Http不定长文件分片连续下载和定长文件断点下载

时间 2012-07-29 16:28:33 Berry的博客
原文 http://www.xiepingjin.com/?p=124
主题 HTTP

好吧，这个标题非常别扭，不过从服务器的角度来说，应用场景还是非常清晰的：前者是提供给用户的下载的内容是动态生成的，在用户触发下载的时候，服务器一边生成资源，一边同步提供下载，这样在用户体验上会好很多，用户不需要等待整个资源生成，立即能下载；后者是比较常见的，比如用户需要下载一部电影，在用户触发下载的时候，服务器端已知文件大小，如果用户使用迅雷等下载工具，资源还会被切成片，下载时不需要下载连续的片，所有分片最后会合并。
先从HTTP报文头来看这两种下载方式的不同吧，HTTP报文头的格式请参考http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html，建议全部看完，非常有价值：）
注意报文头的每一行以”\r\n”(CRLF)结尾，最后一行以”\n\n”结尾，然后才是报文内容。
(1)对于定长文件的断点下载：HTTP头中一般断点下载时才用到Range和Content-Range实体头，Range用户请求头中，指定第一个字节的位置和最后一个字节的位置，如（Range：200-300，Content-Range用于响应头，例如：
请求下载整个文件：
****
GET /berry.zip HTTP/1.1
Connection: close
Host: 111.1.1.1
Range: bytes=0-499
****
Range可以请求实体的一个或者多个子范围，Range的值为0表示第一个字节，即Range计算字节数是从0开始的。
表示头500个字节：bytes=0-499
表示第二个500字节：bytes=500-999

表示最后500个字节：bytes=-500
表示500字节以后的范围：bytes=500-
第一个和最后一个字节：bytes=0-0,-1
同时指定几个范围：bytes=500-600,601-999

一般正常回应
****
HTTP/1.1 206 OK

```
Content-Length: 801
Content-Type: application/octet-stream
Content-Location: http://www.onlinedown.net/hj_index.htm
Content-Range: bytes 0-100/2350 //2350:文件总大小
Last-Modified: Sun, 29 July 2012 16:10:12 GMT
Accept-Ranges: bytes
ETag: “d67a4bc5190c91:512″
Server: Microsoft-IIS/6.0
Date: Wed, 18 Feb 2009 07:55:26 GMT

abcdxxxx….
**********************************
```

注意：如果用户的请求中含有range ，则服务器的相应代码为206。
206 - Partial Content 客户发送了一个带有Range头的GET请求，服务器完成了它（HTTP 1.1新）。
再看一下标准格式文档：

```
The Content-Range entity-header is sent with a partial entity-body to specify where in the full entity-body the p

       Content-Range = "Content-Range" ":" content-range-spec
       content-range-spec      = byte-content-range-spec
       byte-content-range-spec = bytes-unit SP
                                 byte-range-resp-spec "/"
                                 ( instance-length | "*" )
       byte-range-resp-spec = (first-byte-pos "-" last-byte-pos)
                                    | "*"
       instance-length           = 1*DIGIT
The header SHOULD indicate the total length of the full entity-body, unless this length is unknown or difficult t

Unlike byte-ranges-specifier values (see section 14.35.1), a byte- range-resp-spec MUST only specify one range, a

A byte-content-range-spec with a byte-range-resp-spec whose last- byte-pos value is less than its first-byte-pos

A server sending a response with status code 416 (Requested range not satisfiable) SHOULD include a Content-Range

the selected resource. A response with status code 206 (Partial Content) MUST NOT include a Content-Range field wi

Examples of byte-content-range-spec values, assuming that the entity contains a total of 1234 bytes:

      . The first 500 bytes:
       bytes 0-499/1234
      . The second 500 bytes:
       bytes 500-999/1234
      . All except for the first 500 bytes:
       bytes 500-1233/1234
      . The last 500 bytes:
       bytes 734-1233/1234
When an HTTP message includes the content of a single range (for example, a response to a request for a single ra

       HTTP/1.1 206 Partial content
       Date: Wed, 15 Nov 1995 06:25:24 GMT
       Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
       Content-Range: bytes 21010-47021/47022
       Content-Length: 26012
       Content-Type: image/gif
When an HTTP message includes the content of multiple ranges (for example, a response to a request for multiple n

A response to a request for a single range MUST NOT be sent using the multipart/byteranges media type. A response

When a client requests multiple byte-ranges in one request, the server SHOULD return them in the order that they

If the server ignores a byte-range-spec because it is syntactically invalid, the server SHOULD treat the request a

If the server receives a request (other than one including an If- Range request-header field) with an unsatisfiabl

      Note: clients cannot depend on servers to send a 416 (Requested
      range not satisfiable) response instead of a 200 (OK) response for
```

```
an unsatisfiable Range request-header, since not all servers
implement this request-header.
```

