设置为透明时可用。
Excess Server Headers	Pass Through	指定服务器超过最大标头计数值时的传递行为。此设置仅在代理模式设置为透明时可用。

SETTING	DEFAULT	DESCRIPTION
Pipeline Action	Allow	管道操作是另一个简单的切换，如果先前的请求尚未收到响应，则允许或拒绝来自客户端的新请求。你问什么是流水线？想想电话交谈，当您提出问题或发表评论时，另一端的人会做出回应。这是您在没有流水线的情况下看到的行为。现在想想你观察到的一场辩论，其中一位辩论者提出了一个由十部分组成的问题。他通常会一次完成所有操作。在他发表完所有陈述或提出所有问题后，另一位辩手将按顺序处理这些陈述和问题。这是您在流水线中看到的行为。这种管道行动设置是辩论中的主持人将介入并允许或拒绝第一位辩论者的任何陈述/问题，直到第二位辩论者做出回应。
Unknown Method	Allow	这些设置有助于消除未知的攻击向量。如果您的应用程序是一个简单的 GET/POST 应用程序，则无需允许其他任何事情。通过拒绝未知方法并禁用除 GET 和 POST 之外的已启用方法中的所有其他方法，您可以将威胁环境减少到服务器可能面临的问题。仔细管理这些设置应该意味着您与您的应用程序团队有紧密的沟通关系，因此随着应用程序的发展，不会出现在没有配置文件调整的情况下引入新方法的情况。
Known Method	CONNECT DELETE GET HEAD LOCK OPTIONS POST PROPFIND PUT TRACE UNLOCK	优化已启用方法列表中指定的已知 HTTP 方法的行为。如果您从 Enabled Methods 列表中删除已知方法，BIG-IP 系统将应用 Unknown Method 设置来管理该流量。

sFlow

SETTING	DEFAULT	DESCRIPTION
Polling Interval	Default	sFlow 是一种可扩展的数据包采样技术，可让您观察流量模式以实现性能、故障排除、计费和安全目的。 HTTP 配置文件中的两个设置是轮询间隔和采样率。轮询间隔是轮询之间的最大间隔，采样率是观察到的数据包与生成的样本的比率。因此，对于默认的 10 秒和 1024 个数据包，在 10 秒内会有两次轮询，其中系统每观察到 1024 个数据包，就会随机生成一个样本。您可以在配置文件中指定采样率和轮询间隔，也可以接受默认值，它继承了在 System->sFlow->Global Settings 下为 HTTP 指定的设置。 指定两次轮询之间的最大间隔（以秒为单位）。默认值“Default”表示在 System :: sFlow :: Global Settings :: http :: Properties 屏幕上设置的值。初始默认值为 10 秒。

SETTING	DEFAULT	DESCRIPTION
Sampling Rate	Default	指定观察到的数据包与生成的样本的比率。例如，2000 的采样率指定系统每观察 2000 个数据包随机生成 1 个样本。默认值为 Default，表示在 System :: sFlow :: Global Settings :: http :: Properties 屏幕上设置的值。初始默认值为 1024 个数据包。

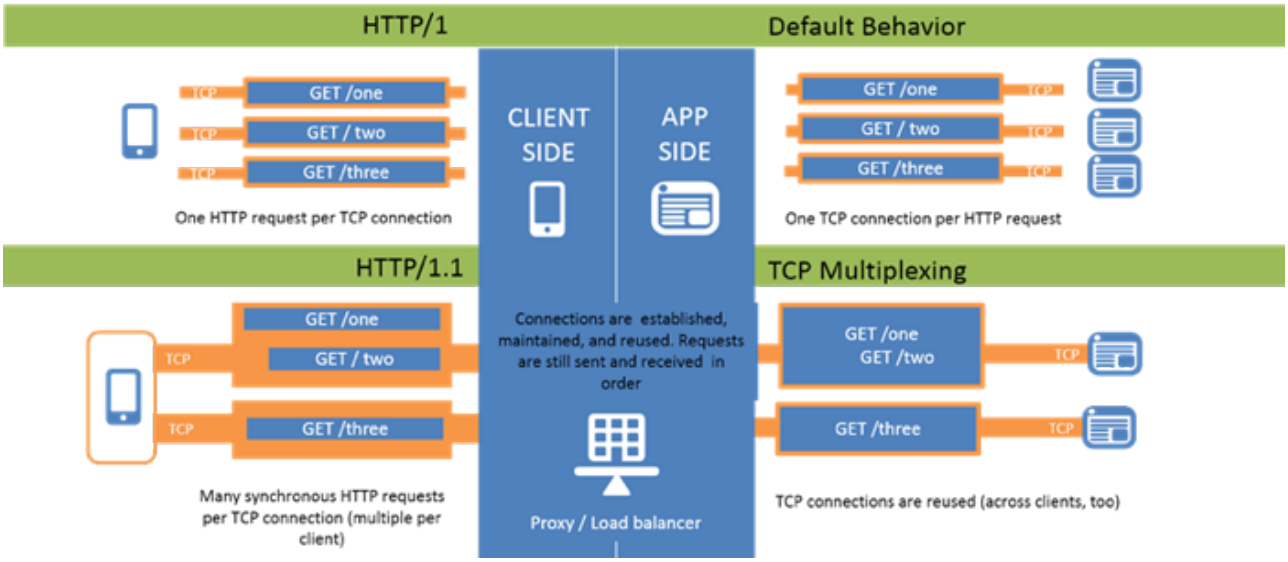
HTTP Strict Transport Security

Note:严格传输安全是一个安全标头，它指示浏览器通过 SSL 提交对该域的每个请求，即使 URL 上的协议是从服务器返回的 HTTP 而不是 HTTPS。 HSTS 是最初作为 iRule（客户端响应的简单标头插入）的设置之一，后来成为产品的一部分。您可以通过选中“模式”框来启用 HSTS。最大年龄设置以秒为单位，默认值约为六个月。如果启用，包含子域复选框将指示浏览器对子域执行相同的操作，这可能是可取的，也可能不是。

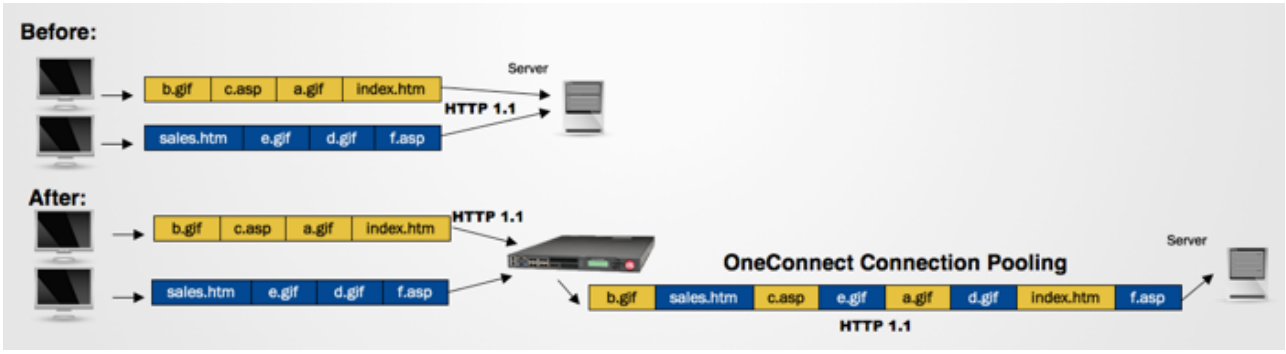
SETTING	DEFAULT	DESCRIPTION
Mode	Disabled	选中（启用）时，启用 HSTS 设置。
Maximum Age	16070400 sec	指定 HSTS 功能请求客户端仅使用 HTTPS 连接到当前主机和当前主机域名的任何子域的最大时间长度（以秒为单位）。值 0 重新启用纯文本 HTTP 访问。
Include Subdomains	Enabled	选中（启用）后，将 HSTS 策略应用于 HSTS 主机及其子域。
Preload	Disabled	选中（启用）后，将 HSTS 主机及其子域添加到浏览器的 HSTS 预加载站点列表中，这些站点仅被视为 HTTPS。默认为禁用。

什么是 **OneConnect** ？

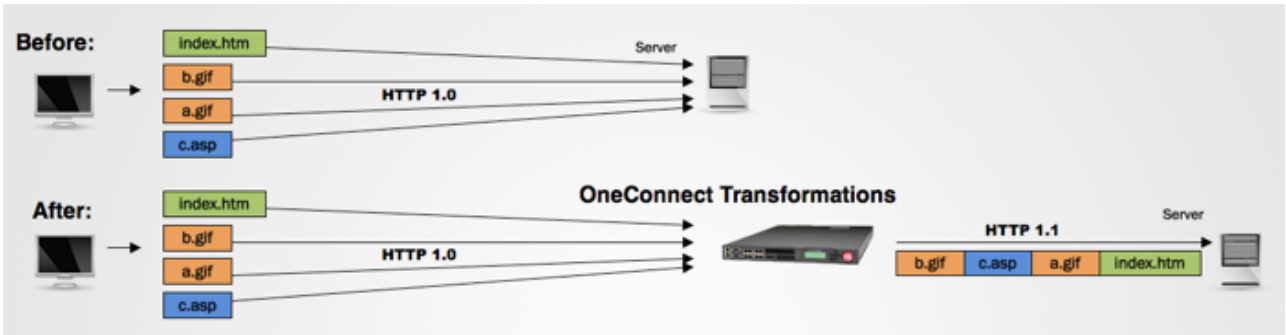
OneConnect 通过重用现有代理到服务器端连接来减少代理和应用程序服务器之间的连接数量。因此，尽管您可能在客户端有数千个连接，但代理到服务器端连接的数量将会减少，以减轻 tcp 连接监听和拆除的开销。



您可以在上图中看到，在 HTTP/1.1 之前，客户端和服务器端每个连接的单个请求会导致效率低下。使用 HTTP/1.1，您可以在客户端的单个连接上复用请求，如下所示，可以将多个客户端连接复用到单个服务器端连接上。这是 OneConnect 提供的强大功能。

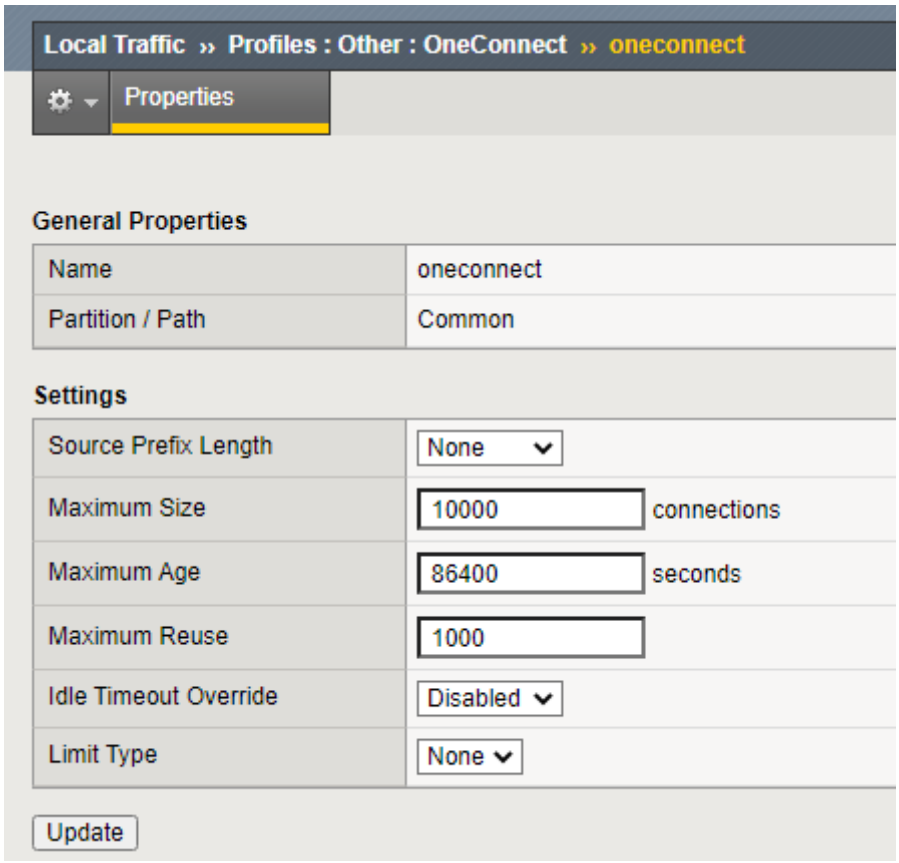


两个客户端在单个 tcp 连接中对每个客户端的多个请求在 负载 和服务器之间的单个 tcp 连接上汇集在一起。通过 OneConnect 转换，HTTP/1.0 也可以这样做。



在这种情况下，借助 OneConnect，负载能够使客户端上的四个单独连接在服务器端转换为单个连接。

OneConnect 配置文件



SETTING	DEFAULT
Source Prefix Length	在早期版本中也被称为源掩码字段，这是你指定连接重用的掩码长度的地方。重要的是要理解，这是服务器端连接的上下文源IP。这意味着，如果你使用SNAT地址，它将在为OneConnect重用资格评估掩码长度之前被应用。前缀长度可以设置为IPv4或IPv6掩码长度，并以CIDR符号而不是传统的netmasks提供。掩码本身并不复杂，掩码长度为0（默认）是最有效的，因为OneConnect会在目标服务器上寻找任何空闲的tcp连接。IPv4的/32或IPv6的/128长度是一个主机匹配，所以只有源主机IP（或SNAT）和目标服务器之间的开放连接才有资格被重复使用。
Maximum Size	最大大小设置表示连接池中保留的最大空闲服务器端连接数。
Maximum Age	设置表示从连接池中删除连接之前允许服务器端连接的最大秒数。
Maximum Reuse	最大重用设置表示通过服务器端连接发送的最大请求数。这个数字应该略低于后端服务器接受的HTTP Keep-Alive请求的最大数量，以防止后端服务器发起连接关闭并进入TIME_WAIT状态。
Idle Timeout Override	Idle Timeout Override 设置表示空闲服务器端连接保持打开的最长时间。降低此值可能会导致空闲服务器端连接数减少，但可能会增加请求延迟和服务器端连接速率。

SETTING	DEFAULT
Limit Type	限制类型设置=控制如何与 OneConnect 一起实施连接限制。可以使用以下值： None（默认）：连接计入池成员限制，具体取决于它们是否有活动的、进行中的请求或响应。
Idle: 空闲：指定当达到 TCP 连接限制时将丢弃空闲连接。对于短时间间隔，在空闲连接被丢弃和新连接建立的重叠期间，可能会超过 TCP 连接限制。	
Strict: 指定遵守 TCP 连接限制，没有例外。这意味着空闲连接将阻止建立新的 TCP 连接，直到它们过期，即使它们可以被重用。这不是推荐的配置，除非在非常特殊的情况下过期超时很短。	

了解 OneConnect 行为

OneConnect 不会覆盖负载均衡算法行为。如果有两个池成员（s1 和 s2）并且我们使用轮询负载均衡方法并应用了 OneConnect，则行为如下：

1. 第一个请求转到 s1
2. 第二个请求转到 s2
3. 第三个请求转到 s1 并重用空闲连接
4. 第四个请求到 s2 并重用空闲连接

将 OneConnect 配置文件应用于没有第七层事务性（阅读：明确定义的请求和响应）协议（如 HTTP）的虚拟服务器，通常被认为是一种错误的配置，并且在这样做时，您可能会看到奇怪的行为。可能会有例外，但在部署到生产环境之前，应该仔细考虑这种配置并进行彻底的测试。 如果在具有HTTP的虚拟服务器上没有OneConnect，你会发现持久性数据似乎并没有被兑现。这是因为在默认情况下，负载系统为每个tcp连接执行负载平衡，而不是每个HTTP请求，也就是基于流数据处理方式，因此在同一个客户端连接上的进一步请求将遵循最初的决定。通过将OneConnect应用于虚拟，你可以有效地在每次请求后将服务器端与客户端连接分离，强制进行新的负载平衡决策，并使用可用的持久性信息。

压缩和缓存

Compression 压缩

在互联网的早期，大部分的内容都是基于文本的。这意味着大部分的资源都是非常小的。随着知名度的提高，人们对丰富图像内容的渴望也在增加，数据资源也开始爆炸性增长。然而，但是带宽并不能一下满足这种需求。

