

✓ 认识 HTTP

HTTP 是一种 超文本传输协议 (Hypertext Transfer Protocol)

超文本传输协议可以进行文字分割：超文本 (Hypertext)、传输 (Transfer)、协议 (Protocol)，关系如下



按照范围的大小 协议 > 传输 > 超文本。下面就分别对这三个名次做一个解释。

什么是超文本

当我们浏览一个页面，要先把页面所对应的文件从提供这个文件的计算机里，通过 Internet 传送到我们自己的计算机中，再由 WWW 浏览器翻译成为我们见到的有文字、有图形甚至有声音的页面。这些页面对应的文件不再是普通的“文本文件”，文件中除包含文字信息外，还包括了一些**具体的链接**。这些包含链接的文件被称为 超文本文件。

和普通文本相比，超文本文件中多了一些对文件内容的注释，这些注释表明了当前文字显示的位置、颜色等信息，更重要的是，在有些注释中包含了对用户计算机应做出何种反应的说明，这些注释的内容经过浏览器的翻译后就成了不同的操作。

什么是传输

上面提到，两台计算机需要进行通信，超文本会被解析为二进制数据包，由传输载体（例如同轴电缆、电话线、光缆）负责把二进制数据包计算机终端传输到另一个终端的过程，称为 传输。

通常，传输数据包的一方称为 请求方，把接到二进制数据包的一方称为 响应方。请求方和响应方可以进行互换，请求方也可以作为响应方接收数据，响应方也可以作为请求方请求数据。

什么是协议

协议，网络协议的简称，网络协议是通信计算机双方必须共同遵从的一组约定。如怎么样建立连接、怎么样互相识别等。

那么 HTTP，就是一个简单的请求-响应协议，它规范个约定了请求方、响应方之间应该传输怎样的文字、图片、音频、视频等超文本数据，以及可能会得到什么样的响应。

那么 HTTP 就是一个在计算机世界里专门在两点之间传输文字、图片、音频、视频等超文本数据的约定和规范

✓ 与 HTTP 有关的概念

随着网络世界演进，HTTP 协议已经几乎成为不可替代的一种协议，在了解了 HTTP 的基本组成后，下面再来带你进一步认识一下 HTTP 协议。

网络模型

网络是一个复杂的系统，不仅包括大量的应用程序、端系统、通信链路、分组交换机等，还有各种各样的协议组成，那么现在我们就来聊一下网络中的协议层次。

为了给网络协议的设计提供一个结构，设计者以 分层(layer) 的方式组织协议，每个协议属于层次模型之一。每一层都是向它的上一层提供 服务(service)，即所谓的 服务模型(service model)。每个分层中所有的协议称为 协议栈(protocol stack)。因特网的协议栈由五个部分组成：物理层、链路层、网络层、运输层和应用层。我们采用自上而下的方法研究其原理，也就是应用层 -> 物理层的方式。

应用层

应用层是网络应用程序和网络协议存放的分层，因特网的应用层包括许多协议，例如我们学 web 离不开的 HTTP，电子邮件传送协议 SMTP、端系统文件上传协议 FTP、还有为我们进行域名解析的 DNS 协议。应用层协议分布在多个端系统上，一个端系统应用程序与另外一个端系统应用程序交换信息分组，我们把位于应用层的信息分组称为 报文(message)。

运输层

因特网的运输层，在应用程序断点之间传送应用程序报文，在这一层主要有两种传输协议 TCP 和 UDP，利用这两者中的任何一个都能够传输报文，不过这两种协议有巨大的不同。

TCP 向它的应用程序提供了面向连接的服务，它能够控制并确认报文是否到达，并提供了拥塞机制来控制网络传输，因此当网络拥塞时，会抑制其传输速率。

UDP 协议向它的应用程序提供了无连接服务。它不具备可靠性的特征，没有流量控制，也没有拥塞控制。我们把运输层的分组称为 报文段(segment)

网络层

因特网的网络层负责将称为 数据报(datagram) 的网络分层从一台主机移动到另一台主机。网络层一个非常重要的协议是 IP 协议，所有具有网络层的因特网组件都必须运行 IP 协议，IP 协议是一种网际协议，除了 IP 协议外，网络层还包括一些其他网际协议和路由选择协议，一般把网络层就称为 IP 层，由此可知 IP 协议的重要性。

链路层

链路层用来处理连接网络的硬件部分，包括控制操作系统、硬件设备驱动、网络适配器以及光纤等物理可见部分。

硬件上的范畴都在链路层的作用范围之内。

链路层的例子包括以太网、WiFi 和电缆接入的 DOCSIS 协议（有线电视数据服务接口规范），因为数据从源目的地传送通常需要经过几条链路，一个数据包可能被沿途不同的链路层协议处理，我们把链路层的分组称为 帧(frame)。

物理层

虽然链路层的作用是将帧从一个端系统运输到 -> 另一个端系统，而物理层的作用是将帧中的一个 比特 从一个节点运输到另一个节点，物理层的协议仍然使用链路层协议，这些协议与实际的物理传输介质有关，例如，以太网有很多物理层协议：关于双绞铜线、关于同轴电缆、关于光纤等等。

五层网络协议的示意图如下



OSI 模型

我们上面讨论的计算网络协议模型不是唯一的 **协议栈**，ISO（国际标准化组织）提出来计算机网络应该按照7层来组织，那么7层网络协议栈与5层的区别在哪里？



从图中可以一眼看出，OSI 要比上面的网络模型多了 **表示层** 和 **会话层**，其他层基本一致。表示层主要包括数据压缩和数据加密以及数据描述，数据描述使得应用程序不必担心计算机内部存储格式的问题，而会话层提供了数据交换的定界和同步功能，包括建立检查点和恢复方案。

浏览器

浏览器正式的名字叫做 **Web Broser**，顾名思义，就是检索、查看互联网上网页资源的应用程序，名字里的 Web，实际上指的就是 **world wide web**，也就是万维网。

Web 服务器

Web 服务器的正式名称叫做 **web server**，Web 服务器一般指的是网站服务器，上面说到浏览器是 HTTP 请求的发起方，那么 Web 服务器就是 HTTP 请求的应答方，Web 服务器可以向浏览器等 Web 客户端提供文档，也可以放置网站文件，让全世界浏览；可以放置数据文件，让全世界下载。目前最主流的三个 Web 服务器是 Apache（稳定、开源、跨平台）、Nginx（轻量级高并发）、IIS。

CDN

CDN 的全称是 **Content Delivery Network**，即 **内容分发网络**，它应用了 HTTP 协议里的缓存和代理技术，代替源站响应客户端的请求。CDN 是构建在现有网络基础之上的网络，它依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户 **就近** 获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。CDN 的关键技术主要有 **内容存储** 和 **分发技术**。

打比方说你要去亚马逊上买书，之前你只能通过购物网站购买后从美国发货过海关等重重关卡送到你的家里，现在在中国建立一个亚马逊分基地，你就不用通过美国进行邮寄，从中国就能把书尽快给你送到。

WAF

WAF 是一种 Web 应用程序防护系统（Web Application Firewall，简称 WAF），它是一种通过执行一系列针对 HTTP / HTTPS 的 **安全策略** 来专门为 Web 应用提供保护的一款产品，它是应用层面的 **防火墙**，专门检测 HTTP 流量，是防护 Web 应用的安全技术。

WAF 通常位于 Web 服务器之前，可以阻止如 SQL 注入、跨站脚本等攻击，目前应用较多的一个开源项目是 ModSecurity（一个入侵探测与阻止的引擎），它能够完全集成进 Apache 或 Nginx。

WebService

WebService 是一种 Web 应用程序，**WebService 是一种跨编程语言和跨操作系统平台的远程调用技术。**

Web Service 是一种由 W3C 定义的应用服务开发规范，使用 client-server 主从架构，通常使用 WSDL 定义服务接口，使用 HTTP 协议传输 XML 或 SOAP（一个简单的基于 XML 的协议）消息，它是一个 **基于 Web（HTTP）的服务架构技术**，既可以运行在内网，也可以在适当保护后运行在外网。

