16-把大象装进冰箱：HTTP传输大文件的方法

上次我们谈到了HTTP报文里的body，知道了HTTP可以传输很多种类的数据，不仅是文本，也能传输图片、音频和视频。

早期互联网上传输的基本上都是只有几K大小的文本和小图片，现在的情况则大有不同。网页里包含的信息实在是太多了，随随便便一个主页HTML就有可能上百K，高质量的图片都以M论，更不要说那些电影、电视剧了，几G、几十G都有可能。

相比之下，100M的光纤固网或者4G移动网络在这些大文件的压力下都变成了“小水管”，无论是上传还是下载，都会把网络传输链路挤的“满满当当”。

所以，如何在有限的带宽下高效快捷地传输这些大文件就成了一个重要的课题。这就好比是已经打开了冰箱门（建立连接），该怎么把大象（文件）塞进去再关上门（完成传输）呢？

今天我们就一起看看HTTP协议里有哪些手段能解决这个问题。

数据压缩

还记得上一讲中说到的“数据类型与编码”吗？如果你还有印象的话，肯定能够想到一个最基本的解决方案，那就是“**数据压缩**”，把大象变成小猪佩奇，再放进冰箱。

通常浏览器在发送请求时都会带着“**Accept-Encoding**”头字段，里面是浏览器支持的压缩格式列表，例如gzip、deflate、br等，这样服务器就可以从中选择一种压缩算法，放进“**Content-Encoding**”响应头里，再把原数据压缩后发给浏览器。

如果压缩率能有50%，也就是说100K的数据能够压缩成50K的大小，那么就相当于在带宽不变的情况下网速提升了一倍，加速的效果是非常明显的。

不过这个解决方法也有个缺点，gzip等压缩算法通常只对文本文件有较好的压缩率，而图片、音频视频等多媒体数据本身就已经是高度压缩的，再用gzip处理也不会变小（甚至还有可能会增大一点），所以它就失效了。

不过数据压缩在处理文本的时候效果还是很好的，所以各大网站的服务器都会使用这个手段作为“保底”。例如，在Nginx里就会使用“gzip on”指令，启用对“text/html”的压缩。

分块传输

在数据压缩之外，还能有什么办法来解决大文件的问题呢？

压缩是把大文件整体变小，我们可以反过来思考，如果大文件整体不能变小，那就把它“拆开”，分解成多个小块，把这些小块分批发给浏览器，浏览器收到后再组装复原。

这样浏览器和服务器都不用在内存里保存文件的全部，每次只收发一小部分，网络也不会被大文件长时间占用，内存、带宽等资源也就节省下来了。

这种“**化整为零**”的思路在HTTP协议里就是“**chunked**”分块传输编码，在响应报文里用头字

段“**Transfer-Encoding: chunked**”来表示，意思是报文里的body部分不是一次性发过来的，而是分成了许多的块（chunk）逐个发送。