更对的需求和和更大的带宽致使HTTP 协议也在不同方向进行了发展。

- 获取或发送部分资源的方法
- 确定使用哪种方法检索资源
- 在传输过程中减小资源大小的方法和收到后可以复用资源的方法

开发了各种控制标头来处理第一种情况，开发缓存来处理第二种情况，并开发压缩来处理第三种情况。数据压缩的基本定义只是减少准确表示资源所需的位数。这样做不仅可以节省网络带宽，还可以节省存储设备的空间。当然，这两个领域的都能节省资金。

在HTTP/1.0中，端到端压缩是可能的，但不是逐跳压缩，因为它没有区分代理层的机制。这在HTTP/1.1中得到了解决，所以代理层可以使用服务器或客户端不知道的复杂算法来压缩它们之间的数据，并在分别与客户端和服务器对话时进行相应的转换。

如果你使用gzip（默认），你会想注意压缩级别。默认的1级从代理上的压缩行为来看是最快的，也就是设备处理压缩的时间最短，但是做了最小的压缩，带宽变化较少。如果一个应用程序非常需要减少网络客户端的带宽利用率，那么提高到6级就可以显著减少带宽使用率，而不会对代理设备造成过多的负担。另外，最好避免压缩已经被压缩的数据，如图像和PDF。压缩它们实际上会使资源变大，而且会浪费设备的资源。SVG格式将是一个例外。另外，不要压缩小文件。配置文件的最小内容长度默认为1M。

Caching 缓存

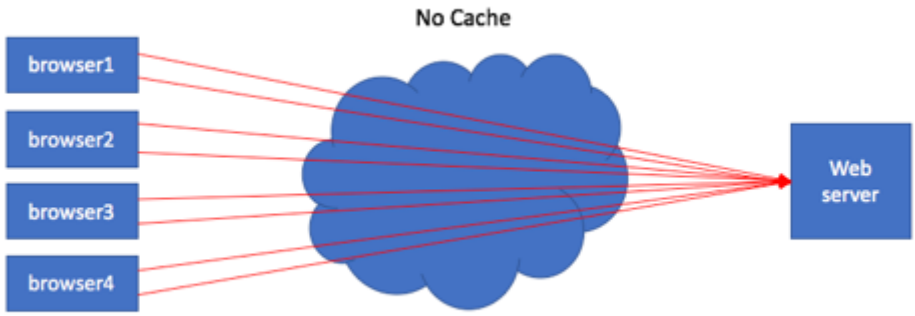
HTTP 缓存存储与请求关联的响应，并将存储的响应复用于后续请求。

可复用性有几个优点。首先，由于不需要将请求传递到源服务器，因此客户端和缓存越近，响应速度就越快。最典型的例子是浏览器本身为浏览器请求存储缓存。

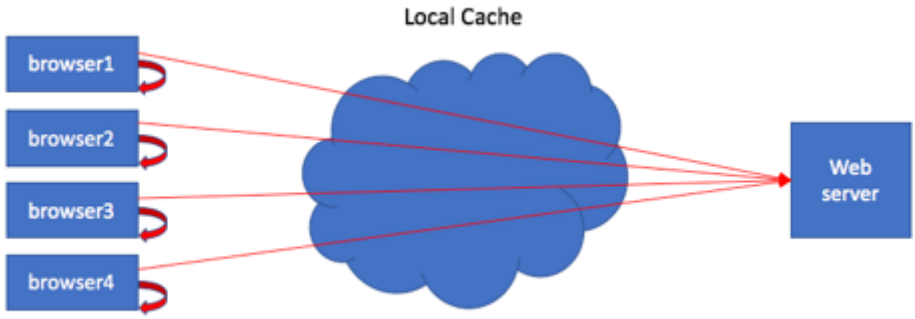
此外，当响应可复用时，源服务器不需要处理请求——因为它不需要解析和路由请求、根据 cookie 恢复会话、查询数据库以获取结果或渲染模板引擎。这减少了服务器上的负载。

缓存的正确操作对系统的稳定运行至关重要。

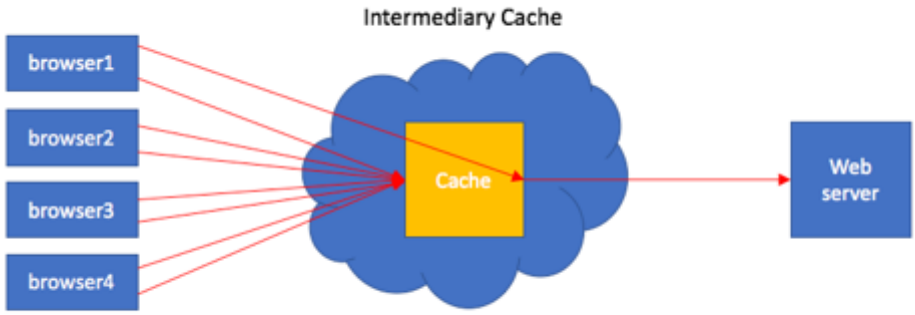
没有缓存的情况:



在这种情况下，来自浏览器的每个请求都与 Web 服务器通信，无论内容更改的频率如何。这是对服务器、网络甚至客户端本身资源的浪费。对于注意力很短暂的用户来说，最重要的资源是时间！与服务器的资源和距离越远，最终用户等待该页面展示的时间就越长。一种改变这种情况的方法是允许在浏览器中进行本地缓存：



这样，第一个请求到达 Web 服务器，并且对同一资源的重复请求将从缓存中提取（假设该资源仍然有效，下面将详细介绍。）最后，还有中间缓存。这可以直接存储到离客户端近的地方，例如企业 LAN、内容分发网络(CDN)、数据中心的服务器或以上所有地方！

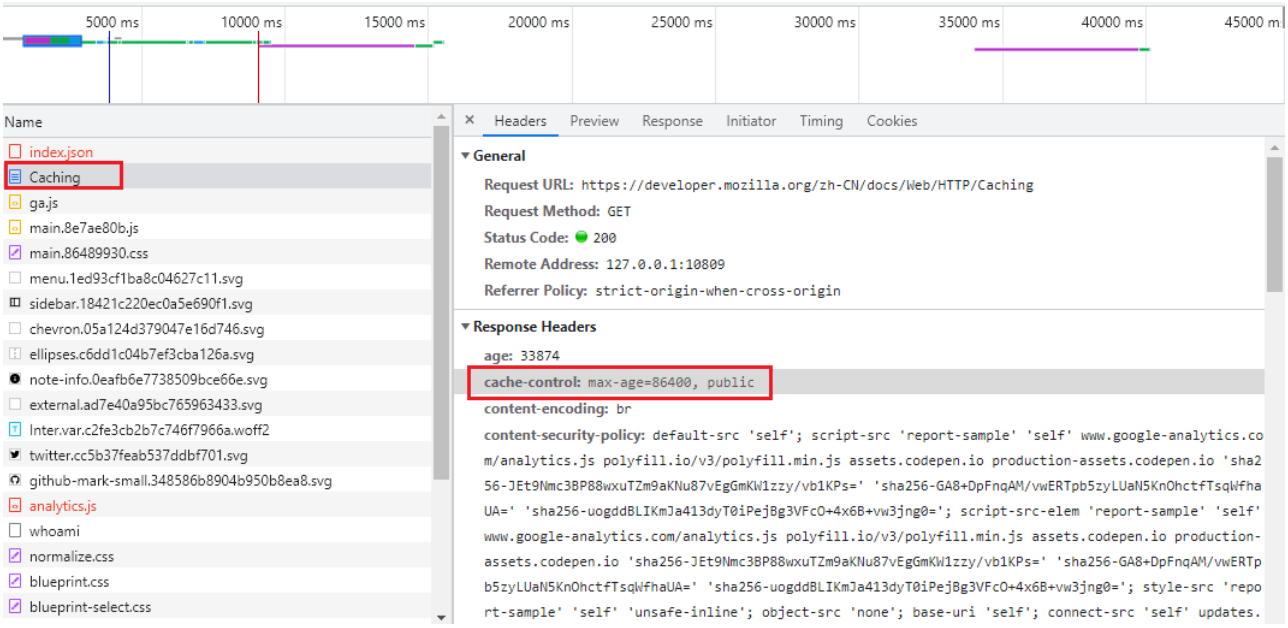


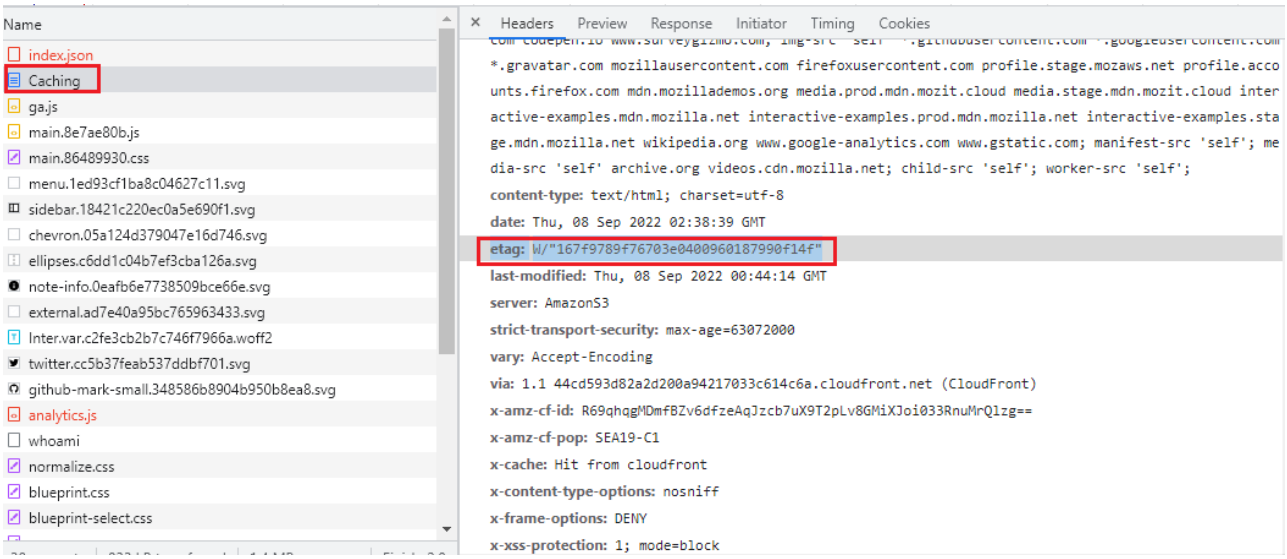
在这种情况下，browser1 客户端请求一个尚未缓存的对象。一旦缓存从服务器获取了资源对象，它就会将其提供给所有浏览器客户端，从而节省了到服务器的请求，节省这样做的时间，并使响应更接近浏览器客户端。

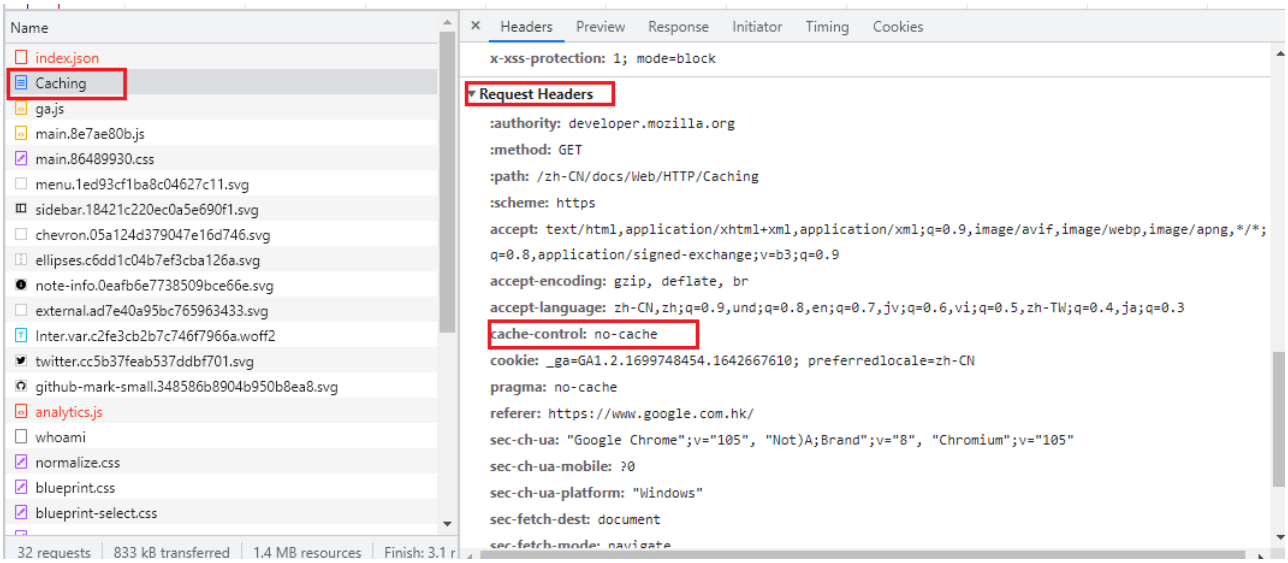
鉴于缓存解决方案的好处，让我们简要谈谈风险。如果您将控制从服务器提供的内容交给中间代理设备，尤其是原始服务器的管理员可能没有权限的代理设备，那么您如何控制浏览器最终加载的内容的真实性？这就是关于缓存控制的 HTTP 标准发挥作用的地方。

HTTP/1.0 引入了 Pragma、If-Modified-Since、Last-Modified 和 Expires 标头用于缓存控制。 HTTP/1.1 中引入了 Cache-Control 和 ETag 标头以及大量“If-”条件标头，但您会在响应中看到许多 HTTP/1.0 缓存标头以及 HTTP/1.1 标头以实现向后的兼容性。

