16-06 HTTP协议分析

腾讯课堂 零声学院 音视频高级开发教程

进一步参考文档：《HTTP协议》《图解HTTP》

报文结构

你也许对 TCP/UDP 的报文格式有所了解，拿 TCP 报文来举例，它在实际要传输的数据之前附加了一个 20 字节的头部数据，存储 TCP 协议必须的额外信息，例如发送方的端口号、接收方的端口号、包序号、标志位等等。
有了这个附加的 TCP 头，数据包才能够正确传输，到了目的地后把头部去掉，就可以拿到真正的数据。

HTTP 协议也是与 TCP/UDP 类似，同样也需要在实际传输的数据前附加一些头数据，不过与 TCP/UDP 不同的是，它是一个“**纯文本**”的协议，所以头数据都是 ASCII 码的文本，可以很容易地用肉眼阅读，不用借助程序解析也能够看懂。

HTTP 协议的请求报文和响应报文的结构基本相同，由三大部分组成：

- **起始行（start line）**：描述请求或响应的基本信息；
- **头部字段集合（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。

这其中前两部分起始行和头部字段经常又合称为“**请求头**”或“**响应头**”，消息正文又称为“**实体**”，但与“**header**”对应，很多时候就直接称为“**body**”。

HTTP 协议规定报文必须有 header，但可以没有 body，而且在 header 之后必须要有一个“空行”，也就是“CRLF”，十六进制的“0D0A”。
所以，一个完整的 HTTP 报文就像是下图的这个样子，注意在 header 和 body 之间有一个“空行”。

看一下我们之前用 Wireshark 抓的包吧。

```
GET / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image
/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
```

在这个浏览器发出的请求报文里。
第一行“GET / HTTP/1.1”就是请求行，而后面的“Host”“Connection”等等都属于 header，报文的最后是一个空白行结束，没有 body。

在很多时候，特别是浏览器发送 GET 请求的时候都是这样，HTTP 报文经常是只有 header 而没 body。

请求行

了解了 HTTP 报文的基本结构后，我们来看看请求报文里的起始行也就是**请求行**（request line），它简要地描述了**客户端想要如何操作服务器端的资源**。
请求行由三部分构成：

1. 请求方法：是一个动词，如 GET/POST，表示对资源的操作；
2. 请求目标：通常是一个 URI，标记了请求方法要操作的资源；
3. 版本号：表示报文使用的 HTTP 协议版本。

这三个部分通常使用空格（**space**）来分隔，最后要用 CRLF 换行表示结束。（图示SP代表空格）

还是用 Wireshark 抓包的数据来举例：

```
1 GET / HTTP/1.1\r\n
2 GET /live/livestream.m3u8 HTTP/1.1\r\n
```

在这个请求行里，“GET”是请求方法，“/”是请求目标，“HTTP/1.1”是版本号，把这三部分连起来，意思就是“服务器你好，我想获取网站根目录下的默认文件，我用的协议版本号是 1.1，请不要用 1.0 或者 2.0 回复我。”

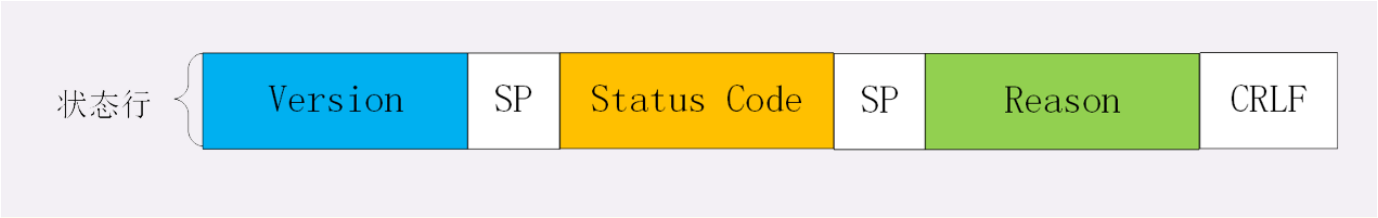
别看请求行就一行，貌似很简单，其实这里面的“讲究”是非常多的，尤其是前面的请求方法和请求目标，组合起来变化多端，后面我还会详细介绍。