这就好比是用魔法把大象变成“乐高积木”，拆散了逐个装进冰箱，到达目的地后再施法拼起来“满血复活”。

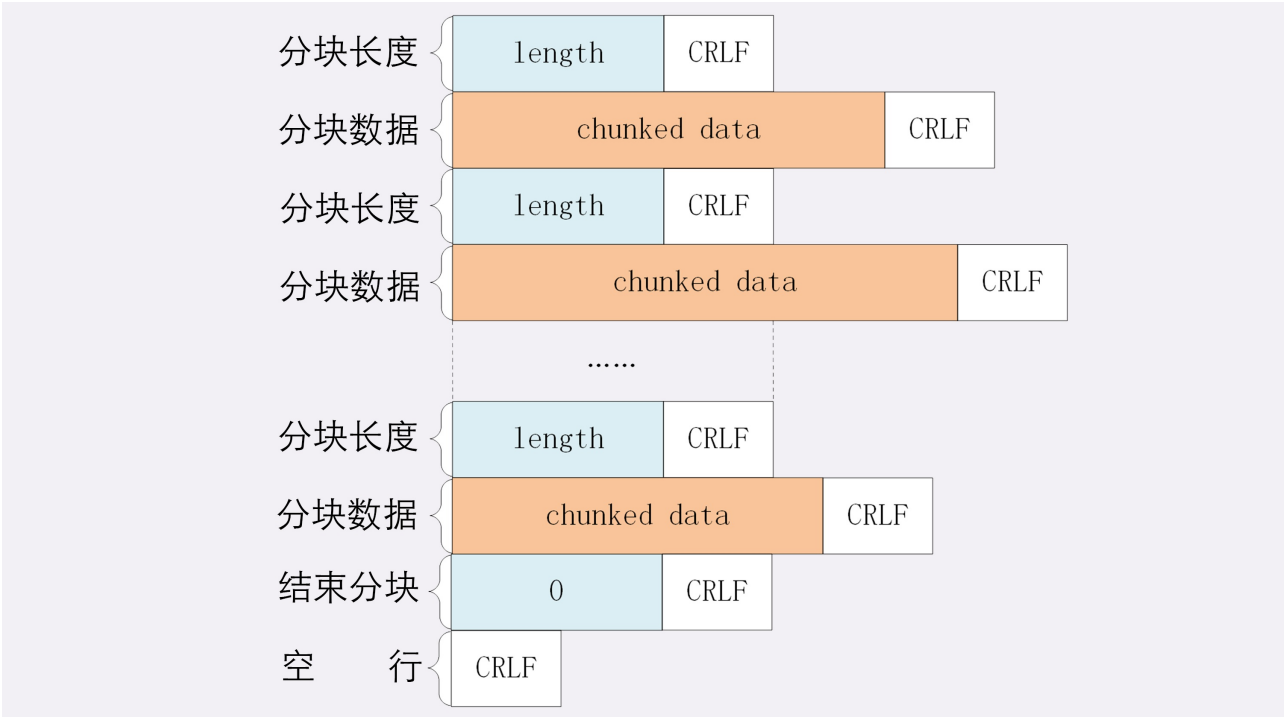
分块传输也可以用于“流式数据”，例如由数据库动态生成的表单页面，这种情况下body数据的长度是未知的，无法在头字段“**Content-Length**”里给出确切的长度，所以也只能用chunked方式分块发送。

“Transfer-Encoding: chunked”和“Content-Length”这两个字段是**互斥的**，也就是说响应报文里这两个字段不能同时出现，一个响应报文的传输要么是长度已知，要么是长度未知（chunked），这一点你一定要记住。

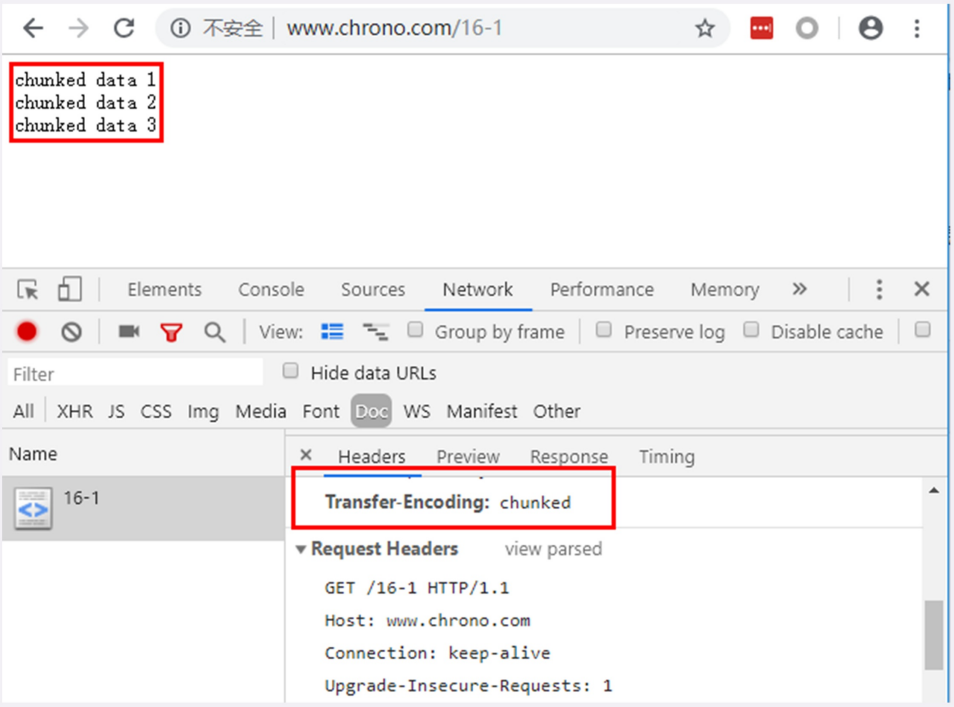
下面我们来看一下分块传输的编码规则，其实也很简单，同样采用了明文的方式，很类似响应头。