您可以从浏览器开发人员工具和 Fiddler 和 HttpWatch 等工具中收集到很多信息。

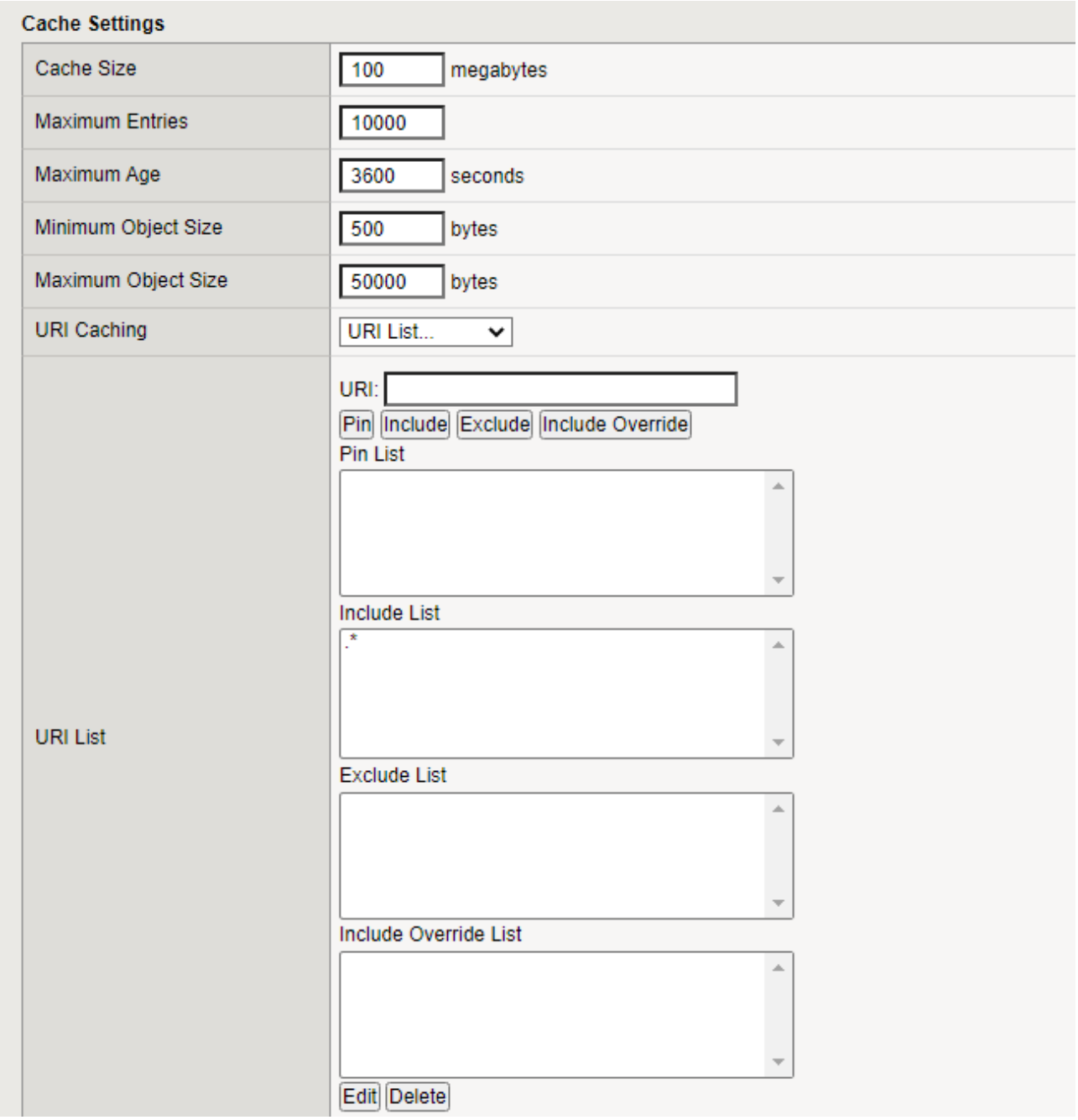


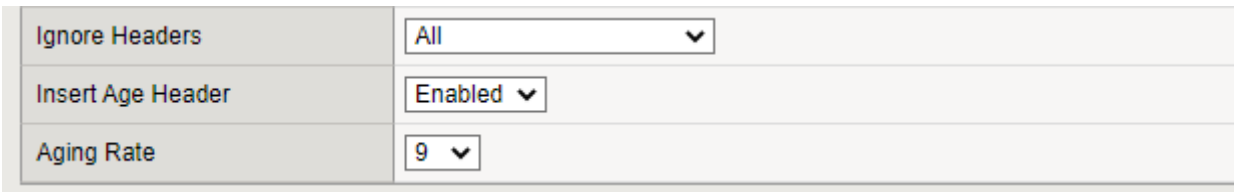




在上面最后一张图，我标记了请求的Cache-Control头，它的值是no-cache。这不是一个很直观的名字，是客户端对缓存的要求，它必须每次都向服务器提交请求，即使内容是新鲜的。这保证了认证得到尊重，同时仍然允许利用中间缓存来传递内容。在响应中，Cache-Control头有两个值：public和max-age。这里的max-age是相当大的，所以这显然是一个预计不会有太大变化的资产。public指令意味着该资源可以被存储在一个共享缓存中。

既然我们已经基本了解了什么是缓存，那么代理是如何处理缓存的呢，我们可以分析下 cache 配置文件。





可缓存内容：

- 200、203、300、301 和 410 HTTP 响应
- 对 HTTP GET 请求的响应
- 指定包含在缓存内容中或在 iRule 中指定的 URI 的其他 HTTP 方法
- 基于 User-Agent 和 Accept-Encoding 值的内容。缓存功能为 Vary 标头保存不同的内容。

不可缓存的内容：

Cache-Control 标头指定的数据：**private, no-store**

SETTING	VALUE	DESCRIPTION
Cache Size	100 MB	指定系统为每个流量管理微内核 (TMM) 缓存提供的内存量。总缓存大小等于缓存大小乘以系统上的 TMM 数量。
Maximum Entries	10000	指定系统允许缓存的最大条目数。
Maximum Age	3600 seconds	指定系统认为缓存内容有效的持续时间。
Minimum Object Size	500 bytes	指定系统存储在缓存中的最小对象大小（以字节为单位）。
Maximum Object Size	50000 bytes	指定系统存储在缓存中的最大对象大小（以字节为单位）。
URI Caching	URI List...	指定系统在缓存中保留或排除的统一资源标识符 (URI)。固定过程强制系统要么无限期地在缓存中保留 URI，缓存通常不适合缓存的 URI，要么不缓存通常适合缓存的 URI。
URI List	No default value	URI 列表指定要缓存或从缓存中排除的 URI。 - Pin List - 列出您希望系统无限期存储在 RAM 缓存中的响应的 URI。 - Include List- 配置要缓存的 URI 列表，并确定是否应根据 RFC2616 缓存规则正常评估请求。默认值为 .*，它指定所有 URI 都是可缓存的。 - Exclude List- 列出通常符合缓存条件的 URI，但系统不会缓存它们。 - Include Override List- 配置应缓存的 URI 列表，即使由于最大对象大小设置或其他设置定义的约束，它们通常不会被缓存。默认值为无。包含覆盖列表中的 URI 是可缓存的，即使它们不在包含列表中。
Ignore Headers	All	指定系统如何处理 Cache-Control 标头。 All: 指定系统不理会请求头中的所有Cache-Control头。 -Cache-Control:max-age。指定系统不理会Cache-Control:max-age请求头中max-age=0的值。 None: 指定系统处理请求头中的所有Cache-Control头。
Insert Age Header	Enabled	指定系统是否在缓存的条目中插入Date和Age头。 Date头包含系统中的当前日期和时间，而Age头包含内容在缓存中的时间长度。
Aging Rate	9	老化率 9 指定系统对缓存条目的老化速度。老化率范围从0（最慢的老化）到10（最快的老化）。

建议： 缓存对于经常请求的内容很有用；例如，如果站点对特定内容的需求量很大，则可以使用缓存。当您为虚拟服务器配置 Web 加速配置文件时，内容服务器只需在每个缓存到期后向代理系统提供一次内容。

如果站点包含大量静态内容（例如 CSS 文件、JavaScript 文件或图像），则缓存非常有用。

对于可压缩数据，缓存功能可以为接受压缩数据的客户端存储数据。当与 代理系统上的压缩功能一起使用时，缓存可以减轻 代理系统和内容服务器的压力。

排除很有用，因为某些 URI 或文件类型可能已经被压缩。建议您不要使用 CPU 资源来压缩已压缩的数据，因为压缩数据的成本通常超过收益。您可能要指定排除的正则表达式示例有 *..pdf*、..gif 或 .*.html。

参考:

HTTP 缓存配置 - zzzzy09

MDN HTTP 缓存类型 - zzzzy09

HTTP/2

HTTP/2 是 HTTP 网络协议的一个重要版本。 HTTP / 2 的主要目标是通过启用完整的请求和响应多路复用来减少 延迟，通过有效压缩 HTTP 标头字段来最小化协议开销，并增加对请求优先级和服务器推送的支持。

HTTP/2 不会修改 HTTP 协议的语义。HTTP 1.1 中的所有核心概念（例如 HTTP 方法，状态码，URI 和 headers）都得以保留。而是修改了 HTTP/2 数据在客户端和服务器之间的格式（帧）和传输方式，这两者都管理整个过程，并在新的框架层内隐藏了应用程序的复杂性。所以，所有现有的应用程序都可以不经修改地交付。

HTTP/2（原名HTTP 2.0）即超文本传输协议第二版，使用于万维网。HTTP/2主要基于SPDY协议，通过对HTTP头字段进行数据压缩、对数据传输采用多路复用和增加服务端推送等举措，来减少网络延迟，提高客户端的页面加载速度。HTTP/2没有改动HTTP的应用语义，仍然使用HTTP的请求方法、状态码和头字段等规则，它主要修改了HTTP的报文传输格式，通过引入二进制分帧实现性能的提升。

HTTP/2解决了什么问题?

HTTP是应用最广泛、采用最多的一个互联网应用协议。早期版本的HTTP协议实现简单：HTTP/0.9只用一行协议就启动了万维网；HTTP/1.0则是对流行的HTTP/0.9扩展的一个正式说明；直到HTTP/1.1，IETF才发布可第一份官方标准。早期为了实现简单是以牺牲应用性能为代价：HTTP/1.1客户端需要使用多个连接才能实现并发和缩短延迟；HTTP/1.1不会压缩请求头字段和响应头字段，从而产生不必要的网络流量；HTTP/1.1不支持有效的资源优先级，致使底层TCP连接的利用率低下等等。

随着网络应用普及到人们的日常生活，它的应用范围、复杂性、重要性也在不断扩大。为了解决HTTP协议问题，HTTP/2应运而生。HTTP/2没有改动HTTP的应用语义，仍然使用HTTP的请求方法、状态码和头字段等规则，它主要修改了HTTP的报文传输格式，通过引入二进制分帧层实现性能的提升。HTTP/2主要基于SPDY协议，通过对HTTP头字段进行首部压缩、对数据传输采用多路复用和增加服务器推送等举措，来减少网络延迟，提高客户端的页面加载速度。

HTTP/2 vs HTTP/1.1

高健壮性

HTTP/1.1，使用基于文本格式，文本表现形式多样、场景多，健壮性不足。HTTP/2使用二进制格式，只有0和1的组合，选择二进制传输，协议解析实现方便且健壮。

高性能

HTTP连接会随着时间进行自我调节，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度。这种调节被称为TCP慢启动。这种调节让具有突发性和短时性的HTTP连接变的十分低效。HTTP/2通过多路复用让所有数据流使用同一个连接，有效使用TCP连接，让高带宽也能真正的服务于HTTP的性能提升。

HTTP/2在应用层和传输层之间增加了二进制分帧，突破了HTTP/1.1性能限制，改进传输性能，实现低延迟和高吞吐量。

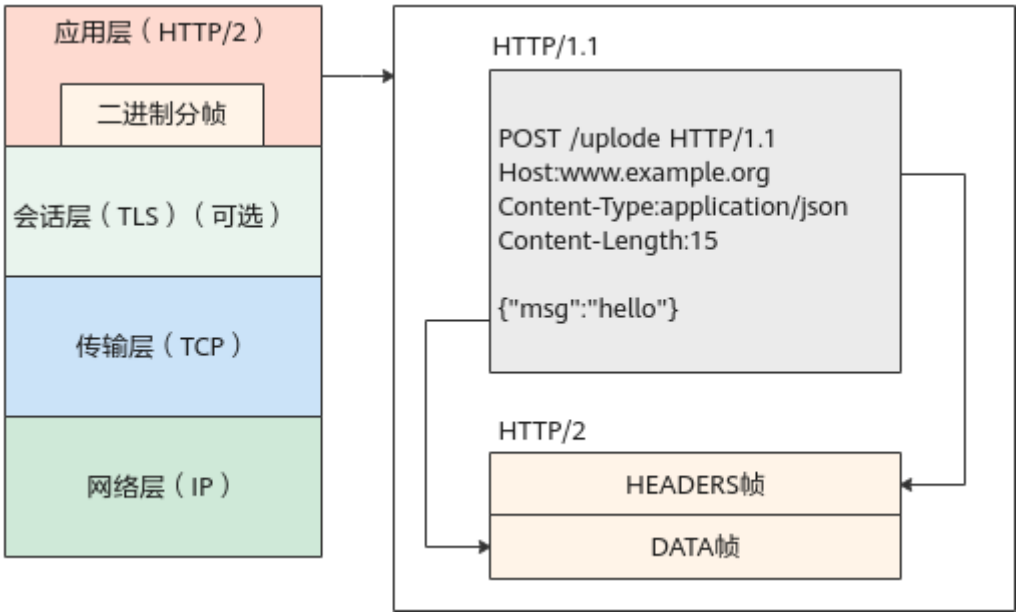
网络开销低

HTTP/2使用HPACK算法来压缩每次请求连接的头字段，降低了网络开销。HPACK算法可以减少需要传输的头字段大小，通讯双方通过建立和维护头字段表，字段表中使用长度较小的索引号表示重复的字符串，在用Huffman编码压缩数据，既避免了重复头字段的传输，又减小了需要传输的大小。

HTTP/2有哪些关键特性?

二进制分帧

HTTP/2所有性能增强的核心在于新的二进制分帧层，如下图所示，它是所有其他功能和性能优化的基础，它定义了如何封装HTTP消息并在客户端与服务器之间传输。



二进制分帧

HTTP/2没有改动HTTP的应用语义，仍然使用HTTP的请求方法、状态码和头字段等规则，它主要修改了HTTP的报文传输格式。HTTP/1.1协议以换行符作为纯文本的分隔符，而HTTP/2将所有传输的信息分割为更小的消息和帧，并采用二进制格式对它们编码，这些帧对应着特定数据流中的消息，他们都在一个TCP连接内复用。

优先级排序

将HTTP消息分解为很多独立的帧之后就可以复用多个数据流中的帧，客户端和服务器交错发送和传输这些帧的顺序就成为关键的性能决定因素。HTTP/2允许每个数据流都有一个关联的权重和依赖关系，数据流依赖关系和权重的组合明确表达了资源优先级，这是一种用于提升浏览性能的关键功能。HTTP/2协议还允许客户端随时更新这些优先级，我们可以根据用户互动和其他信号更改依赖关系和重新分配权重，这进一步优化了浏览器性能。

首部压缩

HTTP每次请求或响应都会携带首部信息用于描述资源属性。HTTP/1.1使用文本的形式传输消息头，消息头中携带cookie每次都需要重复传输几百到几千的字节，这十分占用资源。

HTTP/2使用了HPACK算法来压缩头字段，这种压缩格式对传输的头字段进行编码，减少了头字段的大小。同时，在两端维护了索引表，用于记录出现过的头字段，后面在传输过程中就可以传输已经记录过的头字段的索引号，对端收到数据后就可以通过索引号找到对应的值。

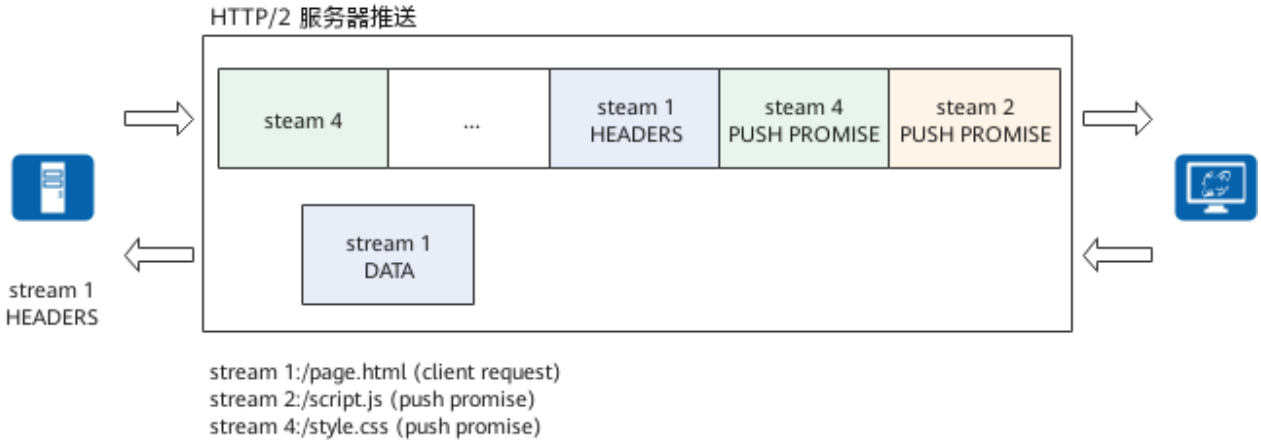
多路复用

多路复用允许同时通过单一的HTTP/2连接发起多重的请求-响应消息，实现多流并行而并不依赖多个TCP连接，HTTP/2把HTTP协议通信的基本单位缩小为一个一个的帧，这些帧对应着逻辑流中的消息，并行地在同一个TCP连接上双向交换消息。

HTTP/2基于二进制分帧层，HTTP/2可以在共享TCP连接的基础上同时发送请求和响应。HTTP消息被分解为独立的帧，而不破坏消息本身的语义交错发出去，在另一端根据流标识符和首部将他们重新组装起来。通过多路复用技术，可以避免HTTP旧版本的消息头阻塞问题，极大提高传输性能。

服务器推送

HTTP2.0的一个强大的新功能，就是服务器可以对一个客户端请求发送多个响应。服务器向客户端推送资源无需客户端明确的请求。服务端根据客户端的请求，提前返回多个响应，推送额外的资源给客户端。如下图所示，客户端请求stream 1，服务端在返回stream 1的消息的同时推送了stream 2和stream 4。



HTTP/2服务器推送

服务端推送是一种在客户端请求之前发送数据的机制。在HTTP/2中，服务器可以对一个客户端的请求发送多个响应。如果一个请求是由你的主页发送的，服务器可能会响应主页内容、logo以及样式表，因为服务端知道客户端会用到这些东西。这样不但减轻了数据传送冗余步骤，也加快了页面响应的速度，提高了用户体验。