状态行

看完了请求行，我们再看响应报文里的起始行，在这里它不叫“响应行”，而是叫“**状态行**”（status line），意思是**服务器响应的状态**。

比起请求行来说，状态行要简单一些，同样也是由三部分构成：

- 版本号：表示报文使用的 HTTP 协议版本；
- 状态码：一个三位数，用代码的形式表示处理的结果，比如 200 是成功，500 是服务器错误；
- 原因：作为数字状态码补充，是更详细的解释文字，帮助人理解原因。



看一下上一讲里 Wireshark 抓包里的响应报文，状态行是：

```
1 HTTP/1.1 200 OK\r\n
```

意思就是：“浏览器你好，我已经处理完了你的请求，这个报文使用的协议版本号是 1.1，状态码是 200，一切 OK。”
而另一个“GET /favicon.ico HTTP/1.1”的响应报文状态行是：

```
1 HTTP/1.1 404 Not Found
```

翻译成人话就是：“抱歉啊浏览器，刚才你的请求收到了，但我没找到你要的资源，错误代码是 404，接下来的事情你就看着办吧。”

头部字段

请求行或状态行再加上头部字段集合就构成了 HTTP 报文里完整的请求头或响应头，我画了两个示意图，你可以看一下。

请求头和响应头的结构是基本一样的，唯一的区别是起始行，所以我把请求头和响应头里的字段放在一起介绍。

头部字段是 **key-value** 的形式，key 和 value 之间用“:”分隔，最后用 CRLF 换行表示字段结束。比如在“Host: 127.0.0.1”这一行里 key 就是“Host”，value 就是“127.0.0.1”。

HTTP 头字段非常灵活，不仅可以使用标准里的 Host、Connection 等已有头，也可以任意添加自定义头，这就给 HTTP 协议带来了无限的扩展可能。

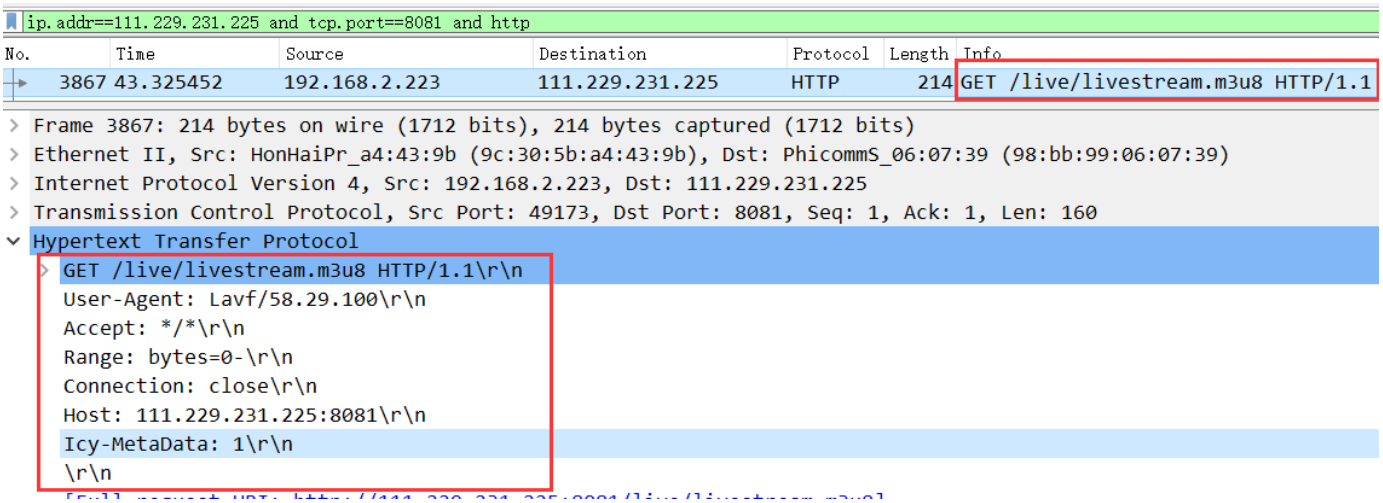
不过使用头字段需要注意下面几点：

- 字段名不区分大小写，例如“Host”也可以写成“host”，但首字母大写的可读性更好；
- 字段名里不允许出现空格，可以使用连字符“-”，但不能使用下划线“_”。例如，“test-name”是合法的字段名，而“test name”“test_name”是不正确的字段名；