HTML

HTML 称为超文本标记语言，是一种标识性的语言。它包括一系列标签。通过这些标签可以将网络上的文档格式统一，使分散的 Internet 资源连接为一个逻辑整体。HTML 文本是由 HTML 命令组成的描述性文本，HTML 命令可以说明文字、图形、动画、声音、表格、链接等。

Web 页面构成

Web 页面 (Web page) 也叫做文档，是由一个个对象组成的。一个 **对象(Object)** 只是一个文件，比如一个 HTML 文件、一个 JPEG 图形、一个 Java 小程序或一个视频片段，它们在网络中可以通过 **URL** 地址寻址。多数的 Web 页面含有一个 **HTML 基本文件** 以及几个引用对象。

举个例子，如果一个 Web 页面包含 HTML 文件和5个 JPEG 图形，那么这个 Web 页面就有6个对象：一个 HTML 文件和5个 JPEG 图形。HTML 基本文件通过 URL 地址引用页面中的其他对象。

✓ 与 HTTP 有关的协议

在互联网中，任何协议都不会单独的完成信息交换，HTTP 也一样。虽然 HTTP 属于应用层的协议，但是它仍然需要其他层次协议的配合完成信息的交换，那么在完成一次 HTTP 请求和响应的过程中，需要一下协议的配合。

TCP/IP

TCP/IP 协议，TCP/IP 我们一般称之为 **协议簇**，就是 TCP/IP 协议簇中不仅仅只有 TCP 协议和 IP 协议，它是一系列网络通信协议的统称。而其中最核心的两个协议就是 TCP / IP 协议，其他的还有 UDP、ICMP、ARP 等等，共同构成了一个复杂但有层次的协议栈。

TCP 协议的全称是 **Transmission Control Protocol** 的缩写，意思是 **传输控制协议**，HTTP 使用 TCP 作为通信协议，这是因为 TCP 是一种可靠的协议，而 **可靠** 能保证数据不丢失。

IP 协议的全称是 **Internet Protocol** 的缩写，它主要解决的是通信双方寻址的问题。IP 协议使用 **IP 地址** 来标识互联网上的每一台计算机，就像手机的电话号码，要与他人通话必须先要知道他人的手机号码，计算机网络中信息交换必须先要知道对方的 IP 地址。

DNS

DNS 的全称是 **域名系统 (Domain Name System)**，它作为将域名和 IP 地址相互映射的一个分布式数据库，能够使人更方便地访问互联网。

比如键入 `www.google.com`，就可以访问到谷歌，DNS 将 IP 地址转换为便于人类记忆的协议就是 `DNS 协议`。

ARP

地址解析协议（Address Resolution Protocol），是一个TCP/IP协议，其基本功能为透过目标设备的IP地址，查询目标设备的MAC地址，以保证通信的顺利进行。

URI / URL

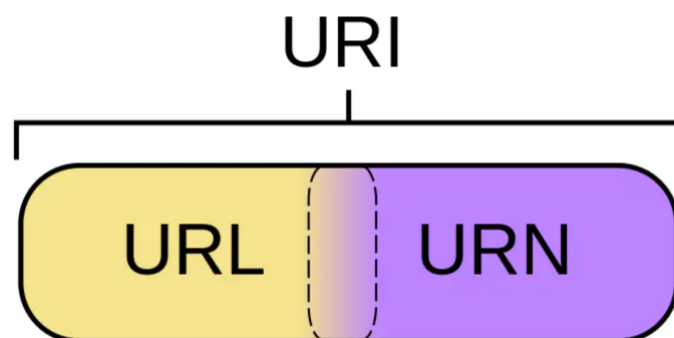
`www.google.com` 地址来访问谷歌的官网，那么输入的地址格式必须要满足 `URI` 的规范。

`URI`（Uniform Resource Identifier），统一资源标识符，使用它能够唯一地标记互联网上资源。

`URL`（Uniform Resource Locator），统一资源定位符，也就是我们俗称的 `网址`，它实际上是 URI 的一种，不仅表示了web资源，还指定了操作或者获取方式，同时指出了主要访问机制和网络位置。

`URN`（Uniform Resource Name），统一资源名称，是URI的一种，是带有特定名字的网络资源。使用URN可以在不知道其网络位置以及访问方式的情况下讨论资源。

URI、URL、URN的关系如下



URI

`http://10.195.240.53:8875/xpluginmgr-web/#/ProtocolManger/ProtocolAdd#objectAttr_1`

`http://` 定义如何访问资源的方式

`10.195.240.53:8875/xpluginmgr-web/#/ProtocolManger/ProtocolAdd` 是资源存放的位置

#objectAttr_1 是资源

HTTPS

HTTP 一般是明文传输，很容易被攻击者窃取重要信息，鉴于此，HTTPS 应运而生。

HTTPS 的全称为（Hyper Text Transfer Protocol over SecureSocket Layer），HTTPS 和 HTTP 有很大的不同在于 HTTPS 是以安全为目标的 HTTP 通道，在 HTTP 的基础上通过传输加密和身份认证保证了传输过程的安全性。HTTPS 在 HTTP 的基础上增加了 **SSL** 层，也就是说 **HTTPS = HTTP + SSL**。

✓ HTTP 请求响应过程

讨论一个问题：在键入URI到页面展示发生了什么？

假设访问的 URL 地址为 <http://www.someSchool.edu/someDepartment/home.index>，当我们输入网址并点击回车时，浏览器内部会进行如下操作



- DNS服务器会首先进行域名的映射，找到访问 www.someSchool.edu 所在的地址，再通过ARP请求找到该地址对应的mac地址，然后HTTP 客户端进程在 80 端口发起一个到服务器 www.someSchool.edu 的 TCP 连接（80 端口是 HTTP 的默认端口）。在客户和服务进程中都会有一个 **套接字** 与其相连。
- HTTP 客户端通过它的套接字向服务器发送一个 **HTTP 请求报文**。该报文中包含了路径 [someDepartment/home.index](http://www.someSchool.edu/someDepartment/home.index) 的资源。
- HTTP 服务器通过它的套接字接受该报文，进行请求的解析工作，并从其 **存储器(RAM 或磁盘)**中检索出对象 www.someSchool.edu/someDepartment/home.index，然后把检索出来的对象进行封装，封装到 **HTTP 响应报文** 中，并通过套接字向客户进行发送。
- HTTP 服务器随即通知 TCP 断开 TCP 连接，实际上是需要等到客户接受完响应报文后才会断开 TCP 连接。
- HTTP 客户端接受完响应报文后，TCP 连接会关闭。HTTP 客户端从响应中提取出报文中是一个 HTML 响应文件，并检查该 HTML 文件，然后循环检查报文中其他内部对象。
- 检查完成后，HTTP 客户端会把对应的资源通过显示器呈现给用户。

至此，键入网址再按下回车的全过程就结束了。上述过程描述的是一种简单的 **请求-响应** 全过程，真实的请求-响应情况可能要比上面描述的过程复杂很多。

具体可以看<https://juejin.cn/post/6945260497864228871#heading-0>

✓ HTTP 请求特征

从上面整个过程中我们可以总结出 HTTP 进行分组传输是具有以下特征

- 支持客户-服务器模式
- 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
- 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
- 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
- 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