(2)对于不定长文件的分片连续下载，使用chunked的方式，先看一下格式：

```
The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its

       Chunked-Body   = *chunk
                        last-chunk
                        trailer
                        CRLF
       chunk          = chunk-size [ chunk-extension ] CRLF
                        chunk-data CRLF
       chunk-size     = 1*HEX
       last-chunk     = 1*("0") [ chunk-extension ] CRLF
       chunk-extension= *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
       chunk-ext-name = token
       chunk-ext-val  = token | quoted-string
       chunk-data     = chunk-size(OCTET)
       trailer        = *(entity-header CRLF)
The chunk-size field is a string of hex digits indicating the size of the chunk. The chunked encoding is ended by

The trailer allows the sender to include additional HTTP header fields at the end of the message. The Trailer heac

A server using chunked transfer-coding in a response MUST NOT use the trailer for any header fields unless at leas

a)the request included a TE header field that indicates "trailers" is acceptable in the transfer-coding of the res

b)the server is the origin server for the response, the trailer fields consist entirely of optional metadata, and

This requirement prevents an interoperability failure when the message is being received by an HTTP/1.1 (or later)

An example process for decoding a Chunked-Body is presented in appendix 19.4.6.

All HTTP/1.1 applications MUST be able to receive and decode the "chunked" transfer-coding, and MUST ignore chunk-
```

请求包很简单，按照上面的格式，接下来用C++示例说一下服务器怎么回包吧：

```
string sHeader = "Content-Type: text/xml;\r\n";//下载一个xml文件
sHeader += "Content-Disposition: attachment; filename=\".1.xml\"\r\n";//最终文件名叫1.xml
sHeader += "Transfer-Encoding: chunked\n\n";//以分片方式连续下载，双\n结束报文头
cout<<sHeader;
cout.flush();//以tcp数据流形式输出，这个时候客户端已经准备好接收了
//假设要输出的内容是10句abcd，动态生成
char sLen[30] = {0};
for(unsigned i = 0; i <10; ++i)
{
    string sContent = "abcd";
    sContent += "\r\n";//CRLF
    snprintf(sLen, sizeof(sLen), "%x\r\n", sContent.size()-2);
    cout<<sLen;//先以16进制输出内容长度
    cout<<sContent;//输出内容
    cout.flush();
}
snprintf(sLen, sizeof(sLen), "0\r\n"); //最后，输出一个长度为0的内容，告诉客户端数据已经传输完毕。
cout<<sLen;
cout.flush();
```

这样一个分片连续下载就完成了，在这个过程中，客户端（如浏览器）的下载条会显示一个进度条，不停的从头到尾滚动，用户不知道什么时候能下载完，所以最好在下载窗口显示预估的下载时间：）
这种方式的下载非常适用实时生成实时下载，比如用户要下载mysql里面的100万条记录（已某种格式输出），这个时候服务器可以select count(*) where一下，得出总数，评估下载时间，然后select xxx where order by xx limit x,y，组织这y-x条记录输出，再继续select，这样效果非