WebSockets vs. HTTP/2 vs. SSE

WEBSOCKET	HTTP/2	SSE
Full-duplex全双工	Half-duplex半双工	Full-duplex全双工

WEBSOCKET	HTTP/2	SSE
Bidirectional双向	Interaction from a client with a specific HTTP method is required需要来自客户端与特定 HTTP 方法的交互	Unidirectional单向
Less Overhead更少的开销	Added overhead to SSL handshake增加了 SSL 握手的开销	
Service Push is a base implementation of the protocol服务推送是协议的基础实现	Only supported in HTTP/2仅在 HTTP/2 中支持	The base technology基础技术
Supported by major browsers主流浏览器支持	Supported in All browsers所有浏览器均支持	Not all browsers support it.并非所有浏览器都支持它。
1024 parallel connections1024个并行连接	6-8 parallel connections6-8个并联	6 parallel connections6个并行连接
Non-Standard Load balancing非标准负载均衡	Standard Load Balancing标准负载均衡	Standard Load Balancing标准负载均衡

参考:

https://developers.google.com/web/fundamentals/performance/http2/

https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference

https://www.rfc-editor.org/rfc/rfc2616.html

当您在浏览器中键入 **URL** 并按 **Enter** 时会发生什么？

当我们通过PC 或者手机浏览网页时不知道浏览器背后执行了哪些操作，对于了解浏览器的处理过程能够加深我们对于http 协议的了解。

1. 当您在浏览器访问**https://www.cnblogs.com/zy09/**
2. 浏览器检查缓存中的 **DNS** 记录，找到 **www.cnblogs.com** 对应的 **IP** 地址。 您可以在浏览器使用**chrome://net-internals/#dns** 清除**dns cache**

> DNS（域名系统）是一个数据库，用于维护网站名称 (URL) 及其链接的特定 IP 地址。互联网上的每个 URL 都分配有一个唯一的 IP 地址。 IP 地址属于托管我们请求访问的网站服务器的计算机。例如，www.cnblogs.com 的 IP 地址为 209.85.227.104。因此，如果您愿意，可以通过在浏览器上输入https://121.40.43.188来访问 www.cnblogs.com。 DNS 是 URL 及其 IP 地址的列表，就像电话簿是名称及其对应电话号码的列表一样。
>
> DNS 的主要目的是人性化导航。您可以通过在浏览器上输入正确的 IP 地址轻松访问网站，但想象一下必须为我们经常访问的所有网站记住不同的数字集？因此，使用 URL 更容易记住网站的名称，并通过将其映射到正确的 IP 让 DNS 为我们完成工作。
>
> 要查找 DNS 记录，浏览器会检查四个缓存。
>
> ● 首先，检查浏览器缓存。浏览器会为您之前访问过的网站维护一个固定期限的 DNS 记录存储库。因此，它是运行 DNS 查询的第一个位置。
>
> ● 其次，浏览器检查操作系统缓存。如果它不在浏览器缓存中，浏览器将对您的底层计算机操作系统进行系统调用（即 Windows 上的 gethostname）以获取记录，因为操作系统还维护 DNS 记录的缓存。
>
> ● 第三，检查local DNS缓存。如果它不在您的计算机上，浏览器将通过客户端去 local DNS 查询缓存
>
> ● 第四，检查ISP DNS 缓存。一般用户的DNS 请求最终都是由ISP DNS 进行处理。

这种多级缓存存在一定的安全和隐私风险，但是缓存对于调节网络流量和改善数据传输

的效率非常重要。

\3. 如果请求的 **URL** 不在缓存中，则 **ISP** 的 **DNS** 服务器会发起 **DNS** 查询以查找托管 **www.cnblogs.com** 的服务器的 **IP** 地址。

DNS 查询的目的是搜索 Internet 上的多个 DNS 服务器，直到找到正确的网站 IP 地址。

> 这种类型的搜索称为递归搜索，因为搜索将重复地从 DNS 服务器继续到 DNS 服务器，

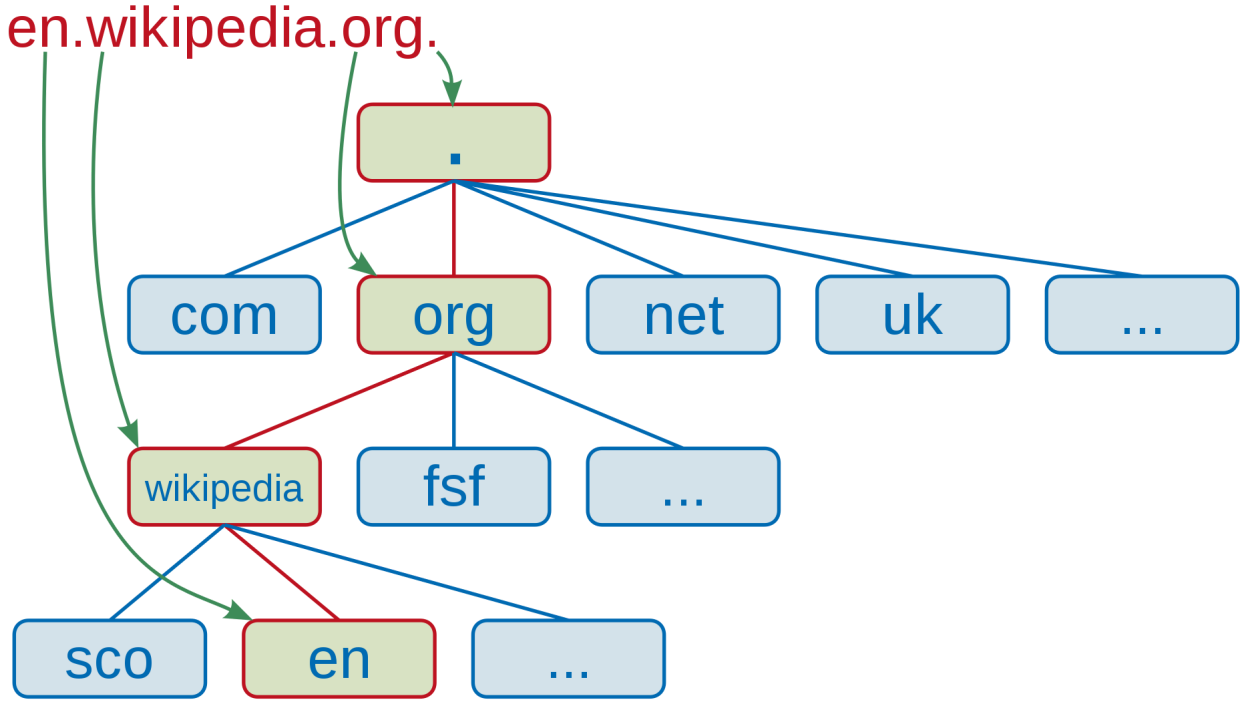
直到找到我们需要的 IP 地址或返回错误响应说它无法找到它。

在这种情况下，我们将 ISP 的 DNS 服务器称为 DNS 递归器，其职责是通过询问

Internet 上的其他 DNS 服务器的答案来找到预期域名的正确 IP 地址。其他 DNS 服务器

称为名称服务器，因为它们根据网站域名的域架构执行 DNS 搜索

例如：



我们今天遇到的很多网站网址都包含三级域、二级域和顶级域。这些级别中的每一个都包含自己的名称服务器，在 DNS 查找过程中会对其进行查询。

这些请求使用小数据包发送，这些数据包包含请求内容和目标 IP 地址（DNS 递归器的 IP 地址）等信息。这些数据包在到达正确的 DNS 服务器之前，会通过客户端和服务器之间的多个网络设备。该设备使用路由表来确定数据包到达其目的地的最快方式。如果这些数据包丢失，您将收到请求失败错误。否则，他们将到达正确的 DNS 服务器，获取正确的 IP 地址，然后返回浏览器。

1. 浏览器发起与服务器的 **TCP** 连接。

一旦浏览器接收到正确的IP地址，它就会与匹配IP地址的服务器建立连接以传输信息。浏览器使用互联网协议来建立这样的连接。可以使用多种不同的 Internet 协议，但 TCP 是用于多种 HTTP 请求的最常用协议。

要在您的计算机（客户端）和服务器之间传输数据包，建立 TCP 连接很重要。此连接是使用称为 TCP/IP 三向握手的过程建立的。这是一个三步过程，客户端和服务器交换 SYN（同步）和 ACK（确认）消息以建立连接。

1. 客户端机器通过 Internet 向服务器发送一个 SYN 数据包，询问它是否为新连接打开。我会在其它文章中详细描述这一过程。
2. 如果服务器有可以接受和发起新连接的开放端口，它将使用 SYN/ACK 数据包以 SYN 数据包的确认响应。
3. 客户端将收到来自服务器的 SYN/ACK 数据包，并通过发送 ACK 数据包来确认它。

然后建立TCP连接进行数据传输！

4. 浏览器向网络服务器发送 **HTTP** 请求。

一旦建立 TCP 连接，就该开始传输数据了！浏览器将发送一个请求 www.cnblogs.com 网页的 GET 请求。如果您正在输入凭据或提交表单，这可能是一个 POST 请求。此请求还将包含其他信息，例如浏览器标识（User-Agent 标头）、它将接受的请求类型（Accept 标头）以及要求它保持 TCP 连接活动以应对其他请求的连接标头。它还将传递从浏览器为此域存储的 cookie 中获取的信息。

GET 请求示例:

```
GET /images/layout/logo.png HTTP/1.1
Host: packetlife.net
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.3)
Gecko/20100423 Ubuntu/10.04 (lucid) Firefox/3.6.3
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie:
__gads=ID=be651be986ac2a7:T=126987862:S=ALNI_MZdKlZ_XhLKVjt8GzrMTlw
EKOuAQ;
__utma=116878343.438472518.1269487857.1275669543.1275673440.259;
__utmz=116878343.1275503662.251.26.utmcsr=google|utmccn=
(organic)|utmcmd=organic|utmctr=path%20mtu%20discovery%20packetlife
; sessionid=c9f137b8fff0639404c94d55032ca85e
```

（如果您对幕后发生的事情感到好奇，可以使用F12 httpwatch ， filddle 等工具查看 HTTP 请求。)

1. 服务器处理请求并返回响应。

服务器包含一个web服务器（即 Apache、IIS），它接收来自浏览器的请求并将其传递给请求处理程序以读取并生成响应。请求处理程序是一个程序（用 ASP.NET、PHP、Ruby 等编写），它读取请求、其标头和 cookie，以检查请求的内容，并在需要时更新服务器上的信息。然后它将以特定格式（JSON、XML、HTML）封装响应。

1. 服务器发出 **HTTP** 响应。

服务器响应包含您请求的网页以及状态码、压缩类型（Content-Encoding）、如何缓存页面（Cache-Control）、要设置的任何 cookie、隐私信息等。

HTTP 服务器响应示例：

```
HTTP/1.1 200 OK
Server: nginx/0.8.34
Date: Fri, 04 Jun 2010 18:43:08 GMT
Content-Type: image/png
Content-Length: 22612
Last-Modified: Wed, 30 Sep 2009 02:12:49 GMT
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Sat, 04 Jun 2011 18:43:08 GMT
Cache-Control: max-age=31536000
Cache-Control: public
Accept-Ranges: bytes
```

如果您查看上面的响应，第一行会显示一个状态代码。这非常重要，因为它告诉我们响应的状态。有五种使用数字代码详细说明的状态。

- 1xx 仅表示信息性消息
- 2xx 表示某种成功
- 3xx 将客户端重定向到另一个 URL
- 4xx 表示客户端出错
- 5xx 表示服务器端出错

因此，如果您遇到错误，您可以查看 HTTP 响应以检查您收到的状态代码类型。

1. 浏览器显示 **HTML**内容（对于 **HTML** 响应，这是最常见的）。

浏览器分阶段显示 HTML 内容。首先，它将呈现 HTML 结构。然后它会检查 HTML 标签并发送 GET 请求以获取网页上的其他元素，例如图像、CSS 样式表、JavaScript 文件等。这些静态文件由浏览器缓存，因此不必获取它们下次访问该页面时再次访问。最后，您会在浏览器上看到 www.cnblogs.com。

这只是一个大致的过程，后面我们会逐步了解浏览器的详细处理和后端web 服务器的处理。

Web 浏览器如何工作?

Web 浏览器是一个集成度很高的软件，浏览器具有定位、检索和显示所需信息的功能，Web 浏览器由各种结构化的功能模块组成，这些功能组件将网页的 HTML、CSS 和 javascript 展示给最终用户。根据他们的功能，这些代码包被定义为浏览器的各种组件，如下所示

- user interface 用户界面
- Browser Engine 浏览器引擎
- Rendering Engine 渲染引擎
- Networking 网络
- Javascript Interpreter Javascript 解释器
- UI Backend 用户界面后端
- Data Persistence 数据持久性存储

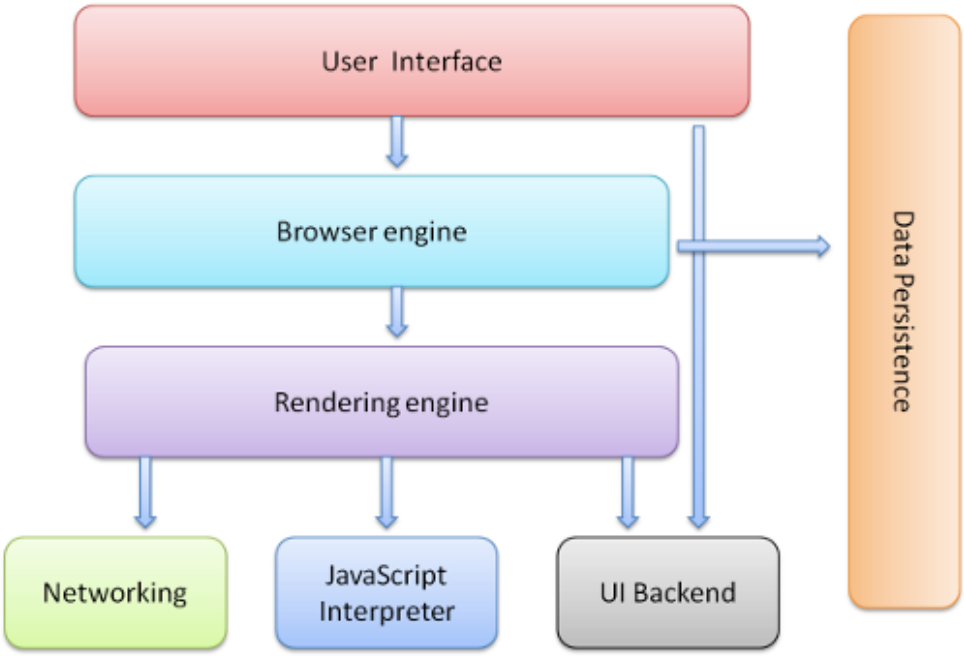


Figure : Browser components

浏览器关键组件介绍

- user interface: 用户界面是用户和浏览器这个软件交互的界面，用户界面由各种元素和按钮组成，例如地址栏、下一步和后退按钮、刷新、停止和书签选项等协助用户与浏览器的交互更具交互性。
- Browser Engine: 它是每个 Web 浏览器的核心组件。浏览器引擎充当用户界面和渲染引擎之间的中介或桥梁。它根据从用户界面接收到的输入来查询和处理渲染引擎。
- Rendering Engine: 顾名思义，这个组件负责在他们的屏幕上渲染用户请求的特定网页。它解释 HTML 和 XML 文档以及使用 CSS 设置样式或格式化的图像，并生

成最终布局，显示在用户界面上。

- Networking: 浏览器的网络组件负责检索 URL。以及该组件负责使用标准协议（如 HTTP 或 FTP）管理网络调用。它还负责处理与互联网通信相关的安全问题。
- Javascript Interpreter:Javascript 解释器仅负责解释和执行 javascript 代码。在解释 javascript 代码后收到的最终结果然后交给渲染引擎。
- UI Backend:该组件使用底层操作系统的用户界面方法。它主要用于绘制基本小部件（窗口和组合框）。
- Data Persistence:它是一个数据存储功能组件。 Web 浏览器需要在本地存储各种类型的数据，例如 cookie。因此，浏览器必须兼容 WebSQL、IndexedDB、FileSystem 等数据存储机制。