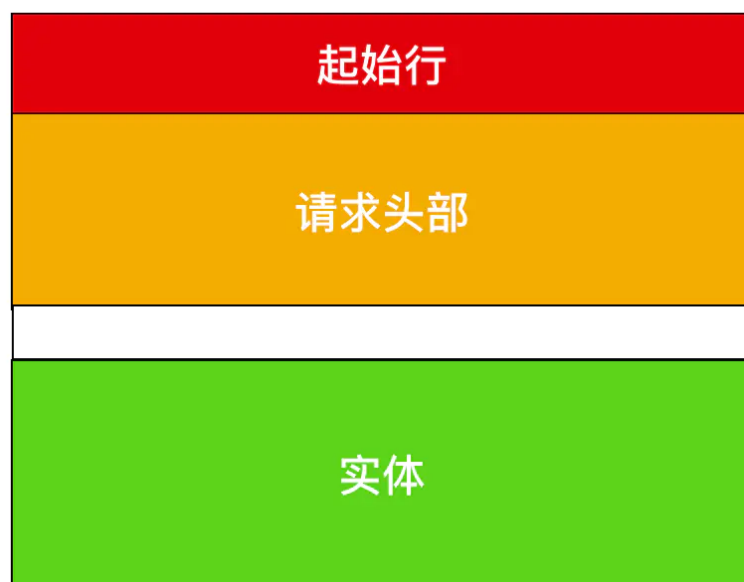
✓ 详解 HTTP 报文

我们上面描述了一下 HTTP 的请求响应过程，下面就来探讨一下 **HTTP 报文是什么样的，它的组成格式是什么？**

HTTP 协议主要由三大部分组成：

- **起始行 (start line)**：描述请求或响应的基本信息；
- **头部字段 (header)**：使用 key-value 形式更详细地说明报文；
- **消息正文 (entity)**：实际传输的数据，它不一定是纯文本，可以是图片、视频等二进制数据。

其中起始行和头部字段并成为 **请求头** 或者 **响应头**，统称为 **Header**；消息正文也叫做实体，称为 **body**。HTTP 协议规定每次发送的报文必须要有 Header，但是可以没有 body，也就是说头信息是必须的，实体信息可以没有。而且在 header 和 body 之间必须要有一个空行 (CRLF)，如果用一幅图来表示一下的话，我觉得应该是下面这样



我们使用上面的那个例子来看一下 http 的请求报文

起始行
`GET /somedir/page.html HTTP/1.1`

请求头部
`Host: www.someschool.edu`
`Connection: close`
`User-agent: Mozilla/5.0`
`Accept-language: fr`

空行

如图，这是 `http://www.someSchool.edu/someDepartment/home.index` 请求的请求头，可以看到

- 每个报文的起始行都是由三个字段组成：**方法**、**URL 字段**和 **HTTP 版本字段**。
- 请求头部之后由空行



(a) 请求消息

第一行称为**请求行**，通过这一行可以大致了解请求的内容

```
<方法><空格><URI><空格><HTTP版本>
<字段名>:<字段值>
...
...
<空行>
<消息体>
```

这一部分称为**消息头**，每行包含一个头字段，用于表示请求的附加信息。消息头的行数根据实际情况可变，一直延伸到空行为止

消息体 (message body) 包含客户端向服务器发送的数据，例如用POST方法向Web服务器发送的网页表单数据

(b) 响应消息

用来解释状态码的短语

```
<HTTP版本><空格><状态码><空格><响应短语>
<字段名>:<字段值>
...
...
<空行>
<消息体>
```

消息头

状态行

消息体包含服务器向客户端发送的数据，例如从文件中读取的数据，或者CGI应用程序输出的数据等。消息体的内容作为二进制数据来处理^①

get请求

```
GET /xpluginmgr-web/web/protocolManager/getDataNameByTaskType.do?apsName=53acsneg&protocol=%E6%B5%B7%E5%BA%B7%E6%95%B0%E6%8D%AE%E5%AF%B9%E6%8E
%A5%E6%A0%B7%E5%B7%B6%E8%A7%B4%E8%8C%83&taskType=0&t=1618920592408 HTTP/1.1
Host: 10.195.240.53:8875
Connection: keep-alive
Accept: application/json, text/plain, */*
X-CSRF-TOKEN:
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36
X-Requested-With: XMLHttpRequest
Referer: http://10.195.240.53:8875/xpluginmgr-web/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=nzKrvWdWwVVE89DghnRTQ9v54hhF1J6TffuXA7G; OPSMGRCASTGC=TGT-276-zmU9jehhwcPu2WldjtrNnISZ1CJ3j1vKeqcZTe4cQrXI690ku2-cas

HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json
traceId: 96bb6eaa719c4ae888adc2af6032772c
Date: Tue, 20 Apr 2021 12:07:18 GMT

{"code": "0", "msg": "SUCCESS", "data": {"aps":
[
.....
]]}GET /xpluginmgr-web/web/protocolManager/getDataNameByTaskType.do?apsName=53acsneg&protocol=
```

post请求

```
,....."]}]POST /xpluginmgr-web/web/task/verify.do HTTP/1.1
Host: 10.195.240.53:8875
Connection: keep-alive
Content-Length: 1167
Accept: application/json, text/plain, */*
X-CSRF-TOKEN:
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36
Content-Type: application/json
Origin: http://10.195.240.53:8875
Referer: http://10.195.240.53:8875/xpluginmgr-web/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=nzKrvwWdwvwVE89DGhnRTQ9v54hhf1J6TffuXA7G; OPSMGRCASTGC=TGT-276-zmU9jehhwcPu2WNdjtrNnI5Z1CJ3jlvKeqcZTe4cQrXI690ku2-cas

{"taskName": ".....", "taskType": 0, "system": [{"apsId": "12345678901234567890", "apsName": "53acsnscg", "protocol": ".....", "apsIp": "http://10.195.240.53:8080/acsncg"}], "dataReceiver": {"protocol": ".....", "apsId": "0", "apsName": ".....", "apsIp": null, "signature": null, "signatureType": null}, "protocolConversionName": ".....", "targetDataTypes": {"aps": [{"component": []}, {"urlStoreType": {"deviceID": "f1c687c5-3248-4dc5-9b27-d729380090bf", "deviceName": "Access Storage for Windows-10.195.240.53-#1", "deviceIP": "10.195.240.53", "devicePort": 6021, "picUploadPort": 6011, "picDownloadPort": 6120, "deviceType": 3, "userName": "admin", "userPwd": "md4MEX3+Ay8PHVdtGD03Ug==", "accessKey": "1c2VTwMyop1KAd", "secretKey": "j/TLESf0mpN7tpPLLhf9iF9bcU2gsRV", "netZoneID": "0", "poolInfoList": [{"poolID": "10000000", "poolName": "UTADefaultNotOverwritePool", "poolType": 0, "deviceType": 3, "poolCycle": 7, "coverType": 2, "lockLimit": 15, "safeType": null, "size": 30720, "freeSize": 30720, "poolStatus": 1, "beginTime": "00-00-00T00:00:00.000Z", "coverTime": "00-00-00T00:00:00.000Z", "domainID": ""}], "id": 9, "dataPipeId": 751}]}HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json
traceId: c25a17f321c946a68c354ccf068b7ebe
Date: Tue, 20 Apr 2021 12:07:24 GMT

{"code": "0", "msg": "SUCCESS", "data": [9]}POST /xpluginmgr-web/web/task/add.do HTTP/1.1
```

▼ General	
Request URL: http://10.195.240.53:8875/xpluginmgr-web/web/protocolManager/getDataNameByTaskType.do?apsName=53acsnscg&protocol=%E6%B5%B7%E5%BA%B7%E6%95%B0%E6%8D%AE%E5%AF%B9%E6%8E%A5%E6%A0%87%E5%87%86%E8%A7%84%E8%8C%83&taskType=0&t=1618920592408	
Request Method: GET	
Status Code: 200 OK	
Remote Address: 10.195.240.53:8875	
Referrer Policy: no-referrer-when-downgrade	
▼ Response Headers	view source
Connection: keep-alive	
Content-Type: application/json	
Date: Tue, 20 Apr 2021 12:07:20 GMT	
traceld: 726e029439c64f6eb0f1b1614994d1d6	
Transfer-Encoding: chunked	
▼ Request Headers	view source
Accept: application/json, text/plain, */*	
Accept-Encoding: gzip, deflate	
Accept-Language: zh-CN,zh;q=0.9	
Connection: keep-alive	
Cookie: JSESSIONID=nzKrvwWdwvwVE89DGhnRTQ9v54hhf1J6TffuXA7G; OPSMGRCASTGC=TGT-276-zmU9jehhwcPu2WNdjtrNnI5Z1CJ3jlvKeqcZTe4cQrXI690ku2-cas	
Host: 10.195.240.53:8875	
Referer: http://10.195.240.53:8875/xpluginmgr-web/	
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36	
X-CSRF-TOKEN	
X-Requested-With: XMLHttpRequest	