- 字段名后面必须紧接着“:”，不能有空格，而“:”后的字段值前可以有多个空格；
- 字段的顺序是没有意义的，可以任意排列不影响语义；
- 字段原则上不能重复，除非这个字段本身的语义允许，例如 Set-Cookie。

用wireshark抓包的分析

请求



响应

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Connection: Keep-Alive\r\n
  Content-Length: 260\r\n
  Content-Type: application/vnd.apple.mpegurl\r\n
  Server: SRS/3.0.141(OuXuli)\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.028100000 seconds]
  [Request in frame: 3867]
  [Request URI: http://111.229.231.225:8081/live/livestream.m3u8]
  File Data: 260 bytes
Media Type
  Media type: application/vnd.apple.mpegurl (260 bytes)
```

```
0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK·
0010  0a 43 6f 6e 6e 65 63 74  69 6f 6e 3a 20 4b 65 65   ·Connect ion: Kee
0020  70 2d 41 6c 69 76 65 0d  0a 43 6f 6e 74 65 6e 74   p-Alive· ·Content
0030  2d 4c 65 6e 67 74 68 3a  20 32 36 30 0d 0a 43 6f   -Length:  260··Co
0040  6e 74 65 6e 74 2d 54 79  70 65 3a 20 61 70 70 6c   ntent-Ty pe: appl
0050  69 63 61 74 69 6f 6e 2f  76 6e 64 2e 61 70 70 6c   ication/ vnd.appl
0060  65 2e 6d 70 65 67 75 72  6c 0d 0a 53 65 72 76 65   e.mpegur l··Serve
0070  72 3a 20 53 52 53 2f 33  2e 30 2e 31 34 31 28 4f   r: SRS/3 .0.141(O
0080  75 58 75 6c 69 29 0d 0a  0d 0a 23 45 58 54 4d 33   uXuli)··  ··#EXTM3
0090  55 0a 23 45 58 54 2d 58  2d 56 45 52 53 49 4f 4e   U·#EXT-X -VERSION
00a0  3a 33 0a 23 45 58 54 2d  58 2d 4d 45 44 49 41 2d   :3·#EXT- X-MEDIA-
00b0  53 45 51 55 45 4e 43 45  3a 31 37 36 0a 23 45 58   SEQUENCE :176·#EX
00c0  54 2d 58 2d 54 41 52 47  45 54 44 55 52 41 54 49   T-X-TARG ETDURATI
00d0  4f 4e 3a 39 0a 23 45 58  54 49 4e 46 3a 35 2e 35   ON:9·#EX TINF:5.5
00e0  35 34 2c 20 6e 6f 20 64  65 73 63 0a 6c 69 76 65   54, no d esc·live
00f0  73 74 72 65 61 6d 2d 31  37 36 2e 74 73 0a 23 45   stream-1 76.ts·#E
0100  58 54 2d 58 2d 44 49 53  43 4f 4e 54 49 4e 55 49   XT-X-DIS CONTINUI
0110  54 59 0a 23 45 58 54 49  4e 46 3a 35 2e 34 36 31   TY·#EXTI NF:5.461
```

常用头字段

HTTP 协议规定了非常多的头部字段，实现各种各样的功能，但基本上可以分为四大类：

- 通用字段：在请求头和响应头里都可以出现；
- 请求字段：仅能出现在请求头里，进一步说明请求信息或者额外的附加条件；
- 响应字段：仅能出现在响应头里，补充说明响应报文的信息；
- 实体字段：它实际上属于通用字段，但专门描述 body 的额外信息。

对 HTTP 报文的解析和处理实际上主要就是对头字段的处理，理解了头字段也就理解了 HTTP 报文。

这里主要讲几个最基本的头，看完了它们你就应该能够读懂大多数 HTTP 报文了。

User-Agent

User-Agent是请求字段，只出现在请求头里。它使用一个字符串来描述发起 HTTP 请求的客户端，服务器可以依据它来返回最合适此浏览器显示的页面。
但由于历史的原因，User-Agent 非常混乱，每个浏览器都自称是“Mozilla”“Chrome”“Safari”，企图使用这个字段来互相“伪装”，导致 User-Agent 变得越来越长，最终变得毫无意义。
不过有的比较“诚实”的爬虫会在 User-Agent 里用“spider”标明自己是爬虫，所以可以利用这个字段实现简单的反爬虫策略。

Accept

Accept是请求字段，代表客户端希望接受的数据类型。比如Accept：text/xml（application/json）；
代表客户端希望接受的数据类型是xml（json ）类型；比如Accept: */*则说明客户端接收所有类型的数据。

Host

首先要说的是**Host**字段，它属于请求字段，只能出现在请求头里，它同时也是唯一一个 HTTP/1.1 规范里要求**必须出现**的字段，也就是说，如果请求头里没有 Host，那这就是一个错误的报文。
Host 字段告诉服务器这个请求应该由哪个主机来处理，当一台计算机上托管了多个虚拟主机的时候，服务器端就需要用 Host 字段来选择，有点像是一个简单的“路由重定向”。
例如我们的试验环境，在 127.0.0.1 上有三个虚拟主
机：“www.chrono.com”“www.metroid.net”和“origin.io”。那么当使用域名的方式访问时，就必须要用 Host 字段来区分这三个 IP 相同但域名不同的网站，否则服务器就会找不到合适的虚拟主机，无法处理。

Range

Range是**请求字段**。
比如Range: bytes=5001-10000 对于只需获资源的范围请求，包含首部字段 Range 即可告知服务器资源的指定范围。上面的示例表示请求获取从第 5001 字节到第 10000 字节的资源。

比如Range: bytes=0- 则是请求所有的数据。

接收到附带 Range 首部字段请求的服务器，会在处理请求之后返回状态码为 206 Partial Content 的响应。无法处理该范围请求时，则会返回状态码 200 OK 的响应及全部资源。

Connection

管理持久连接

①：`close` 断开连接