1. 每个分块包含两个部分，长度头和数据块；
2. 长度头是以CRLF（回车换行，即\r\n）结尾的一行明文，用16进制数字表示长度；
3. 数据块紧跟在长度头后，最后也用CRLF结尾，但数据不包含CRLF；
4. 最后用一个长度为0的块表示结束，即“0\r\n\r\n”。

听起来好像有点难懂，看一下图就好理解了：



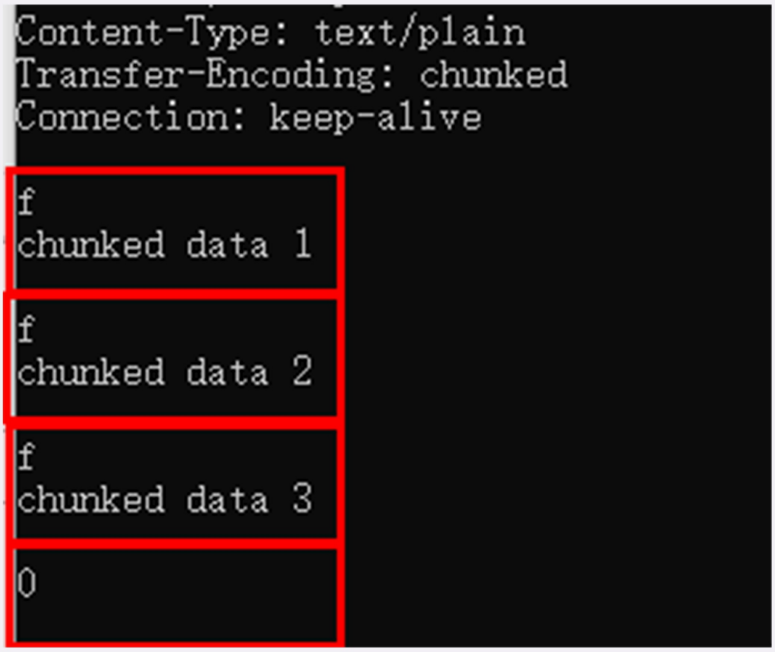
实验环境里的URI“/16-1”简单地模拟了分块传输，可以用Chrome访问这个地址看一下效果：



不过浏览器在收到分块传输的数据后会自动按照规则去掉分块编码，重新组装出内容，所以想要看到服务器发出的原始报文形态就得用Telnet手工发送请求（或者用Wireshark抓包）：

```
GET /16-1 HTTP/1.1
Host: www.chrono.com
```

因为Telnet只是收到响应报文就完事了，不会解析分块数据，所以可以很清楚地看到响应报文里的chunked数据格式：先是一行16进制长度，然后是数据，然后再是16进制长度和数据，如此重复，最后是0长度分块结束。



范围请求

有了分块传输编码，服务器就可以轻松地收发大文件了，但对于上G的超大文件，还有一些问题需要考虑。

比如，你在看当下正热播的某穿越剧，想跳过片头，直接看正片，或者有段剧情很无聊，想拖动进度条快进几分钟，这实际上是想获取一个大文件其中的片段数据，而分块传输并没有这个能力。

HTTP协议为了满足这样的需求，提出了“**范围请求**”（range requests）的概念，允许客户端在请求头里使用专用字段来表示只获取文件的一部分，相当于是**客户端的“化整为零”**。

范围请求不是Web服务器必备的功能，可以实现也可以不实现，所以服务器必须在响应头里使用字段“**Accept-Ranges: bytes**”明确告知客户端：“我是支持范围请求的”。

如果不支持的话该怎么办呢？服务器可以发送“Accept-Ranges: none”，或者干脆不发送“Accept-Ranges”字段，这样客户端就认为服务器没有实现范围请求功能，只能老老实实地收发整块文件了。

请求头**Range**是HTTP范围请求的专用字段，格式是“**bytes=x-y**”，其中的x和y是以字节为单位的数据范围。

要注意x、y表示的是“偏移量”，范围必须从0计数，例如前10个字节表示为“0-9”，第二个10字节表示为“10-19”，而“0-10”实际上是前11个字节。

Range的格式也很灵活，起点x和终点y可以省略，能够很方便地表示正数或者倒数的范围。假设文件是100个字节，那么：

- “0-”表示从文档起点到文档终点，相当于“0-99”，即整个文件；
- “10-”是从第10个字节开始到文档末尾，相当于“10-99”；
- “-1”是文档的最后一个字节，相当于“99-99”；
- “-10”是从文档末尾倒数10个字节，相当于“90-99”。