HTTP 请求方法

HTTP 请求方法一般分为 8 种，它们分别是

- **GET 获取资源**，用来请求访问已被 URI 识别的资源。指定的资源经服务器端解析后返回响应内容。也就是说，如果请求的资源是文本，那就保持原样返回；
- **POST 传输实体**，虽然 GET 方法也可以传输主体信息，但是便于区分，我们一般不用 GET 传输实体信息，反而使用 POST 传输实体信息，

- **PUT 传输文件**，用来传输文件。就像 FTP 协议的文件上传一样，要求在请求报文的主体中包含文件内容，然后保存到请求 URI 指定的位置。

但是，鉴于 HTTP 的 PUT 方法自身不带验证机制，任何人都可以上传文件，存在安全性问题，因此一般的 Web 网站不使用该方法。

- **HEAD 获得响应首部**，和 GET 方法一样，只是不返回报文主体部分。用于确认 URI 的有效性 & 资源更新的日期时间等。
- **DELETE 删除文件**，用来删除文件，是与 PUT 相反的方法。DELETE 方法按请求 URI 删除指定的资源。
- **OPTIONS 询问支持的方法**，用来查询针对请求 URI 指定的资源支持的方法。
- **TRACE 追踪路径**，是让 Web 服务器端将之前的请求通信环回给客户端的方法。
- **CONNECT** 要求用隧道协议连接代理，CONNECT 方法要求在与代理服务器通信时建立隧道，实现用隧道协议进行 TCP 通信。主要使用 **SSL (Secure Sockets Layer, 安全套接层)** 和 **TLS (Transport Layer Security, 传输层安全)** 协议把通信内容加密后经网络隧道传输。

我们一般最常用的方法也就是 GET 方法和 POST 方法，其他方法暂时了解即可。下面是 HTTP1.0 和 HTTP1.1 支持的方法清单

方法	说明	支持的 HTTP 协议版本
GET	获取资源	1.0、1.1
POST	传输实体主体	1.0、1.1
PUT	传输文件	1.0、1.1
HEAD	获得报文首部	1.0、1.1
DELETE	删除文件	1.0、1.1
OPTIONS	询问支持的方法	1.1
TRACE	追踪路径	1.1
CONNECT	要求用隧道协议连接代理	1.1
LINK	建立和资源之间的联系	1.0
UNLINE	断开连接关系	1.0

HTTP 请求 URL

HTTP 协议使用 URI 定位互联网上的资源。正是因为 URI 的特定功能，在互联网上任意位置的资源都能访问到。URL 带有请求对象的标识符。在上面的例子中，浏览器正在请求对象 `/somedir/page.html` 的资源。

我们再通过一个完整的域名解析一下 URL

比如 `http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument`。

首先出场的是 `http`

方案或协议

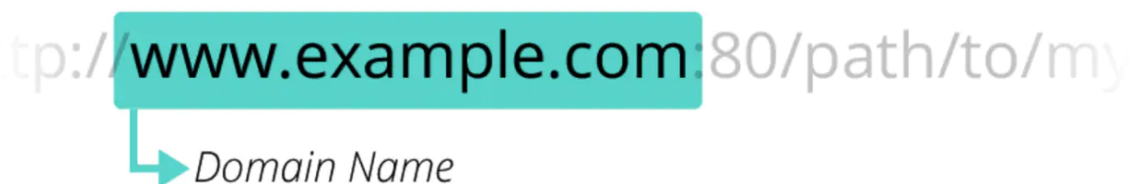


`http://` *Protocol*

`http://` 告诉浏览器使用何种协议。对于大部分 Web 资源，通常使用 HTTP 协议或其安全版本，HTTPS 协议。另外，浏览器也知道如何处理其他协议。例如，`mailto:` 协议指示浏览器打开邮件客户端；`ftp:` 协议指示浏览器处理文件传输。

第二个出场的是 `主机`

主机




`www.example.com` *Domain Name*

`www.example.com` 既是一个域名，也代表管理该域名的机构。它指示了需要向网络上的哪一台主机发起请求。当然，也可以直接向主机的 [IP address](#) 地址发起请求。但直接使用 IP 地址的场景并不常见。

第三个出场的是 `端口`

端口

com:80/path/to/myfile.html?key1=value1



我们前面说到，两个主机之间要发起 TCP 连接需要两个条件，主机 + 端口。它表示用于访问 Web 服务器上资源的入口。如果访问的该 Web 服务器使用 HTTP 协议的标准端口（HTTP 为 80，HTTPS 为 443）授予对其资源的访问权限，则通常省略此部分。否则端口就是 URI 必须的部分。

上面是请求 URL 所必须包含的部分，下面就是 URL 具体请求资源路径

第四个出场的是 路径

路径

n:80/path/to/myfile.html?key1=value1

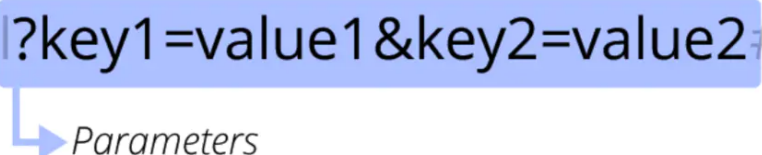


/path/to/myfile.html 是 Web 服务器上资源的路径。以端口后面的第一个 / 开始，到 ? 号之前结束，中间的每一个 / 都代表了层级（上下级）关系。这个 URL 的请求资源是一个 html 页面。

紧跟着路径后面的是 查询参数

查询

html?key1=value1&key2=value2#Some



`?key1=value1&key2=value2` 是提供给 Web 服务器的额外参数。如果是 GET 请求，一般带有请求 URL 参数，如果是 POST 请求，则不会在路径后面直接加参数。这些参数是用 & 符号分隔的 **键/值对** 列表。key1 = value1 是第一对，key2 = value2 是第二对参数

紧跟着参数的是 **锚点**

片段

ue2 **#SomewhereInTheDocument**
Anchor

#SomewhereInTheDocument 是资源本身的某一部分的一个锚点。锚点代表资源内的一种“书签”，它给予浏览器显示位于该“加书签”点的内容的指示。例如，在 HTML 文档上，浏览器将滚动到定义锚点的那个点上；在视频或音频文档上，浏览器将转到锚点代表的那个时间。值得注意的是 # 号后面的部分，也称为片段标识符，永远不会与请求一起发送到服务器。

HTTP 版本

表示报文使用的 HTTP 协议版本。

请求头部

在表述完了起始行之后再来看一下 **请求头部**，来看一下它的请求头部

```
GET /xpluginmgr-web/web/protocolManager/getDataNameByTaskType.do?apsName=53acsncg&protocol=%E6%B5%B7%E5%BA%B7%E6%95%B0%E6%8D%AE%E5%AF%B9%E6%BE%A5%E6%A0%E8%7E%58%78%6E%8A%78%4E%8C%83&taskType=0&t=1618920592408 HTTP/1.1
Host: 10.195.240.53:8875
Connection: keep-alive
Accept: application/json, text/plain, */*
X-CSRF-TOKEN:
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36
X-Requested-With: XMLHttpRequest
Referer: http://10.195.240.53:8875/xpluginmgr-web/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=nzKrvwMdwvVE89DGhnRTQ9v54hhF1J6TffuXA7G; OPSMGRCASTGC=TGT-276-zmU9jehhwcPu2WldjtrNnISZ1CJ3j1vKeqcZTe4cQrXI690ku2-cas

HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked
Content-Type: application/json
traceId: 96bb6eaa719c4ae888adc2af6032772c
Date: Tue, 20 Apr 2021 12:07:18 GMT

{"code":0,"msg":"SUCCESS","data":{"aps":
[
.....
]]}}GET /xpluginmgr-web/web/protocolManager/getDataNameByTaskType.do?apsName=53acsncg&protocol=
```

