Abstract

This document describes Chunked Progress, an extension to Transfer-Encoding: Chunked as defined in RFC2616 [RFC2616]. Chunked Progress introduces a backwards-compatible, RFC2616 compliant method to notify the client of transfer advancement in situations where the server has knowledge of progress but cannot know the resource size ahead of time.

1. Overview

User agents have been displaying progress as feedback to the user for some time already. As the nature of applications using HTTP as evolved to an increasingly dynamic nature, having the User Agent estimate progress based on bytes received is getting less useful, as data has to be generated a priori as a whole before being sent over the wire together with Content-Size, or Content-Size has to be omitted altogether, even when the server could actually estimate progress on the fly. In fact, Content-Size is used as a proxy for actual progress towards completion.
While informing the client of the requested resource content size is useful, it is useless when the exact resource size in bytes is not known ahead of time. Nonetheless, the server may have knowledge of the progress expressable as a unitless number towards completion.

It is important to distinguish the two notions of progress and completion. Completion is handled by having received Content-Size bytes, or with chunked content encoding without a Content-Size header, when a zero-length chunk is received. Content-size is a measure of completion, from which progress can be derived in increasingly limited cases.

Even then, it can be an awkward proxy: imagine a scenario where a known number of files of various sizes will have their SHA1 value computed and sent over the wire. The content size is known beforehand by a simple operation, yet progress is best measured by the number of bytes read, not the number of bytes sent.

Thankfully, Transfer-Encoding: chunked amounts to a form of multiplexing, where metadata can be sent interleaved with data on the same channel. By providing complementary information establishing progress towards completion, this extension aims to reduce latency and resource usage while increasing feedback in a backwards compatible way. Real world typical scenarios unable to generate progress even though it can be known on the server side include:

- gzipping files on the fly
- generating data from result rows of a database request
- generating data by walking a tree of files

2. Progress Chunk Extension

2.1. Request Header

The client SHOULD send the Chunk-Extensions: progress header in its request if it supports the feature. If the client sent the Chunk-Extensions: progress header, the server SHOULD include the progress chunk extension in the response chunks, compliant with either basic or extended mode of operation. Otherwise, the server MAY include the progress chunk extension, but MUST comply with the basic mode of operation.

2.2 Basic mode of operation

Since all HTTP/1.1 applications MUST be able to receive and decode the "chunked" transfer-coding, and MUST ignore chunk-extension extensions they do not understand, the server MAY reliably include the extension in chunks regardless of actual client-side support.

The chunk extension name MUST be the string "progress", while the chunk extension value MUST be a short "floating point" number comprised between 0 and 1, representing absolute progress towards completion. The special, negative value "-1" SHOULD be used when server progress tracking has been lost or compromised, and means progress status is "undefined".

```
chunk-ext-name = "progress"
chunk-ext-val = "-1"
| ( "0" [ ",\n 0*3DIGIT ] )
| ( "1" [ ",\n 0*3("0") ] )
```
The progress chunk extension MAY be omitted. If no progress chunk extension is present, the client MUST assume the current progress chunk extension value to be equal to the previous progress chunk extension value. If no previous chunk extension has been encountered yet, the chunk extension value MUST be assumed to be "undefined".

The client MUST NOT assume the progress value to be monotonically increasing, as the server MAY send any value it deems significant, including "undefined". Nonetheless, the client MAY implement logic presenting this information as monotonically increasing.

The client MUST NOT assume the value 1 to mean completion, due to possible rounding errors and insufficient precision.

If the client did not send the appropriate Request Header, the server MUST NOT send zero-length chunks unless all data has been sent, this to ensure backwards compatibility. To comply with Transfer-Encoding: chunked, the client, having not sent the request header, MUST accept a zero-length chunk as an end of data, whether or not this chunk has progress extension.

2.3 Enhanced mode of operation

If the client sent the appropriate Request Header, the server MAY send zero-length chunks with progress information. In such a case, the client MUST NOT assume that all data has been sent as is the case with naked transfer encoding chunked, and MUST wait for a zero-length chunk without a progress extension to assume completion. Indeed in most cases, the server MAY skip sending the last progress chunk and end the data stream with this last zero-length bare chunk instead of sending two consecutive zero-length chunks. Nonetheless, the server MAY, out of courtesy, send both, one notifying progress having reached 1 and one marking completion of chunked data transfer.

3. Notes

The mechanism by which this extension sends additional metadata via chunked extensions, even when there is no actual data to be sent, effectively implements a much more general multiplexing extension, which other chunked extensions may use. This area needs particular scrutiny, as it alters the Transfer-Encoding: chunked mode of operation. Maybe this should best be extracted in a newly defined Transfer-Encoding, entirely distinct from chunked.

Suggestions of improvements are welcome to increase compatibility with proxies, especially in the enhanced mode of operation.

It is regrettable that much state handling has to be done in order to implement those modes of operations, but it is necessary to extend current functionality all the while keeping backwards compatibility.

Clients and servers may restrict themselves to implement only basic mode of operation, greatly simplifying the required work but limiting potential functionality in the most dynamic cases.

Author’s Addresses

Loic Nageleisen
Email: loic.nageleisen@gmail.com