服务器收到Range字段后，需要做四件事。

第一，它必须检查范围是否合法，比如文件只有100个字节，但请求“200-300”，这就是范围越界了。服务器就会返回状态码**416**，意思是“你的范围请求有误，我无法处理，请再检查一下”。

第二，如果范围正确，服务器就可以根据Range头计算偏移量，读取文件的片段了，返回状态码“**206 Partial Content**”，和200的意思差不多，但表示body只是原数据的一部分。

第三，服务器要添加一个响应头字段**Content-Range**，告诉片段的实际偏移量和资源的总大小，格式是“**bytes x-y/length**”，与Range头区别在没有“=”，范围后多了总长度。例如，对于“0-10”的范围请求，值就是“bytes 0-10/100”。

最后剩下的就是发送数据了，直接把片段用TCP发给客户端，一个范围请求就算是处理完了。

你可以用实验环境的URI“/16-2”来测试范围请求，它处理的对象是“/mime/a.txt”。不过我们不能用Chrome浏览器，因为它没有编辑HTTP请求头的功能（这点上不如Firefox方便），所以还是要用Telnet。

例如下面的这个请求使用Range字段获取了文件的前32个字节：

```
GET /16-2 HTTP/1.1
Host: www.chrono.com
Range: bytes=0-31
```

返回的数据是（去掉了几个无关字段）：

```
HTTP/1.1 206 Partial Content
Content-Length: 32
Accept-Ranges: bytes
Content-Range: bytes 0-31/96

// this is a plain text json doc
```

有了范围请求之后，HTTP处理大文件就更加轻松了，看视频时可以根据时间点计算出文件的Range，不用下载整个文件，直接精确获取片段所在的数据内容。

不仅看视频的拖拽进度需要范围请求，常用的下载工具里的多段下载、断点续传也是基于它实现的，要点是：

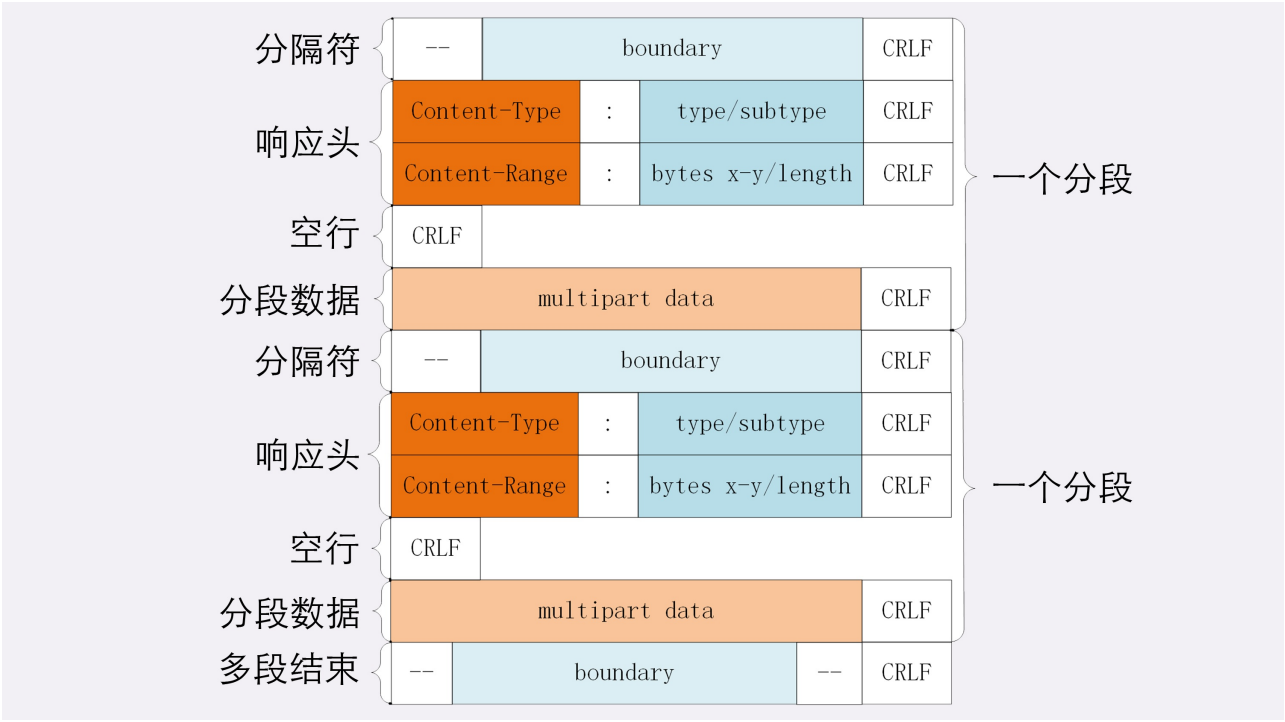
- 先发个HEAD，看服务器是否支持范围请求，同时获取文件的大小；
- 开N个线程，每个线程使用Range字段划分出各自负责下载的片段，发请求传输数据；
- 下载意外中断也不怕，不必重头再来一遍，只要根据上次的下载记录，用Range请求剩下的那一部分就可以了。