首先 Host 表示的是对象所在的主机。你也许认为这个 Host 是不需要的，因为 URL 不是已经指明了请求对象的路径了吗？这个首部行提供的信息是 **web 代理高速缓存** 所需要的。

Connection: keep-alive 表示的是浏览器需要告诉服务器使用的是 **持久连接**。它要求服务器在发送完响应的对象后就暂不关闭连接。

User-agent：告诉 Web 服务器，浏览器使用的内核是 **Mozilla/5.0**。

Accept-language 告诉 Web 服务器，浏览器想要得到对象的语法版本，前提是服务器需要支持法语类型，否则将会发送服务器的默认版本。

Accept-Encoding：指示对实体应用了何种编码

下面针对主要的实体字段进行介绍（具体的可以参考 developer.mozilla.org/zh-CN/docs/... MDN 官网学习）

HTTP 的请求标头分为四种：**通用标头**、**请求标头**、**响应标头** 和 **实体标头**，依次来进行详解。

通用标头

通用标头主要有三个，分别是 **Date**、**Cache-Control** 和 **Connection**，可以出现在请求标头和响应标头中。

Date

Date 是一个通用标头，它的基本表示如下

```
Date: Wed, 21 Oct 2015 07:28:00 GMT
```

表示的是格林威治标准时间，这个时间要比北京时间慢八个小时

[首页](#) » [时区换算](#) »

北京时间 → 格林威治标准时间

北京时间: 20:00 (8:00 PM) ◆

格林威治标准时间: 12:00 (12:00 PM)

格林威治标准时间 → 北京时间

格林威治标准时间: 12:00 (12:00 PM)) ◆

北京时间: 20:00 (8:00 PM)

换级: 00:00 ◆

Cache-Control

Cache-Control 是一个通用标头，Cache-Control 的种类比较多，虽然说这是一个通用标头，但是又一些特性是请求标头具有的，有一些是响应标头才有的。主要大类有 可缓存性、 阈值性、 重新验证并重新加载 和 其他特性

可缓存性是唯—响应标头才具有的特性，我们会在响应标头中详述。

阈值性，这个我翻译可能不准确，它的原英文是 Expiration，我是根据它的值来翻译的，你看到这些值可能会觉得我翻译的有点道理

- **max-age** : 资源被认为仍然有效的最长时间，与 **Expires** 不同，这个请求是相对于 request 标头的时间，而 Expires 是相对于响应标头。（请求标头）
- **s-maxage** : 重写了 max-age 和 Expires 请求头，仅仅适用于共享缓存，被私有缓存所忽略（这块不理解，看完响应头的 Cache-Control 再进行理解）（请求标头）
- **max-stale** : 表示客户端将接受的最大响应时间，以秒为单位。（响应标头）
- **min-fresh** : 表示客户端希望响应在指定的最小时间内有效。（响应标头）

Connection

Connection 决定当前事务（一次三次握手和四次挥手）完成后，是否会关闭网络连接。Connection 有两种，一种是 持久性连接，即一次事务完成后不关闭网络连接

```
Connection: keep-alive
```

另一种是 非持久性连接，即一次事务完成后关闭网络连接

```
Connection: close
```

HTTP1.1 其他通用标头如下

首部字段名	说明
Cache-Control	控制缓存的行为
Connection	逐跳首部、连接的管理
Date	创建报文的日期时间
Pragma	报文指令
Trailer	报文末端的首部一览
Transfer-Encoding	指定报文主体的传输编码方式
Upgrade	升级为其他协议
Via	代理服务器的相关信息
Warning	错误通知

实体标头

实体标头是描述消息正文内容的 HTTP 标头。实体标头用于 HTTP 请求和响应中。头部 `Content-Length`、`Content-Language`、`Content-Encoding` 是实体头。

- Content-Length 实体报头指示实体主体的大小，以字节为单位，发送到接收方。
- Content-Language 实体报头描述了客户端或者服务端能够接受的语言，例如

```
Content-Language: de-DE
Content-Language: en-US
Content-Language: de-DE, en-CA
```

- Content-Encoding 这又是一个比较麻烦的属性，这个实体报头用来压缩媒体类型。Content-Encoding 指示对实体应用了何种编码。

常见的内容编码有这几种：**gzip**、**compress**、**deflate**、**identity**，这个属性可以应用在请求报文和响应报文中。这样服务器就可以从中选择一个压缩算法，放进Content-Encoding响应头里，再把原数据压缩后发给浏览器。

```
Accept-Encoding: gzip, deflate //请求头
Content-Encoding: gzip //响应头
```

下面是一些实体标头字段

首部字段名	说明
Allow	资源可支持的HTTP方法
Content-Encoding	实体主体适用的编码方式
Content-Language	实体主体的自然语言
Content-Length	实体主体的大小（单位：字节）
Content-Location	替代对应资源的URI
Content-MD5	实体主体的报文摘要
Content-Range	实体主体的位置范围
Content-Type	实体主体的媒体类型
Expires	实体主体过期的日期时间
Last-Modified	资源的最后修改日期时间

请求标头

上面给出的例子请求报文的属性比较少，下面给出一个 MDN 官网的例子

```
GET /home.html HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS x 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
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, br
Referer: https://developer.mozilla.org/testpage.html
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT
If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"
Cache-Control: max-age=0
```

Host

Host 请求头指明了服务器的域名（对于虚拟主机来说），以及（可选的）服务器监听的TCP端口号。如果没有给定端口号，会自动使用被请求服务的默认端口（比如请求一个 HTTP 的 URL 会自动使用80作为端口）。

```
Host: developer.mozilla.org
```

上面的 **Accpet**、**Accept-Language**、**Accept-Encoding** 都是属于内容协商的请求标头，我们会在下面说明

Referer

HTTP Referer 属性是请求标头的一部分，当浏览器向 web 服务器发送请求的时候，一般会带上 Referer，告诉服务器该网页是从哪个页面链接过来的，服务器因此可以获得一些信息用于处理。

```
Referer: https://developer.mozilla.org/testpage.html
```

Upgrade-Insecure-Requests

Upgrade-Insecure-Requests 是一个请求标头，用来向服务器端发送信号，表示客户端优先选择加密及带有身份验证的响应。

```
Upgrade-Insecure-Requests: 1
```

If-Modified-Since

HTTP 的 If-Modified-Since 使其成为 **条件请求**：

- 返回200，只有在给定日期的最后一次修改资源后，服务器才会以200状态发送回请求的资源。
- 如果请求从开始以来没有被修改过，响应会返回304并且没有任何响应体

If-Modified-Since 通常会与 If-None-Match 搭配使用，If-Modified-Since 用于确认代理或客户端拥有的本地资源的有效性。获取资源的更新日期时间，可通过确认首部字段 **Last-Modified** 来确定。

大白话说就是如果在 **Last-Modified** 之后更新了服务器资源，那么服务器会响应 200，如果在 **Last-Modified** 之后没有更新过资源，则返回 304。

```
If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT
```