```
Connection: close
```

HTTP/1.1版本的默认连接都是持久连接。为此，客户端会在持久连接上连续发送请求。当服务器端想明确断开连接时，则指定`Connection` 首部字段的值为 `close`。

②：`Keep-Alive` 保持连接

```
Connection: keep-alive
```

HTTP/1.1 之前的版本的默认连接都是非持久连接。为此，如果想在旧版本的HTTP协议上维持持续连接，则需要指定 `Connection` 首部字段的值为 `keep-alive`。
在客户单发送请求给服务器时，携带此参数和值，服务器也会加上字段和值进行返回响应。

http是一个无状态的面向连接的协议。
http无状态：无状态协议是指http协议本身对于事务处理没有记忆功能，服务器不知道浏览器的状态。通俗的即使你登录了，去访问同一个网站的不同网页，服务器都不会知道你是谁，如果需要记录登录用户的信息，用户操作，用户行为等数据需要使用cookie或session来存储。

keep-alive：从HTTP/1.1起，**浏览器默认都开启了Keep-Alive**，保持连接特性，客户端和服务器都能选择随时关闭连接，则请求头中为connection:close。简单地说，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的TCP连接。但是Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。
误解：无状态不代表HTTP不能保持TCP连接，更不能代表HTTP使用的是UDP协议（无连接）。即使http在无状态下，只要客户端和服务器的头部信息connection:keep-alive，则在有效期内他们使用同一条TCP连接。

Date

Date字段是一个通用字段，但通常出现在响应头里，表示 HTTP 报文创建的时间，客户端可以使用这个时间再搭配其他字段决定缓存策略。

Server

Server字段是响应字段，只能出现在响应头里。它告诉客户端当前正在提供 Web 服务的软件名称和版本号， Server 字段也不是必须要出现的，因为这会把服务器的一部分信息暴露给外界，如果这个版本恰好存在 bug，那么黑客就有可能利用 bug 攻陷服务器。所以，有的网站响应头里要么没有这个字段，要么就给出一个完全无关的描述信息。
比如 GitHub，它的 Server 字段里就看不出是使用了 Apache 还是 Nginx，只是显示为“GitHub.com”。

再比如srs流媒体服务器的响应 `Server: SRS/3.0.141(OuXuli)\r\n`

Content-Type

Content-Type是**实体字段**，表发送端（客户端|服务器）发送的实体数据的数据类型。比如：Content-Type：text/html（application/json）；代表发送端发送的数据格式是html（json）。

Content-Length

实体字段里要说的一个是**Content-Length**，它表示报文里 body 的长度，也就是请求头或响应头空行后面数据的长度。服务器看到这个字段，就知道了后续有多少数据，可以直接接收。如果没有这个字段，那么 body 就是不定长的，需要使用 chunked 方式分段传输。