多段数据

刚才说的范围请求一次只获取一个片段，其实它还支持在Range头里使用多个“x-y”，一次性获取多个片段数据。

这种情况需要使用一种特殊的MIME类型：“**multipart/byteranges**”，表示报文的body是由多段字节序列组成的，并且还要用一个参数“**boundary=xxx**”给出段之间的分隔标记。

多段数据的格式与分块传输也比较类似，但它需要用分隔标记boundary来区分不同的片段，可以通过图来对比一下。



每一个分段必须以“- -boundary”开始（前面加两个“-”），之后要用“Content-Type”和“Content-Range”标记这段数据的类型和所在范围，然后就像普通的响应头一样以回车换行结束，再加上分段数据，最后用一个“- -boundary- -”（前后各有两个“-”）表示所有的分段结束。

例如，我们在实验环境里用Telnet发出有两个范围的请求：

```
GET /16-2 HTTP/1.1
Host: www.chrono.com
Range: bytes=0-9, 20-29
```

得到的就会是下面这样：

```
HTTP/1.1 206 Partial Content
Content-Type: multipart/byteranges; boundary=00000000001
Content-Length: 189
Connection: keep-alive
Accept-Ranges: bytes


--00000000001
Content-Type: text/plain
Content-Range: bytes 0-9/96

// this is
--00000000001
Content-Type: text/plain
Content-Range: bytes 20-29/96

ext json d
--00000000001--
```

报文里的“- -00000000001”就是多段的分隔符，使用它客户端就可以很容易地区分出多段Range 数据。

小结

今天我们学习了HTTP传输大文件相关的知识，在这里做一下简单小结：

1. 压缩HTML等文本文件是传输大文件最基本的方法；
2. 分块传输可以流式收发数据，节约内存和带宽，使用响应头字段“Transfer-Encoding: chunked”来表示，分块的格式是16进制长度头+数据块；
3. 范围请求可以只获取部分数据，即“分块请求”，实现视频拖拽或者断点续传，使用请求头字段“Range”和响应头字段“Content-Range”，响应状态码必须是206；
4. 也可以一次请求多个范围，这时候响应报文的数据类型是“multipart/byteranges”，body里的多个部分会用boundary字符串分隔。

要注意这四种方法不是互斥的，而是可以混合起来使用，例如压缩后再分块传输，或者分段后再分块，实验环境的URI“/16-3”就模拟了后一种的情形，你可以自己用Telnet试一下。

课下作业

1. 分块传输数据的时候，如果数据里含有回车换行（\r\n）是否会影响分块的处理呢？
2. 如果对一个被gzip的文件执行范围请求，比如“Range: bytes=10-19”，那么这个范围是应用于原文件还是压缩后的文件呢？

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 gzip 的压缩率通常能够超过 60%，而 br 算法是专为 HTML 设计的，压缩效率和性能比 gzip 还要好，能够再提高 20% 的压缩密度。

02 Nginx 的“gzip on”指令很智能，只会压缩文本数据，不会压缩图片、音频、视频。

03 Transfer-Encoding 字段最常见的值是 chunked，但也可以用 gzip、deflate 等，表示传输时使用了压缩编码。注意这与 Content - Encoding 不同，Transfer - Encoding 在传输后会被自动解码还原出原始数据，而 Content - Encoding 则必须由应用自行解码。

04 分块传输在末尾还允许有“拖尾数据”，由响应头字段 Trailer 指定。

05 与 Range 有关的还有一个 If-Range，即条件范围请求，将在第 20 讲介绍。

精选留言：

- Aaaaaaaaaayou 2019-07-03 08:47:58
  老师，有个问题：http交给tcp进行传输的时候本来就会分块，那http分块的意义是什么呢？ [5赞]