If-None-Match

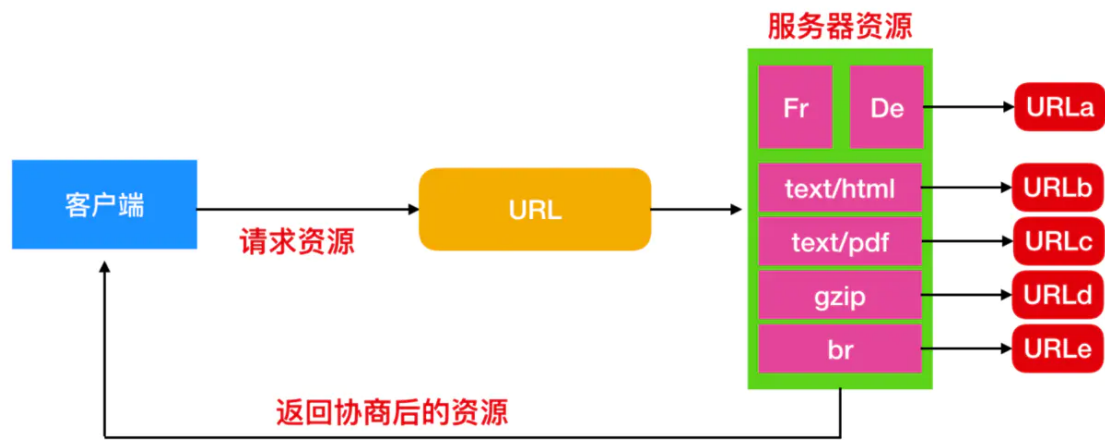
If-None-Match HTTP请求标头使请求成为条件请求。对于 GET 和 HEAD 方法，仅当服务器没有与给定资源匹配的 **ETag** 时，服务器才会以200状态发送回请求的资源。对于其他方法，仅当最终现有资源的 **ETag** 与列出的任何值都不匹配时，才会处理请求。

```
If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"
```

ETag 属于响应标头，后面进行介绍。

内容协商

内容协商机制是指客户端和服务端就响应的资源内容进行交涉，然后提供给客户端最为适合的资源。内容协商会以响应资源的语言、字符集、编码方式等作为判断的标准。



内容协商功能图

内容协商主要有以下3种类型：

- 服务器驱动协商 (Server-driven Negotiation)

这种协商方式是由服务器端进行内容协商。服务器端会根据请求首部字段进行自动处理

- 客户端驱动协商 (Agent-driven Negotiation)

这种协商方式是由客户端来进行内容协商。

- 透明协商 (Transparent Negotiation)

是服务器驱动和客户端驱动的结合体，是由服务器端和客户端各自进行内容协商的一种方法。

内容协商的分类有很多种，主要的几种类型是 **Accept**、**Accept-Charset**、**Accept-Encoding**、**Accept-Language**、**Content-Language**。

Accept

接受请求 HTTP 标头会通告客户端其能够理解的 MIME 类型

那么什么是 MIME 类型呢？在回答这个问题前你应该先了解一下什么是 MIME

MIME: MIME (Multipurpose Internet Mail Extensions) 是描述消息内容类型的因特网标准。MIME 消息能包含文本、图像、音频、视频以及其他应用程序专用的数据。

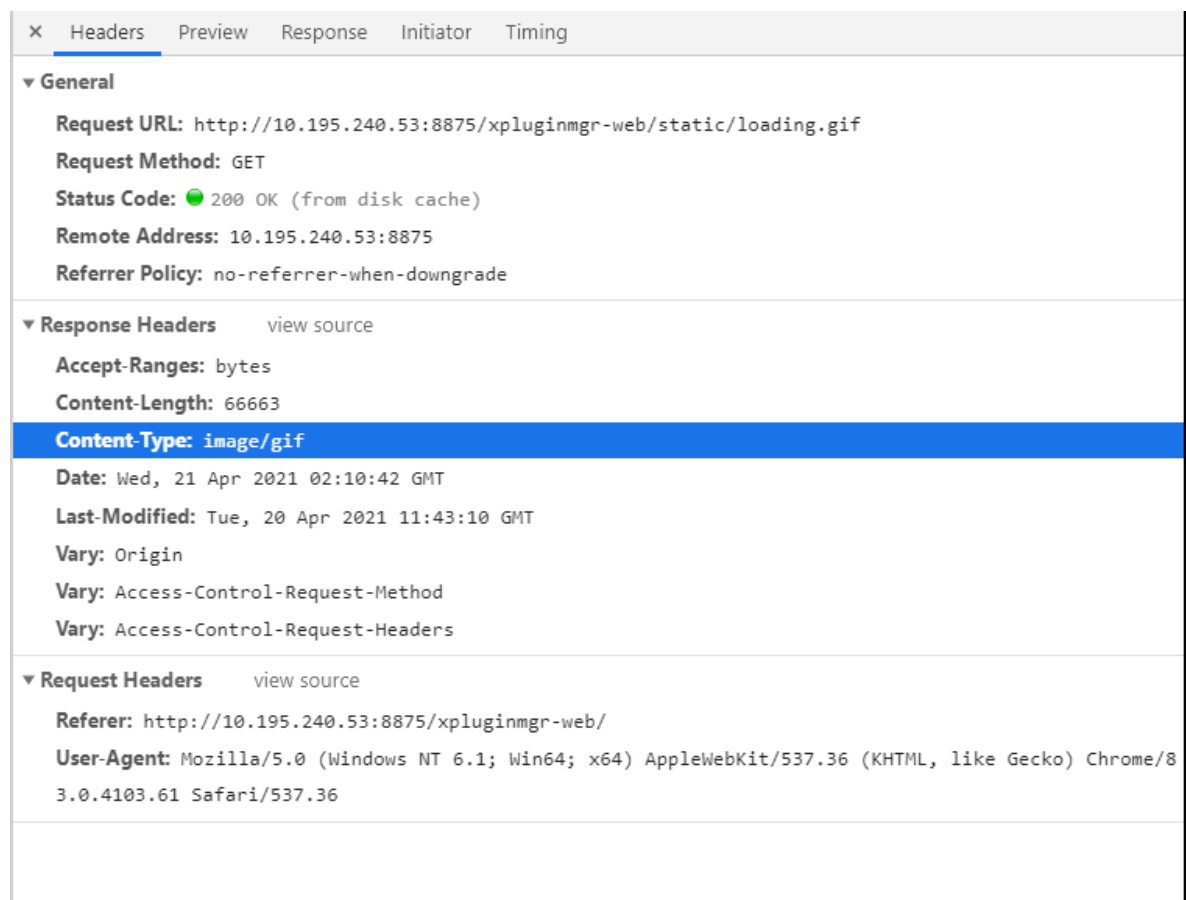
也就是说，MIME 类型其实就是一系列消息内容类型的集合。那么 MIME 类型都有哪些呢？

文本文件：text/html、text/plain、text/css、application/xhtml+xml、application/xml

图片文件：image/jpeg、image/gif、image/png

视频文件：video/mpeg、video/quicktime

应用程序二进制文件：application/octet-stream、application/zip



Request URL: http://10.195.240.53:8875/xpluginmgr-web/web/protocol/downloadTemplate.do?sourceProtocolName=test&targetProtocolName=%E6%B5%B7%E5%BA%B7%E6%95%B0%E6%8D%AE%E5%AF%B9%E6%8E%A5%E6%A0%87%E5%87%86%E8%A7%84%E8%8C%83&fileType=0&t=1618972933688	
Request Method: GET	
Status Code: ● 200 OK	
Remote Address: 10.195.240.53:8875	
Referrer Policy: no-referrer-when-downgrade	
▼ Response Headers	view source
Connection: keep-alive	
Content-Disposition: attachment;filename=test_%E6%B5%B7%E5%BA%B7%E6%95%B0%E6%8D%AE%E5%AF%B9%E6%8E%A5%E6%A0%87%E5%87%86%E8%A7%84%E8%8C%83_pc.xls	
Content-Type: application/octet-stream	
Date: Wed, 21 Apr 2021 02:39:39 GMT	
traceld: 9d99c4879b0d4e009316e292c72467bf	
Transfer-Encoding: chunked	
▼ Request Headers	view source
Accept: application/json, text/plain, */*	
Accept-Encoding: gzip, deflate	
Accept-Language: zh-CN,zh;q=0.9	
Connection: keep-alive	
Cookie: JSESSIONID=E3uqU8p2Js1jrr_vKkfZ5lQKpfFt5I2tEvxVr0D; OPSMGRCASTGC=TGT-276-zmU9jehhwcPu2wNdjtrNnISZlCJ3jlvKeqcZTe4cQrXI690ku2-cas	
Host: 10.195.240.53:8875	
Referer: http://10.195.240.53:8875/xpluginmgr-web/	
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36	
X-CSRF-TOKEN	