浏览器引擎

负责呈现网页的浏览器引擎的主要 2 个组件是：**Browser Engine** 和 **Rendering Engine**

渲染引擎和浏览器引擎的主要区别在于它们的工作区域。当用户请求浏览网页时，浏览器引擎将请求发送给渲染引擎，渲染引擎进一步调用浏览器的网络组件，然后网络组件向渲染引擎提供所需的HTML、CSS和js文件。渲染引擎然后将这些文件转换为浏览器引擎可以理解的特定格式，并转发解释后的代码。然后，浏览器引擎接受解释后的代码，并进一步显示给终端用户。

渲染引擎将接收到的代码解释为浏览器可以理解的代码，浏览器引擎充当用户和渲染引擎之间的接口，在从渲染进程接收到解释后的代码后，帮助将结果传递给用户。

Rendering Engine 工作过程

一旦用户请求特定文档，渲染引擎就会开始获取所请求文档的内容。这是通过网络层完成的。渲染引擎开始以 8 KB 的块从网络层接收该特定文档的内容。之后，渲染引擎的基本流程就开始了。

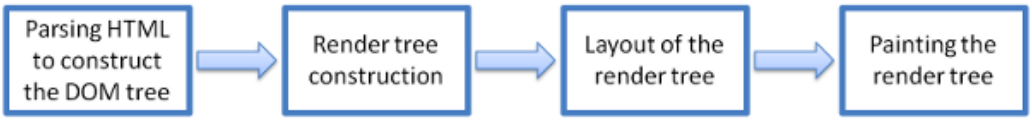
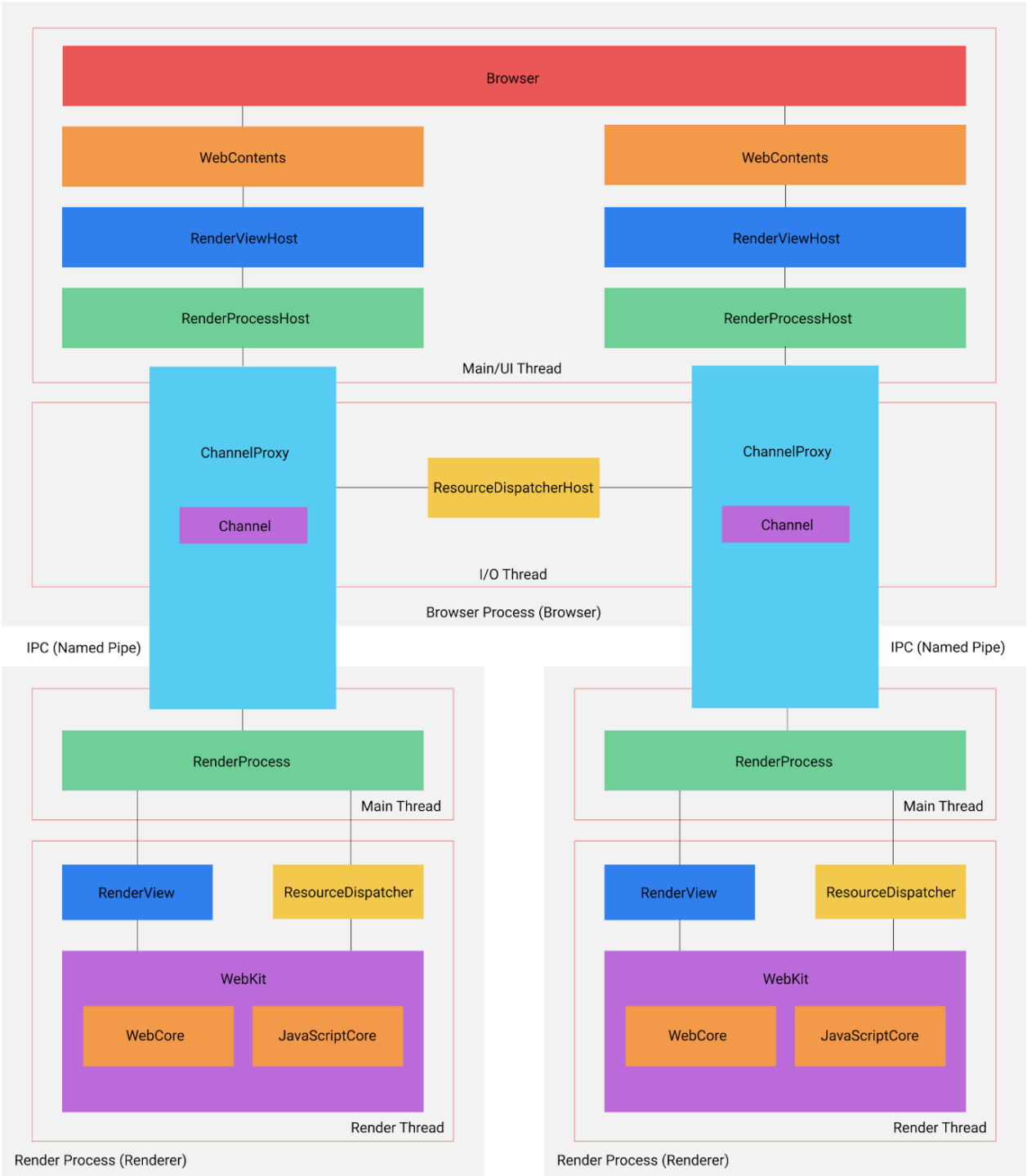


Figure : Rendering engine basic flow

四个基本步骤包括：

1. 请求的 HTML 页面由渲染引擎以块的形式解析，包括外部 CSS 文件和样式元素。然后将 HTML 元素转换为 DOM 节点以形成“内容树”或“DOM 树”。也就是整个页面的内容骨架，包含DOM 树和CSSOM 树。

2. 同时，浏览器也会创建一个渲染树。渲染是通过合并 DOM 和 CSSOM 树来构建的。渲染树包含可见内容的所有信息，包括它们的 CSS 样式。该树包括样式信息以及定义元素显示顺序的视觉说明。渲染树确保内容以所需的顺序显示。
3. 此外，渲染树会经历布局过程。在这个阶段，渲染树中的节点在各自的页面上被分配了它们的宽度、高度和 x - y 坐标。浏览器通过从渲染树的根元素到最后一个节点遍历渲染树来确定每个节点的大小和位置。创建渲染树时，不会分配位置或大小值。计算用于评估所需位置的值的整个过程称为布局过程。在这个过程中，每个节点都被分配了精确的坐标。这确保了每个节点都出现在屏幕上的准确位置。
4. 最后一步是绘制屏幕，其中遍历渲染树，并调用渲染器的paint() 方法，该方法使用UI 后端层绘制屏幕上的每个节点。



针对上上述内容详细信息可参考:

https://web.dev/howbrowserswork/ (最全面)

https://gs.statcounter.com/browser-market-share

https://zicodeng.medium.com/explore-the-magic-behind-google-chrome-c3563dbd2739

https://stackoverflow.com/questions/46169376/whats-the-difference-between-a-browser-engine-and-rendering-engine

Web server 如何工作**?**

Web 服务器是计算机软件和底层硬件，通过 HTTP（为分发 Web 内容而创建的网络协议）或其安全变体 HTTPS 接受请求。用户代理，通常是网络浏览器或网络爬虫，通过使用 HTTP 对网页或其他资源发出请求来启动通信，服务器以该资源的内容或错误消息进行响应。，Web 服务器还可以接受和存储从用户代理发送的资源。

从 web 服务器发送的资源可以是 web 服务器可用的预先存储在本地的文件（静态内容），也可以在请求时由与服务器软件通信的另一个程序生成（业务应用程序）。前者通常可以更快地提供服务，并且可以更容易地缓存重复请求，而后者是支持更广泛的应用程序。

随着网络流量爆发式增长和业务场景的快速变化，HTTP 协议发展出了各种变化，例如: websocket 、SPDY、HTTP/2以及最新推出的HTTP/3。

Web 服务器通用基本功能

尽管 Web 服务器程序在实现方式上有所不同，但它们中的大多数都提供以下共同功能。

- 静态内容服务：能够通过 HTTP 协议向客户端提供静态内容（Web 文件）。
- HTTP：支持一个或多个版本的 HTTP 协议，以便发送给客户端 HTTP 请求版本兼容的 HTTP 响应版本，例如HTTP/1.0、HTTP/1.1、HTTPS，还有 HTTP/2、HTTP/3。
- 日志记录：通常 Web 服务器还具有将一些信息（关于客户端请求和服务器响应）记录到日志文件中以便用于安全和统计的能力。
- 动态内容服务：能够通过 HTTP 协议向客户端提供动态内容（后端应用）
- 虚拟主机：能够仅使用一个 IP 地址服务于多个网站（域名）。
- 授权：能够允许、禁止或授权访问部分网站路径（网络资源）。
- 内容缓存：能够缓存静态和/或动态内容，以加快服务器响应速度。
- 带宽限流：限制内容响应的速度，以免网络拥塞并能够为更多客户端提供服务；
- 重写引擎：重写引擎是对 URL（统一资源定位器）执行重写、修改的软件组件。
- 自定义错误页面：支持自定义 HTTP 错误消息。

Web 服务器通用任务

Web 服务器程序在运行时通常会执行几项任务

启动:web 应用可选地读取其配置文件或以其它方式读取配置，可选地打开日志文件，开始监听客户端连接/请求；

启动条件： 有选择的启动例如:Web服务器的设置、客户端请求使用的 HTTP 版本、HTTP 请求类型、内容是否被缓存、请求的内容是静态的还是动态、内容是否被压缩、连接是否加密、Web 服务器与其客户端之间的平均网络速度、活动 TCP 连接的数量、Web 服务器管理的活动进程数（包括外部 CGI、SCGI、FCGI 程序）以及其它等等设置

接收客户端请求（读取 HTTP 消息）:

- 读取并验证每个 HTTP 请求消息；
- 通常执行URL 规范化(例如将URL 最后增加/或者转为小写)；
- 通常执行 URL 映射（路径资源映射）
- 通常执行 URL 各种安全检查；

执行或拒绝请求的 HTTP 方法：

- 可执行 URL 授权；
- 可执行URL 重定向；
- 可选择执行对静态资源（文件内容）的请求：
- 可选地管理对动态资源的请求：
- 可以选择管理程序或模块的处理，检查用于生成动态内容的外部程序的可用性、开始和最终停止执行。
- 可以选择管理与外部程序/内部模块的通信，用于生成动态内容。

响应客户端请求:

- 可选择将客户的请求和/或其响应（部分或全部）记录到外部用户日志文件或通过syslog记录到系统日志文件，通常使用普通的日志格式。
- 可选择使用syslog或其他一些系统设施记录关于检测到的异常或其他值得注意的事件的处理信息（例如，在客户请求或其内部功能中）；这些日志信息通常有一个调试、警告、错误、警报级别，可以根据一些设置进行过滤（不记录），另见严重程度级别。
- 可选地生成有关网络流量或其性能的统计数据。
- 其它任务

读取请求处理步骤

Web 服务器程序能够:

1. 读取 HTTP 请求消息;
2. 解释执行
3. 语法验证
4. 识别已知的 HTTP 标头并从中提取它们的值。
5. 一旦 HTTP 请求被解码和验证，它所携带的数据就是进行进一步的逻辑处理

web 服务器开发

实现一个简单的web 服务器,请参考下面链接。

参考:

http://aosabook.org/en/500L/a-simple-web-server.html#figure-22.1

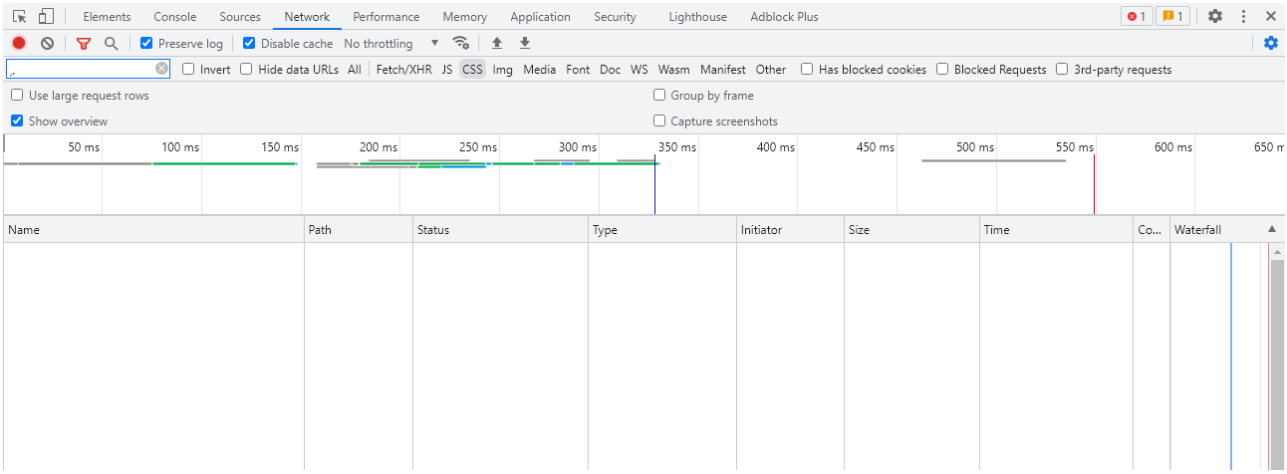
Chrome DevTools

我前面用了很大篇幅来说明关于HTTP 协议的各个部分内容，但是作为一名工程师终究会在具体的工作遇到关于HTTP 协议各种各样的问题，只懂得HTTP 协议概念是远远解决不了问题的。通常我们有两种手段来分析， 数据包分析和浏览器分析。数据包分析我会单独在另一个文档中进行描述，使用浏览器自带的开发工具能够让我们快速发现问题和解决问题，我通过业务系统运维人员提供的DevTools 截图信息解决了很多问题。这里我们以Chrome DevTools 来说明，因为chrome 的市场占有率到达了 65%. 掌握相关技巧能够帮助你准确快速的分析出HTTP 协议应用相关问题。至于前端的 HTML 、CSS、Javascript 等相关知识可以参考:

前端 - zzzzy09

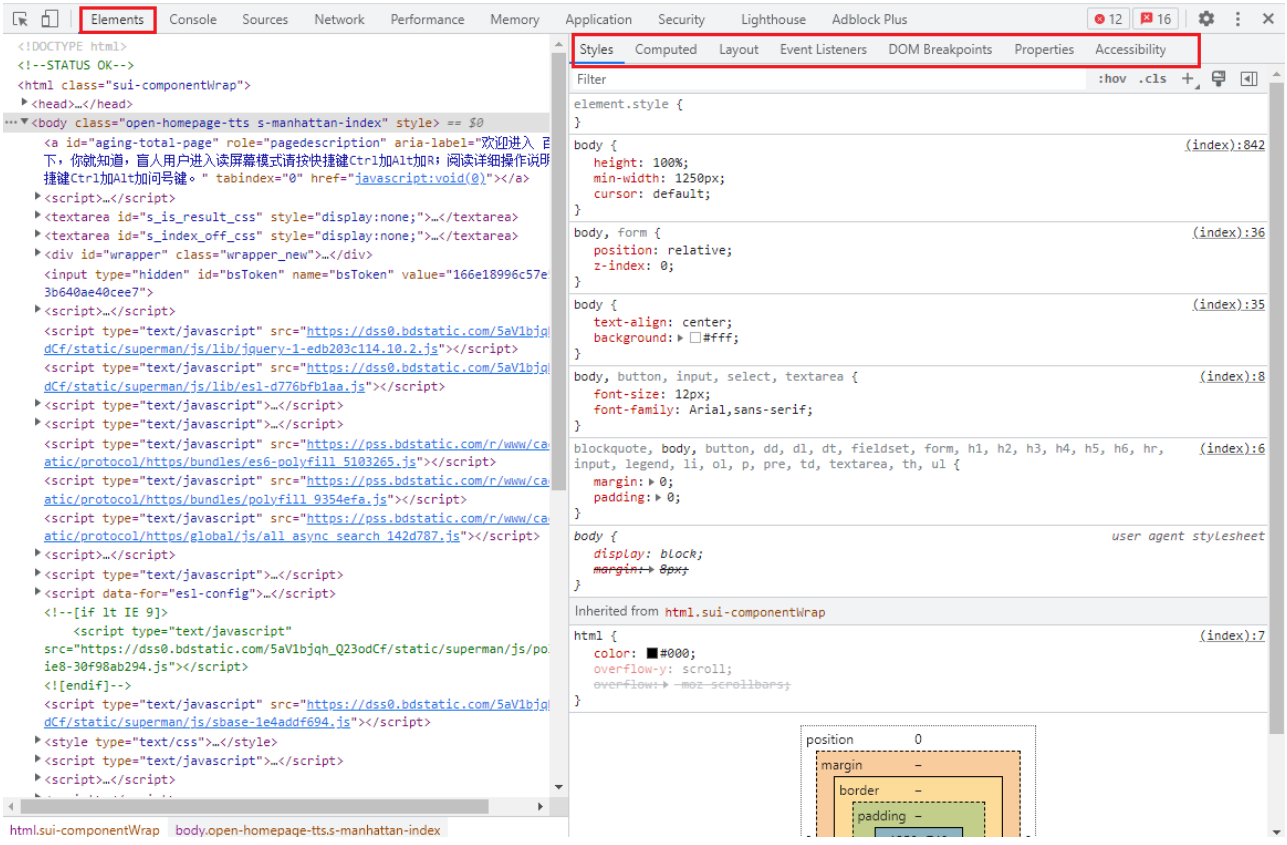
Javascript - zzzzy09

通过F12 或者浏览器按钮开发者工具打开DevTools



- Elements panel :查看和更改 DOM 和 CSS。
- Console panel : 从控制台查看消息并运行 JavaScript。
- Sources panel: 调试 JavaScript，在页面重新加载时保留在 DevTools 中所做的更改，保存和运行 JavaScript 片段，以及将您在 DevTools 中所做的更改保存到磁盘。
- Network panel : 查看和调试网络活动。
- Performance panel : 寻找提高负载和运行时性能的方法。
- Memory panel: 查看资源内容使用情况。
- Application panel : 检查所有加载的资源，包括 IndexedDB 或 Web SQL 数据库、本地和会话存储、cookie、应用程序缓存、图像、字体和样式表。
- Security panel : 调试混合内容问题、证书问题等。