  作者回复2019-07-03 09:12:33
  在http层是看不到tcp的，它不知道下层协议是否会分块，下层是否分块对它来说没有意义，不关心。

  在http里一个报文必须是完整交付，在处理大文件的时候就很不方便，所以就要分块，在http层面方便处理。

  chunked主要是在http的层次来解决问题。

- chengzise 2019-07-03 11:26:53
  1. 分块传输中数据里含有回车换行（\r\n）不影响分块处理，因为分块前有数据长度说明
  2. 范围是应用于压缩后的文件 [2赞]

  作者回复2019-07-03 12:14:07
  1正确。

  2需要分情况，看原文件是什么形式。如果原来的文件是gzip的，那就正确。如果原文件是文本，而是在传输过程中被压缩，那么就应用于压缩前的数据。

  总之，range是针对原文件的。

- 白了少年头 2019-07-03 11:23:37
  1.数据里有回车换行，会影响分块的处理
  2.范围应用于压缩后文件
  不知道对不对，辛苦老师解答一下，谢谢！ [1赞]

  作者回复2019-07-03 12:15:07
  1不对，前面已经有同学回答了，因为分块包含了长度，所以回车换行不影响。

2需要分情况，看原文件是什么形式。如果原来的文件是gzip的，那就正确。如果原文件是文本，而是在传输过程中被压缩，那么就应用于压缩前的数据。

总之，range是针对原文件的。

- 啦啦啦 2019-07-03 00:24:50
  打卡打卡收获颇多 [1赞]

- Gopher 2019-07-03 13:58:31
  这个专栏质量很棒，老师很负责，知识讲解很通透，很容易就get、解惑了。

  哈哈哈，特此留言就是想说，老师，你认真做事的样子真帅(*•̀ᴗ•́*)و ̑̑

- -W.LI- 2019-07-03 13:12:52
  老师好!区分是请求字段,响应字段还是通用字段是查阅文档，文档上会有标注是么?。然后同一个header后面可以同时设置多个值用分割符风格就会一起生效。
  比如说Transfer-Encoding: chunked,gzip这个样子是么?有哪些字段支持组合啊?还是都支持

- Geek_54edc1 2019-07-03 12:04:05
  1、因为分块数据是明文传输，如果数据里有\r\n，是会影响分块处理的
  2、个人感觉应该是应用于原文件

  作者回复2019-07-03 12:35:15
  1不对，因为chunked格式里已经有长度了。

  2正确。

  看来有不少同学对第二个问题比较迷惑，我再说具体一点。

  比如说，有一个1M的纯文本，range请求其中的500K，然后服务器编码为gzip（Content-Encoding: gzip），压缩成200k，浏览器收到后解压缩，就得到了这部分的500k数据。

- 王小勃 2019-07-03 10:34:54
  讲的真好，简单清晰，赞

  作者回复2019-07-03 12:09:20
  thanks。

- 威~~微冷。。。 2019-07-03 10:25:52
  楼上的兄弟们多不用上班写bug的么？一个个的速度好快

- 一粟 2019-07-03 09:57:21
  迅雷下载或者在线视频播放器是不是在使用分块或者任意请求功能?

  作者回复2019-07-03 12:10:38
  只要是用http传输，就会用range，但他们也有可能用自己的协议而不是http，比如rmtp。

- -W.LI- 2019-07-03 09:34:17
  老师好!Transfer-Encoding: chunked表示分段传输，改成Transfer-Encoding: gzip以后会自动解压，分段传输的语意还在么

作者回复2019-07-03 12:15:45

看字段的值，没有chunked，那就不是分块，只是压缩。

- -W.LI- 2019-07-03 09:19:51

  老师好!在带宽固定的情况下，范围请求没发提高下载速度。如果服务器对客户端每个累链接限速的情况下，可通过多线程并发下载，提高下载速度是么?还有几个问题

  分块传输:顺序传一次一小块

  范围请求:支持跳跃式传输，还可以并发获取不同的range最后合并。

  多段数据:一次请求多个范围，范围可以不连续是么?如果必须联系的话和请求一个大范围没差别了。

  这几个拒的例子都是服务端这么返回的。

  客户端上传的时候怎么使用呢?老师后面会讲么。

  只读到了这么点，希望老师补充下每个的作用，和解决的问题，谢谢老师。

  作者回复2019-07-03 12:18:10

  1，是的，通过多线程并发下载，提高下载速度。

  2，范围可以不连续，例子里就是这样。

  3，客户端上传的时候也可以用chunked、gzip，但不能用range。

  注意这些字段的类型，只要是实体字段，那就在请求响应里都可以用。