比如，如果浏览器不支持 PNG 图片的显示，那 Accept 就不指定 image/png，而指定可处理的 image/gif 和 image/jpeg 等图片类型。

一般 MIME 类型也会和 **q** 这个属性一起使用，q 是什么？q 表示的是权重，来看一个例子

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

复制代码

这是什么意思呢？若想要给显示的**媒体类型增加优先级**，则使用 q= 来额外表示权重值，没有显示权重的时候默认值是1.0，我给你列个表格你就明白了

q	MIME
1.0	text/html
1.0	application/xhtml+xml
0.9	application/xml
0.8	* / *

也就是说，这是一个放置顺序，权重高的在前，低的在后， `application/xml;q=`

`0.9` 是不可分割的整体。

Accept-Charset

accept-charset 属性规定服务器处理表单数据所接受的字符集。

accept-charset 属性允许您指定一系列字符集，服务器必须支持这些字符集，从而得以正确解释表单中的数据。

该属性的值是用引号包含字符集名称列表。如果可接受字符集与用户所使用的字符即不相匹配的话，浏览器可以选择忽略表单或是将该表单区别对待。

此属性的默认值是 `unknown`，表示表单的字符集与包含表单的文档的字符集相同。

常用的字符集有：UTF-8 - Unicode 字符编码；ISO-8859-1 - 拉丁字母表的字符编码

Accept-Language

首部字段 Accept-Language 用来告知服务器用户代理能够处理的自然语言集（指中文或英文等），以及自然语言集的相对优先级。可一次指定多种自然语言集。和 Accept 首部字段一样，按权重值 `q` 来表示相对优先级。

```
Accept-Language: en-US,en;q=0.5
```

请求标头我们大概就介绍这几种，后面会有一篇文章详细深挖所有的响应头的，下面是一个响应头的汇总，基于 HTTP 1.1

首部字段名	说明
Accept	用户代理可处理的媒体类型
Accept-Charset	优先的字符集
Accept-Encoding	优先的内容编码
Accept-Language	优先的语言（自然语言）
Authorization	Web认证信息
Expect	期待服务器的特定行为
From	用户的电子邮箱地址
Host	请求资源所在服务器
If-Match	比较实体标记（ETag）
If-Modified-Since	比较资源的更新时间
If-None-Match	比较实体标记（与 If-Match 相反）
If-Range	资源未更新时发送实体 Byte 的范围请求
If-Unmodified-Since	比较资源的更新时间（与If-Modified-Since相反）
Max-Forwards	最大传输逐跳数
Proxy-Authorization	代理服务器要求客户端的认证信息
Range	实体的字节范围请求
Referer	对请求中 URI 的原始获取方
TE	传输编码的优先级
User-Agent	HTTP 客户端程序的信息

响应标头

响应标头是可以在 HTTP 响应种使用的 HTTP 标头。并不是所有出现在响应中的标头都是响应标头。还有一些特殊的我们上面说过，有通用标头和实体标头也会出现在响应标头中，比如 **Content-Length** 就是一个实体标头，但是，在这种情况下，这些实体请求通常称为响应头。下面以一个例子为例和你探讨一下响应头

```
200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Mon, 18 Jul 2016 16:06:00 GMT
Etag: "c561c68d0ba92bbeb8b0f612a9199f722e3a621a"
Keep-Alive: timeout=5, max=997
Last-Modified: Mon, 18 Jul 2016 02:36:04 GMT
Server: Apache
Set-Cookie: mykey=myvalue; expires=Mon, 17-Jul-2017 16:06:00 GMT; Max-Age=31449600; Path=/; secure
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
x-frame-options: DENY
```

响应状态码

首先出现的应该就是 **200 OK**，这是 HTTP 响应标头的状态码，它表示着响应成功完成。HTTP 响应标头的状态码有很多，并做了如下规定

以 **2xx** 为开头的都表示请求成功响应。

状态码	含义
200	成功响应
204	请求处理成功，但是没有资源可以返回
206	对资源某一部分进行响应，由Content-Range 指定范围的实体内容。

以 **3xx** 为开头的都表示需要进行附加操作以完成请求

状态码	含义
301	永久性重定向，该状态码表示请求的资源已经重新分配 URI，以后应该使用资源现有的 URI
302	临时性重定向。该状态码表示请求的资源已被分配了新的 URI，希望用户（本次）能使用新的 URI 访问。
303	该状态码表示由于请求对应的资源存在着另一个 URI，应使用 GET 方法定向获取请求的资源。
304	该状态码表示客户端发送附带条件的请求时，服务器端允许请求访问资源，但未满足条件的情况。
307	临时重定向。该状态码与 302 Found 有着相同的含义。

以 **4xx** 的响应结果表明客户端是发生错误的原因所在。

状态码	含义
400	该状态码表示请求报文中存在语法错误。当错误发生时，需修改请求的内容后再次发送请求。
401	该状态码表示发送的请求需要有通过 HTTP 认证（BASIC 认证、DIGEST 认证）的认证信息。
403	该状态码表明对请求资源的访问被服务器拒绝了。
404	该状态码表明服务器上无法找到请求的资源。

以 **5xx** 为开头的响应标头都表示服务器本身发生错误

状态码	含义
500	该状态码表明服务器端在执行请求时发生了错误。
503	该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

Access-Control-Allow-Origin

一个返回的 HTTP 标头可能会具有 Access-Control-Allow-Origin，**Access-Control-Allow-Origin** 指定一个来源，它告诉浏览器允许该来源进行资源访问。否则-对于没有凭据的请求 * 通配符，告诉浏览器允许任何源访问资源。例如，要允许源 **https://mozilla.org** 的代码访问资源，可以指定：

```
Access-Control-Allow-Origin: https://mozilla.org
Vary: Origin
```

如果服务器指定单个来源而不是 * 通配符的话，则服务器还应在 Vary 响应标头中包含 **Origin**，以向客户端指示 服务器响应将根据原始请求标头的值而有所不同。

Keep-Alive

上面我们提到，HTTP 报文标头会分为四种，这其实是按着 **上下文** 来分类的
还有一种分类是根据 **代理** 进行分类，根据代理会分为 **端到端头** 和 **逐跳标头**
而 Keep-Alive 表示的是 Connection 非持续连接的存活时间，如下

```
Connection: Keep-Alive
Keep-Alive: timeout=5, max=997
```