Elements



Elements 面板的 DOM 树是您看到的所有直观能看到的内容。可以看到DOM 树 ， HTML 标签的css 样式，布局。关于 Elements 我们遇到常见问题:

HTML:

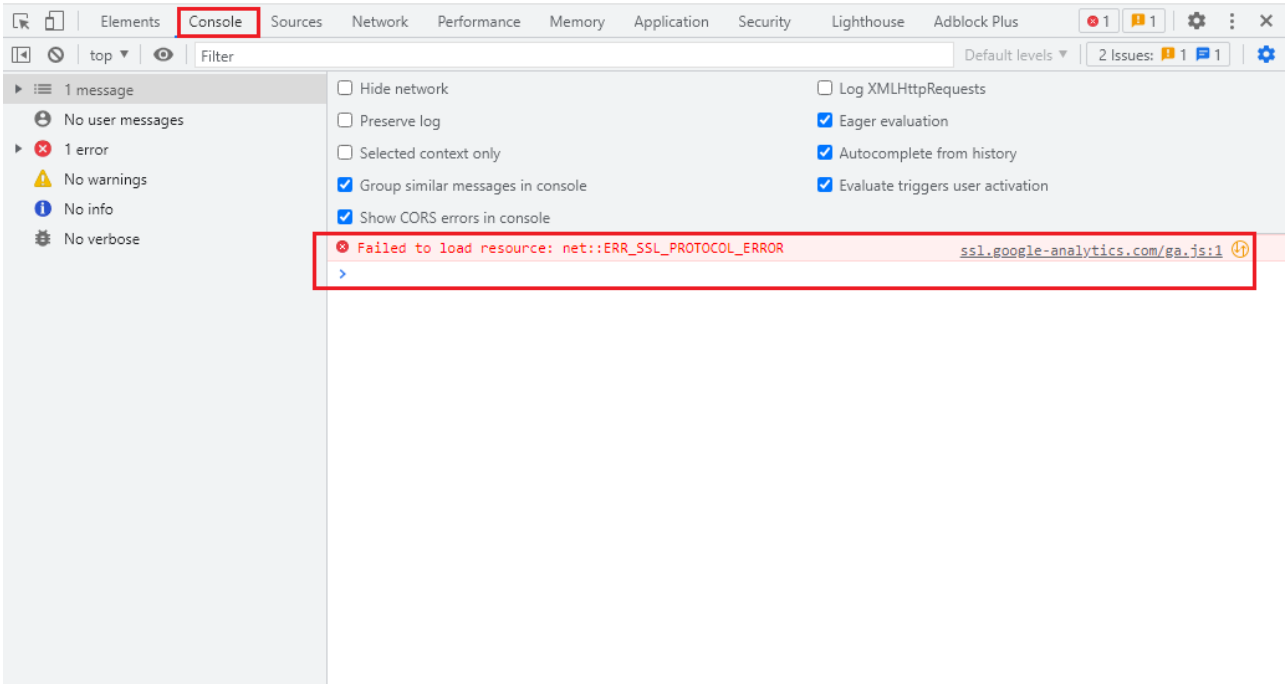
- 基本结构
- 基本问题语义
- 超链接
- 图像& 多媒体
- 脚本和样式
- 嵌入内容

CSS:

- CSS 基本使用
- 样式和内容
- 盒子和布局

以上内容是关于 Elements 常见的问题，如果有人找你咨询系统上述相关问题，那么这大概率设计前端开发。

Console



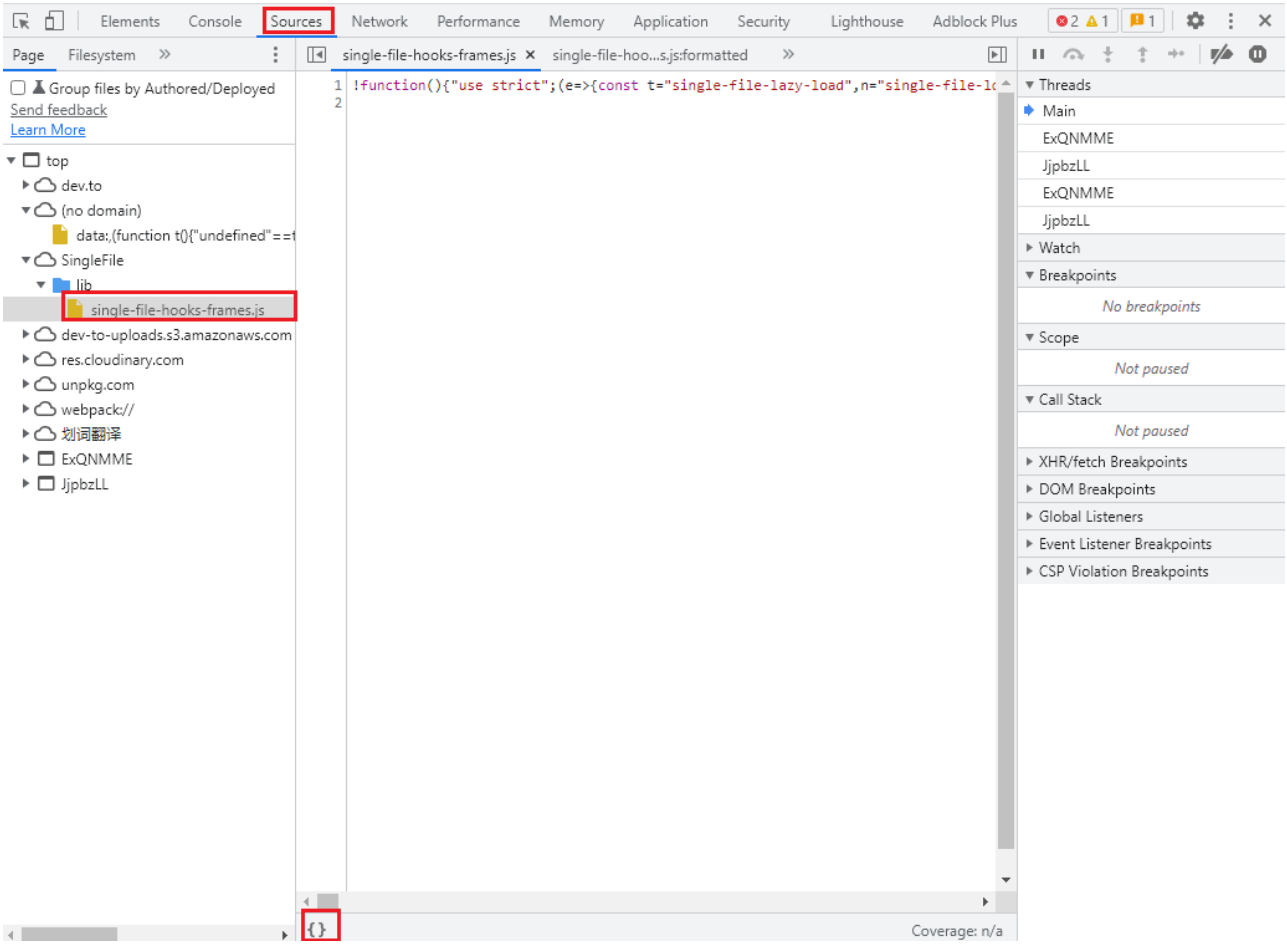
console panel 是前端开发调试javascript 的工具，javascript 的日志输出和告警都在这里输出。

常见问题:

- 确保代码以正确的顺序执行。
- 在某个时刻检查变量的值。
- 语法错误
- 函数执行错误

关于javascript 执行的报错我们 能得到很多有用的信息，例如: 当页面里的表单或者列表的渲染出现问题，一般你可以在这里找到问题的答案。

Sources

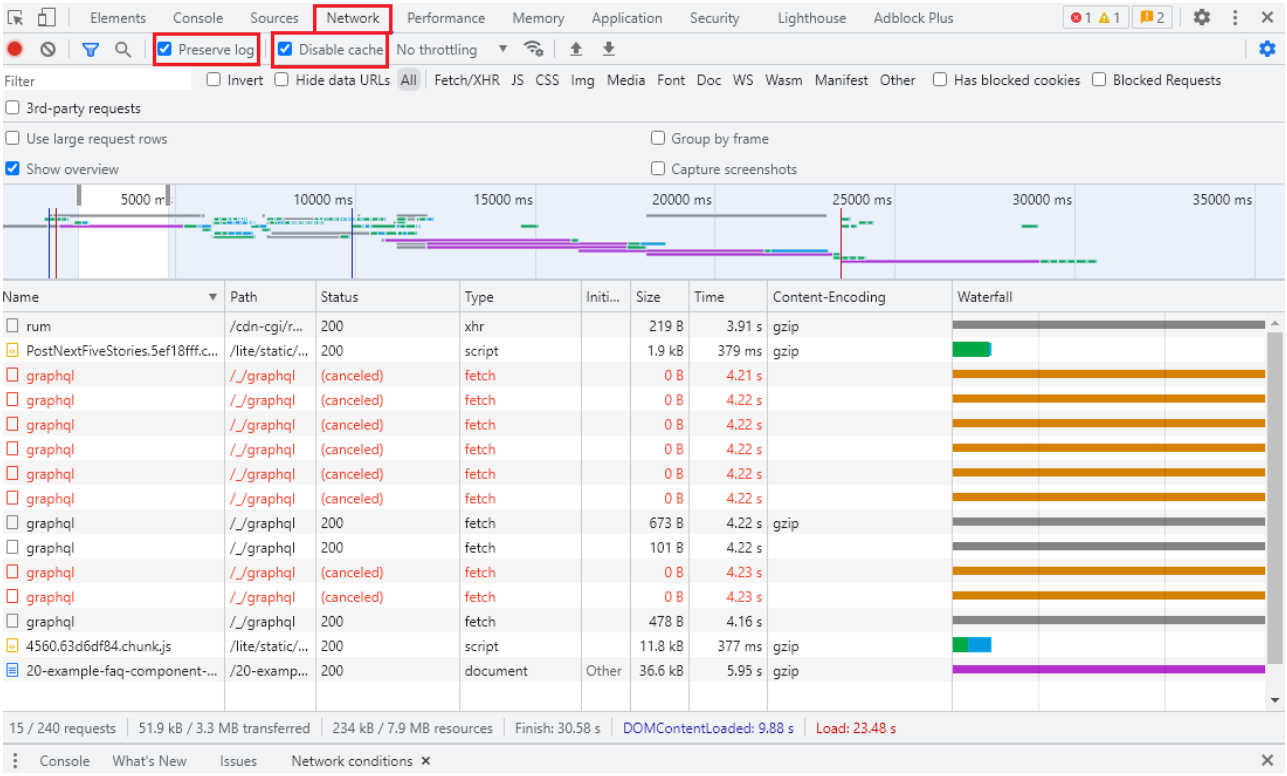


功能:

- 查看文件
- 编辑 CSS 和 JavaScript
- 创建并保存可以在任何页面上运行的 JavaScript 片段。片段类似于书签
- 调试 JavaScript
- 设置工作区，以便将您在 DevTools 中所做的更改保存到文件系统上的代码中

sources panel 我们可以直观的看到页面加载内容的结构，在问题处理时用的并不是特别多，注意底部的 {} ,当你想详细分析javascript 脚本时点击 {} ,脚本会展开显示，不是压缩格式，否则你根本无法分析。

Network

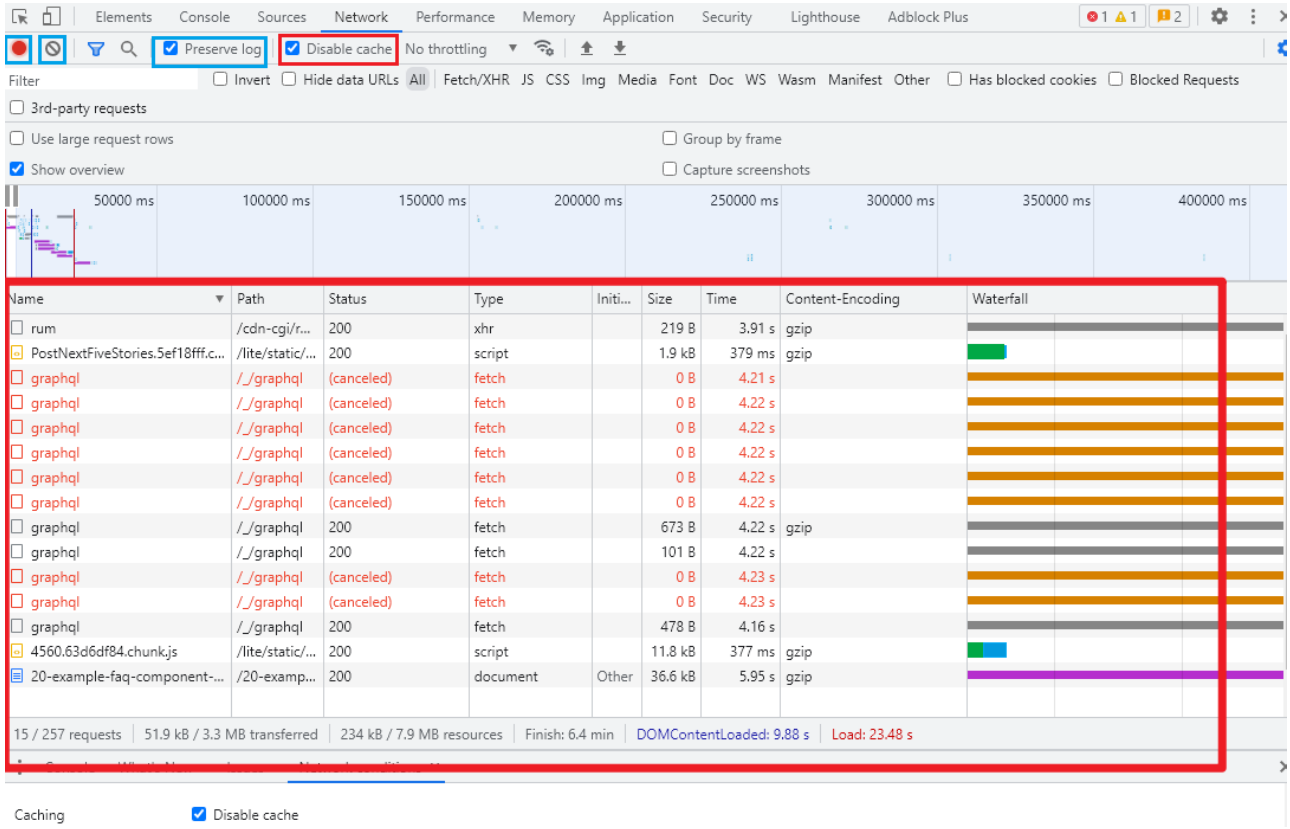


Network panel: 我们来到了最常使用的内容部分，这个部分内容非常多。network 是浏览器关于资源相关网络活动的输出。

功能:

- 检查资源上传或下载实际上的网络情况。
- 检查单个资源的属性，例如其 HTTP 标头、内容、大小等。

记录网络活动



网络日志的每一行代表一个资源。默认情况下，资源按时间顺序列出。顶部资源通常是主要的 HTML 文档。底部资源是最后请求的资源。

每列代表有关资源的信息。红框显示了默认列：

- **Status** :HTTP 响应代码
- **path** : 请求资源路径
- **Type**：资源类型
- **Initiator**: 发起人。是什么导致资源被请求。单击 Initiator 列中的链接会将您带到导致请求的源代码。
- **Time**:请求花费了多长时间。
- **Waterfall**：瀑布。请求不同阶段的图形表示。将鼠标悬停在瀑布上以查看故障。
- **Size**: 请求资源大小
- **Content-Encoding**： 响应资源编码格式

注意顶端四个标记按钮:

默认打开 Devtools 就开始记录网络请求

红色按钮: 停止记录网络请求

黑色按钮: 清除请求

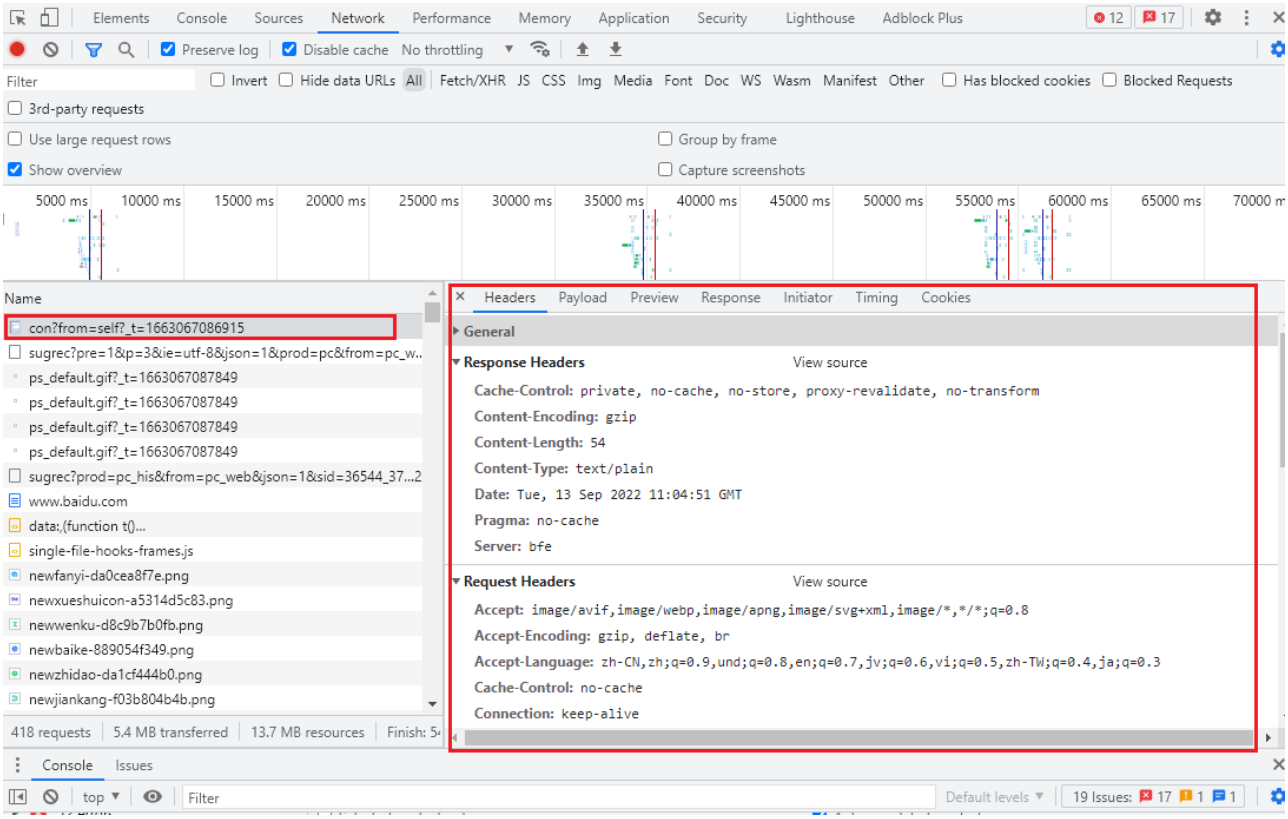
Preserve log：跨页面保存加载请求

Disable cache: 禁用缓存

实际有很多的功能，如您感兴趣可以通过官网了解

检查资源的详细信息

单击资源以了解有关它的更多信息



Headers:显示标题选项卡。使用此选项卡检查 HTTP 标头。

Preview：显示了 HTML 的基本呈现

Response:单击响应选项卡,显示了 HTML 源代码。

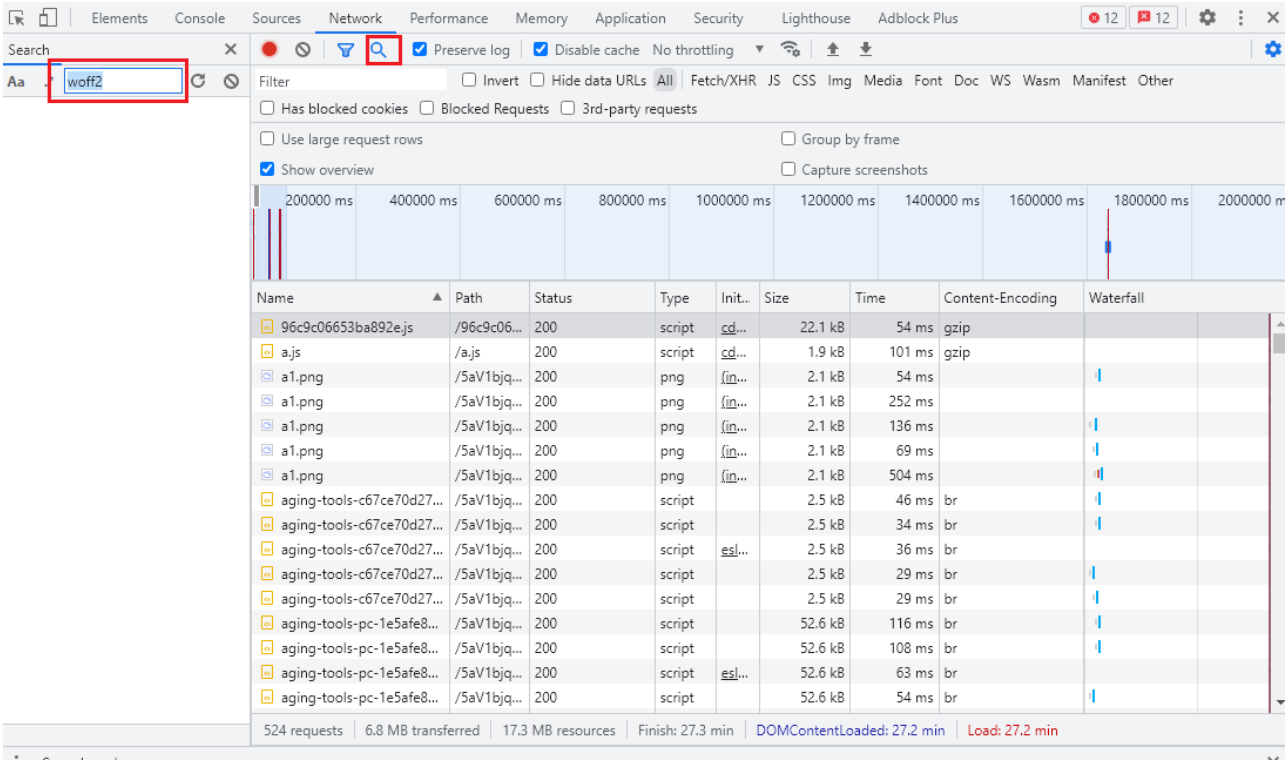
Timing：单击时序选项卡。显示此资源的网络活动详细信息。

payload：请求的数据，一般javascript 脚本执行能看到

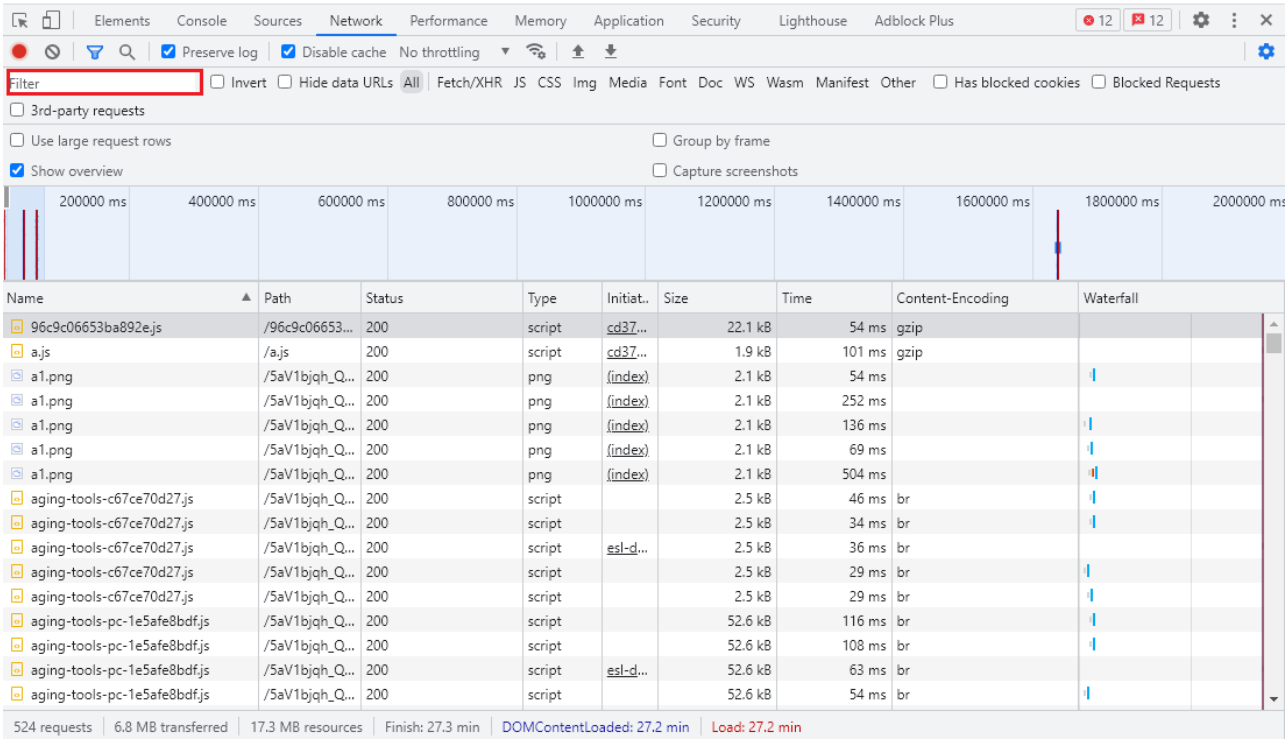
cookies： cookie 信息

搜索网络标头和响应

点击 搜索按钮会出现侧边栏搜索框，当您需要在所有资源的 HTTP 标头和响应中搜索某个字符串或正则表达式时，请使用“搜索”窗格。

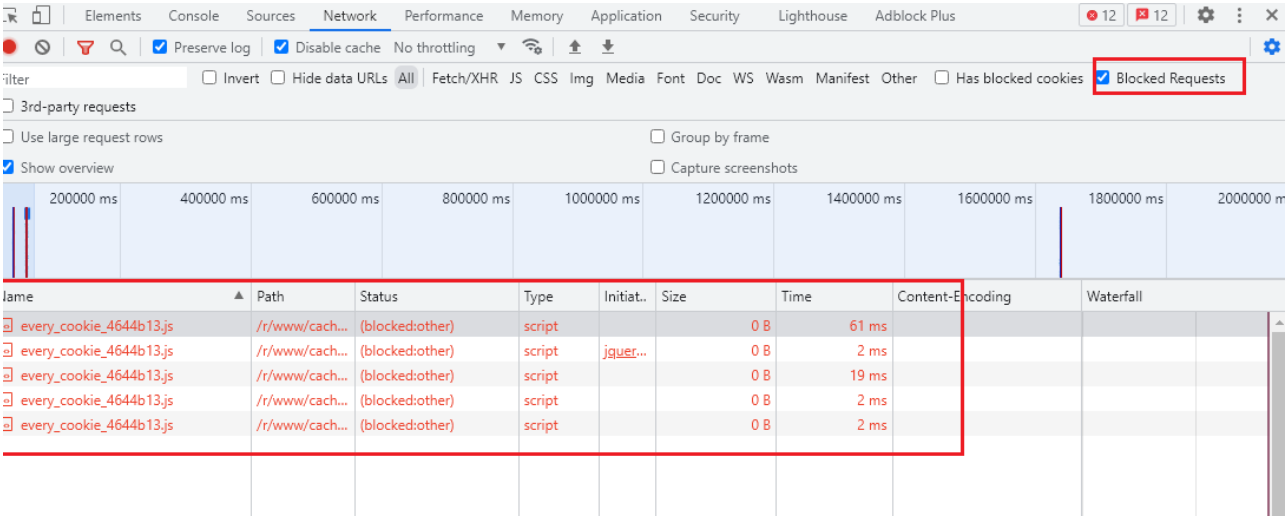


过滤资源

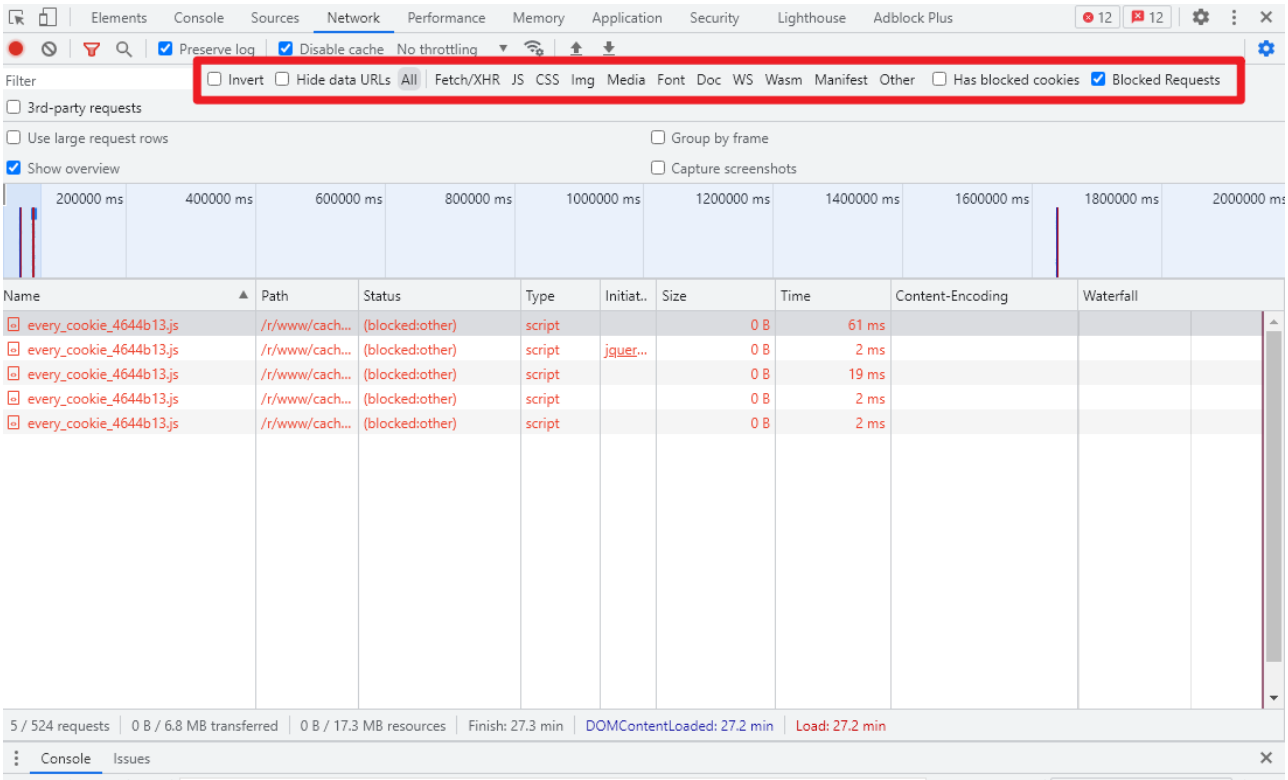


可以按照按字符串、正则表达式或属性过滤、 按资源类型过滤

显示阻止请求



高级功能:

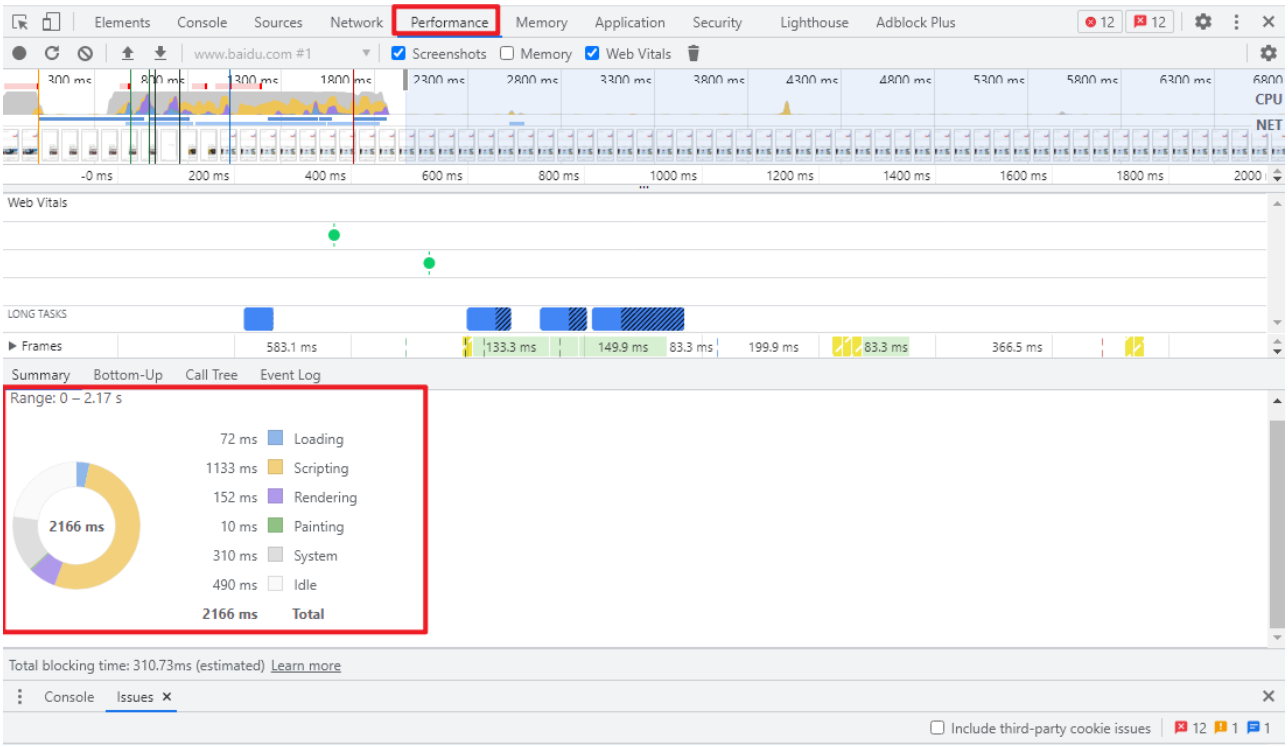


关于其它高级功能参考:https://developer.chrome.com/docs/devtools/network/reference/

在网络部分可以看到所有资源的请求和响应情况，非常便于分析数据上传下载的问题。

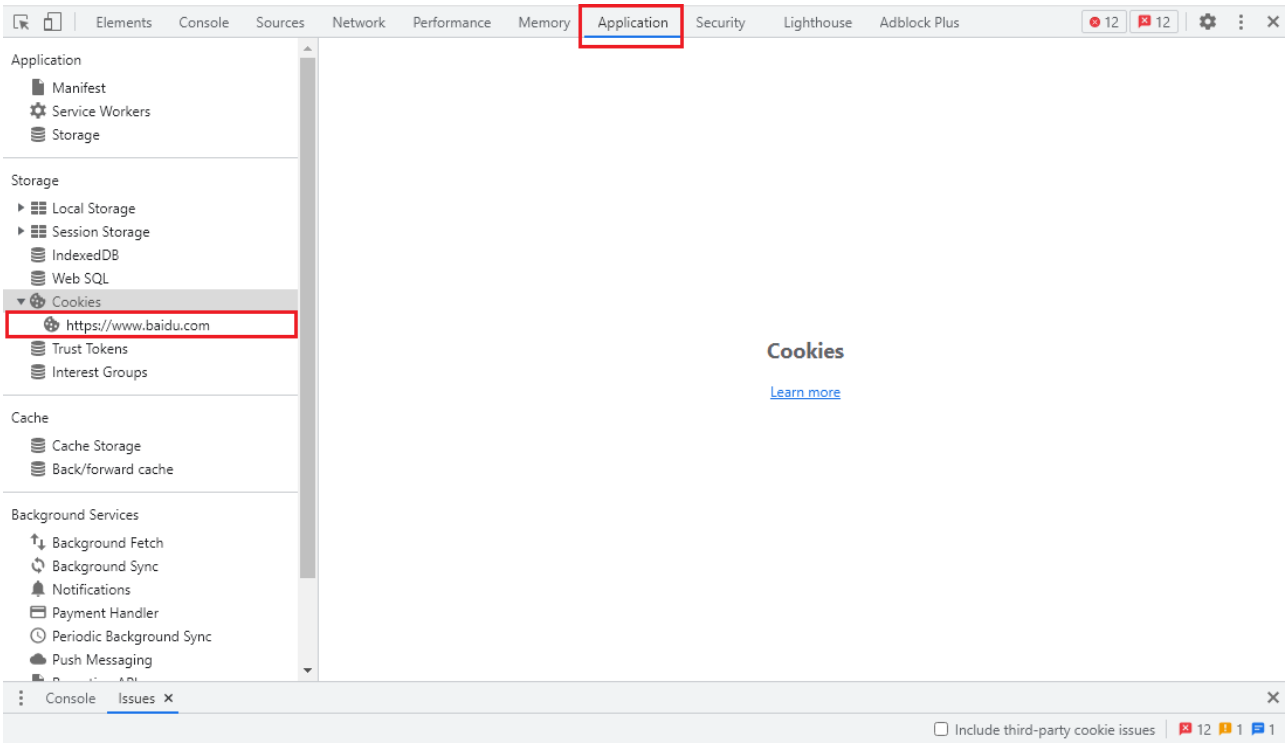
Performance

性能界面输出浏览器在加载页面、渲染、布局、等一系列操作所需的时间。



Application

使用“应用程序”面板检查、修改和调试 Web 应用程序、服务工作情况和服务相关缓存。



可参考:HTTP cookies - zzzzy09

详说 Cookie, LocalStorage 与 SessionStorage - zzzzy09

curl HTTP 请求指南

curl 是常用的命令行工具，用来请求 Web 服务器。它的名字就是客户端（client）的 URL 工具的意思。

它的功能非常强大，命令行参数多达几十种。如果熟练的话，完全可以取代 Postman 这一类的图形界面工具。本章节介绍它的主要命令行参数，作为日常的参考，方便查阅。

它支持许多开箱即用的协议，包括 HTTP、HTTPS、FTP、FTPS、SFTP、IMAP、SMTP、POP3 等等。

它是通用的，可以在 Linux、Mac、Windows 上运行。请参阅官方安装指南将其安装到您的系统上。

HTTP GET 请求

当您执行请求时，curl 将返回响应的正文

```
curl https://www.cnblogs.com/zy09/
```

获取**HTTP** 响应标头

默认情况下，响应标头隐藏在 curl 的输出中。要显示它们，请使用 i 选项

```
curl -i https://www.cnblogs.com/zy09/
```

仅获取 **HTTP** 响应标头

使用 I 选项，您只能获取标头，而不能获取响应正文

```
curl -I https://www.cnblogs.com/zy09/
```

执行 **HTTP POST** 请求

X 选项允许您更改使用的 HTTP 方法。默认使用GET，和写一样

```
curl -I -X GET https://www.cnblogs.com/zy09/
HTTP/1.1 200 OK
Date: Tue, 13 Sep 2022 12:22:51 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
```

```
Connection: keep-alive
Vary: Accept-Encoding
Cache-Control: no-cache, no-store
Pragma: no-cache
Set-Cookie: .AspNetCore.Antiforgery.b8-
pDmTq1XM=CfDJ8EOBBtWq0dNFoDS-ZHPSe53_wsO-
Mjyj49aTpB9j2RfXTJrVnuX3wS1wkqlXZqKWsuBhYDwkHQiQdQhMF64mnH_2qjgydU5
Vlsmyo2GwXfWDqmEAtO9119HK-MdnIbcRwdCnC6ZtcsVdtrcXztm-Mq0; path=/;
httponly
Strict-Transport-Security: max-age=2592000; includeSubDomains;
preload
X-Frame-Options: SAMEORIGIN


curl -I -X POST https://www.cnblogs.com/zy09/
HTTP/1.1 200 OK
Date: Tue, 13 Sep 2022 12:23:42 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Cache-Control: no-cache, no-store
Pragma: no-cache
Set-Cookie: .AspNetCore.Antiforgery.b8-
pDmTq1XM=CfDJ8EOBBtWq0dNFoDS-
ZHPSe5268BMlz29I8jPLV1FagMcveJCWGbOlSxxA-TDfgb5qI17-
U1KAuPBoGcMuLDdvcKrjXPotN8riIQhjoIeltGv9VR3LOUUN4pr2r3eMBGlsTX_GnAE
-vhYanRyB3y__BMA; path=/; httponly
Strict-Transport-Security: max-age=2592000; includeSubDomains;
preload
X-Frame-Options: SAMEORIGIN
```

可以使用 -d 传递参数

```
curl -d "option=value&something=anothervalue"  -I -X POST
https://www.cnblogs.com/zy09/
```

发送 **JSON** 的 **HTTP POST** 请求

您可能希望发送 JSON，而不是像上面的示例那样发布 URL 编码的数据。在这种情况下，您需要使用 H 选项显式设置 Content-Type 标头

```
curl -i   -H "Content-Type: application/json"
 -X POST https://www.cnblogs.com/zy09/
-d "{\"option\": \"value\", \"something\": \"anothervalue\"}"
```

windows，-d参数的数据，需要使用双引号，json里的双引号使用反斜杠转义才可以

-X：指定http请求的方法。如果使用了-d，默认是使用POST，可以省略-X参数。

Follow redirect

像 301 这样指定 Location 响应标头的重定向响应可以通过指定 L 选项自动跟随

```
curl -i -L -I  http://www.cnblogs.com/zy09/
HTTP/1.1 301 Moved Permanently
Date: Tue, 13 Sep 2022 12:39:24 GMT
Connection: keep-alive
Location: https://www.cnblogs.com/zy09/

HTTP/1.1 200 OK
Date: Tue, 13 Sep 2022 12:39:25 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Vary: Accept-Encoding
Cache-Control: no-cache, no-store
Pragma: no-cache
Set-Cookie: .AspNetCore.Antiforgery.b8-
pDmTq1XM=CfDJ8EOBBtWq0dNFoDS-
ZHPSe53y8ZErseuYBQ_z7u24qG09RCvTIQtbsvgxO_OuIcwGdx2zw4HKATzrts3u8PP
falXC21dYZThImwWMfNFGZ62_ufId7ItsksFmGJP61ZpKNg45PtPPCzMWnbnzHWfdX5
Y; path=/; httponly
Strict-Transport-Security: max-age=2592000; includeSubDomains;
preload
X-Frame-Options: SAMEORIGIN
```

检查请求和响应的所有细节

使用 --verbose 选项使 curl 输出请求的所有详细信息和响应

```
   curl --verbose -I  http://www.cnblogs.com/zy09/

* Trying 121.40.43.188:80...
```

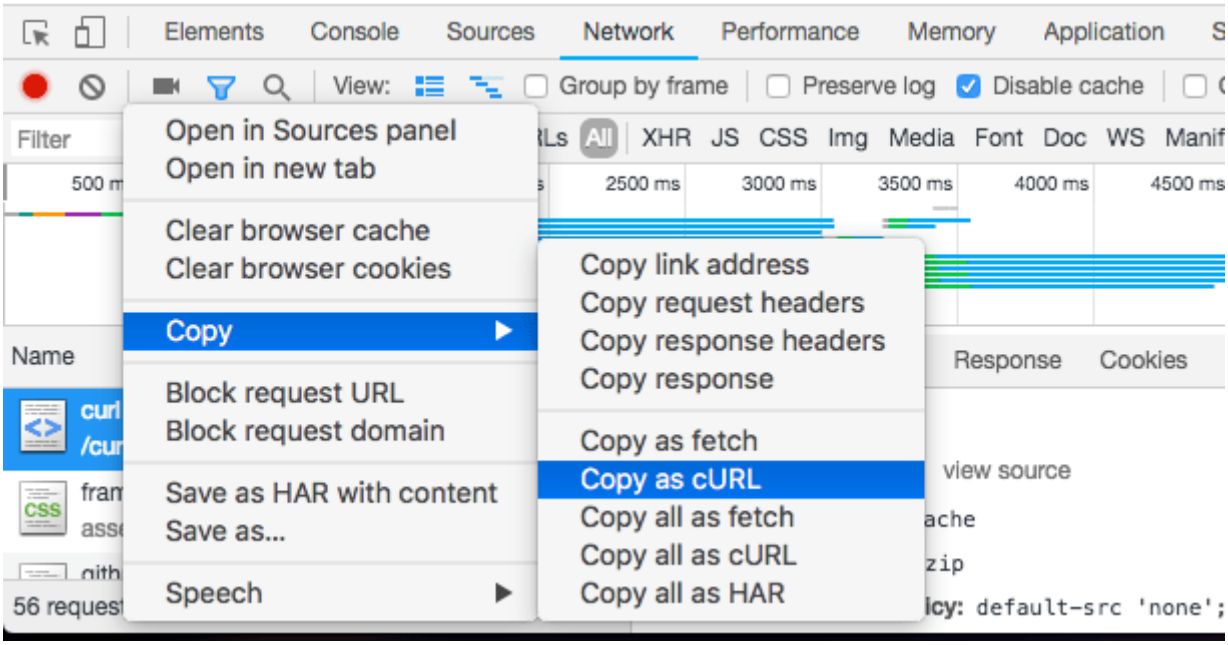
```
* Connected to www.cnblogs.com (121.40.43.188) port 80 (#0)

  > HEAD /zy09/ HTTP/1.1
  > Host: www.cnblogs.com
  > User-Agent: curl/7.83.1
  > Accept: */*
* Mark bundle as not supporting multiuse
  < HTTP/1.1 301 Moved Permanently
  HTTP/1.1 301 Moved Permanently
  < Date: Tue, 13 Sep 2022 12:41:05 GMT
  Date: Tue, 13 Sep 2022 12:41:05 GMT
  < Connection: keep-alive
  Connection: keep-alive
  < Location: https://www.cnblogs.com/zy09/
  Location: https://www.cnblogs.com/zy09/



* Connection #0 to host www.cnblogs.com left intact
```

复制浏览器网络请求到 **curl** 命令



参考链接:

https://curl.se/docs/

https://catonmat.net/cookbooks/curl

“Wer mit Ungeheuern kämpft, mag zusehn, daß er nicht dabei zum Ungeheuer wird. Und wenn du lange in einen Abgrund blickst, blickt der Abgrund auch in dich hinein.”

-- Friedrich Nietzsche, Beyond Good and Evil