Keep-Alive 有两个参数，它们是以逗号分隔的参数列表，每个参数由一个标识符和一个由等号 = 分隔的值组成。

timeout：指示空闲连接必须保持打开状态的最短时间（以秒为单位）。

max：指示在关闭连接之前可以在此连接上发送的最大请求数。

上述 HTTP 代码的意思就是限制最大的超时时间是 5s 和 最大的连接请求是 997 个。

Server

服务器标头包含有关原始服务器用来处理请求的软件的信息。

应该避免使用过于冗长和详细的 Server 值，因为它们可能会泄露内部实施细节，这可能会使攻击者容易地发现并利用已知的安全漏洞。例如下面这种写法

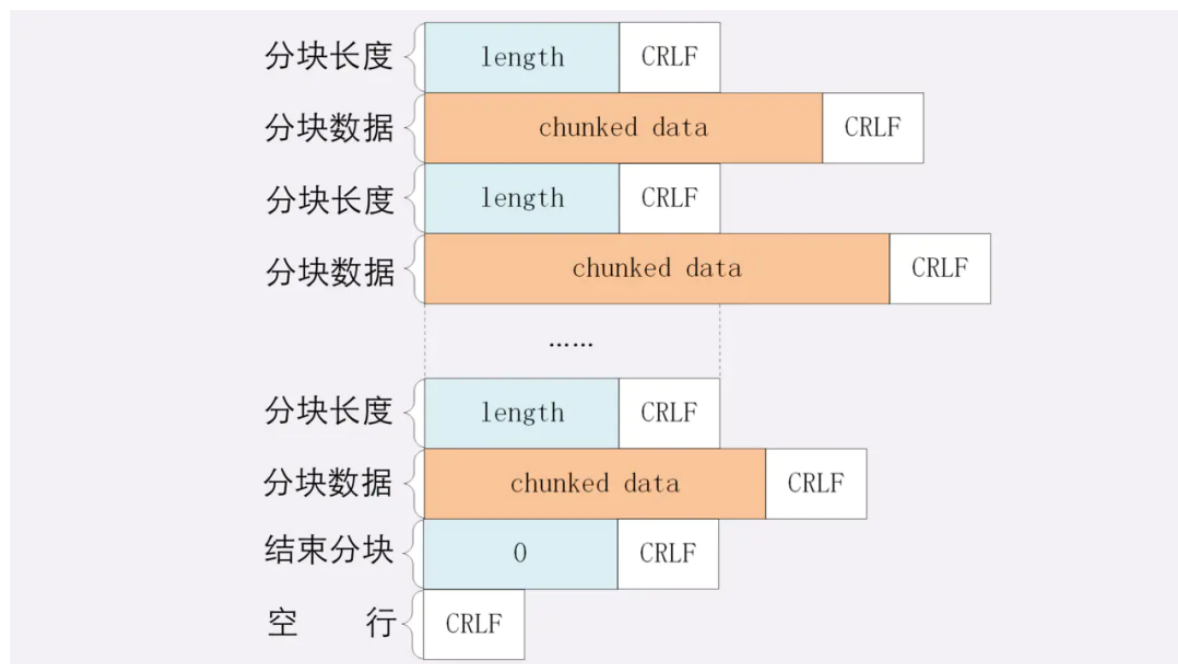
```
Server: Apache/2.4.1 (Unix)
```


Set-Cookie

Transfer-Encoding

首部字段 Transfer-Encoding 规定了传输报文主体时采用的编码方式。

分块传输，就是将传输的文件分解成多个小块，然后分发给浏览器，浏览器收到后再重新组装复原。



每个分块包含两个部分，分块长度和分块数据（长度头和数据块），长度头以CRLF结尾的一行明文，数据块紧跟在长度头后面，也是用CRLF结尾，最后用一个长度为0的块表示结束。

在响应报文里用头字段Transfer-Encoding:chunked表示报文里的body部分不是一次性发送过来的，而是分成了许多块逐个发送的。

在Transfer-Encoding: chunked和Content-Length中，这两个字段是互斥的。

```
Transfer-Encoding: chunked
```

HTTP /1.1 的传输编码方式仅对分块传输编码有效。

X-Frame-Options

HTTP 首部字段是可以自行扩展的。所以在 Web 服务器和浏览器的应用上，会出现各种非标准的首部字段。

首部字段 **X-Frame-Options** 属于 HTTP 响应首部，用于控制网站内容在其他 Web 网站的 Frame 标签内的显示问题。其主要目的是为了防止点击劫持（clickjacking）攻击。

下面是一个响应头的汇总，基于 HTTP 1.1

首部字段名	说明
Accept-Ranges	是否接受字节范围请求
Age	推算资源创建经过时间
ETag	资源的匹配信息
Location	令客户端重定向至指定URI
Proxy-Authenticate	代理服务器对客户端的认证信息
Retry-After	对再次发起请求的时机要求
Server	HTTP服务器的安装信息
Vary	代理服务器缓存的管理信息
WWW-Authenticate	服务器对客户端的认证信息

非 HTTP/1.1 首部字段

在 HTTP 协议通信交互中使用到的首部字段，不限于 RFC2616 中定义的 47 种首部字段。还有 Cookie、Set-Cookie 和 Content-Disposition 等在其他 RFC 中定义的首部字段，它们的使用频率也很高。这些非正式的首部字段统一归纳在 RFC4229 HTTP Header Field Registrations 中。

End-to-end 首部和 Hop-by-hop 首部

HTTP 首部字段将定义成缓存代理和非缓存代理的行为，分成 2 种类型。

一种是 **End-to-end** 首部和 **Hop-by-hop** 首部

End-to-end（端到端）首部

这些标头必须发送给消息的最终接收者：请求的服务器，或响应的客户端。中间代理必须重新传输未经修改的标头，并且缓存必须存储这些信息

Hop-by-hop（逐跳）首部

分在此类别中的首部只对单次转发有效，会因通过缓存或代理而不再转发。

下面列举了 HTTP/1.1 中的逐跳首部字段。除这 8 个首部字段之外，其他所有字段都属于端到端首部。

Connection、Keep-Alive、Proxy-Authenticate、Proxy-Authorization、Trailer、TE、Transfer-Encoding、Upgrade

✓ HTTP 的优点和缺点

HTTP 的优点

简单灵活易扩展

HTTP 最重要也是最突出的优点是 **简单、灵活、易于扩展**。

HTTP 的协议比较简单，它的主要组成就是 **header + body**，头部信息也是简单的文本格式，而且 HTTP 的请求报文根据英文也能猜出来个大概的意思，降低学习门槛，能够让更多的人研究和开发 HTTP 应用。

所以，在简单的基础上，HTTP 协议又多了 **灵活** 和 **易扩展** 的优点。

HTTP 协议里的请求方法、URI、状态码、原因短语、头字段等每一个核心组成要素都没有被制定死，允许开发者任意定制、扩充或解释，给予了浏览器和服务器的最大程度的信任和自由。

应用广泛、环境成熟

因为过于简单，普及，因此应用很广泛。因为 HTTP 协议本身不属于一种语言，它并不限定某种编程语言或者操作系统，所以天然具有**跨语言、跨平台**的优越性。而且，因为本身的简单特性很容易实现，所以几乎所有的编程语言都有 HTTP 调用库和外围的开发测试工具。

随着移动互联网的发展，HTTP 的触角已经延伸到了世界的每一个角落，从简单的 Web 页面到复杂的 JSON、XML 数据，从台式机上的浏览器到手机上的各种 APP、新闻、论坛、购物、手机游戏，你很难找到一个没有使用 HTTP 的地方。

无状态

无状态其实既是优点又是缺点。因为服务器没有记忆能力，所以就不需要额外的资源来记录状态信息，不仅实现上会简单一些，而且还能减轻服务器的负担，能够把更多的 CPU 和内存用来对外提供服务。

HTTP 的缺点

无状态

既然服务器没有记忆能力，它就无法支持需要连续多个步骤的 **事务** 操作。每次都得问一遍身份信息，不仅麻烦，而且还增加了不必要的数据传输量。由此出现了 **Cookie** 技术。

明文

HTTP 协议里还有一把优缺点一体的双刃剑，就是**明文传输**。明文意思就是协议里的报文（准确地说是 header 部分）不使用二进制数据，而是用简单可阅读的文本形式。

对比 TCP、UDP 这样的二进制协议，它的优点显而易见，不需要借助任何外部工具，用浏览器、Wireshark 或者 tcpdump 抓包后，直接用肉眼就可以很容易地查看或者修改，为我们的开发调试工作带来极大的便利。

当然缺点也是显而易见的，就是 **不安全**，可以被监听和被窥探。因为无法判断通信双方的身份，不能判断报文是否被更改过。