Ruby master - Bug #16672

net/http leaves original content-length header intact after inflating response

03/04/2020 09:22 PM - jmreid (Justin Reid)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.6.5p114 (2019-10-01 revision 67812) [x86_64-darwin19]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

When using net/http to make a request to a resource, the default request headers are the following (when you have ZLIB available): "accept-encoding"=>["gzip;q=1.0,deflate;q=0.6,identity;q=0.3"], "accept"=>["*/*"], "user-agent"=>["Ruby"]

This means that a resource will return a gzipped response if it can provide it. Take this URL for example: https://storage.googleapis.com/justin-reid-test/test.js

This is a JS file that has a content-length of 2733 when gzipped and 9995 when inflated:

```
curl "https://storage.googleapis.com/justin-reid-test/test.js" -H "accept-encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3" | wc -c
2733

curl "https://storage.googleapis.com/justin-reid-test/test.js" | wc -c
9995
```

When making a simple request for this asset using net/http:

```
uri = URI('https://storage.googleapis.com/justin-reid-test/test.js')
res = Net::HTTP.get_response(uri)
```

Ruby will (https://github.com/ruby/ruby/blob/f08cd708b11dd5b293986b92bb5e227731665b36/lib/net/http/response.rb#L264-L278):

- Delete the content-encoding header
- inflate the body
- return the inflated body

The issue here is that Ruby also leaves the content-length header set to the original request's value:

```
require 'net/http'

uri = URI('https://storage.googleapis.com/justin-reid-test/test.js')
res = Net::HTTP.get_response(uri)

puts "Fetching: https://storage.googleapis.com/justin-reid-test/test.js"
puts "Body size using String#bytesize: #{res.body.to_s.bytesize}"
puts "Content-Length response header: #{res.content_length}"
```

Results in:

```
Fetching: https://storage.googleapis.com/justin-reid-test/test.js
Body size using String#bytesize: 9995
Content-Length response header: 2733
```

This means that an incorrect content-length header is passed back when net/http makes requests for gzip objects and inflates them.

This issue was noticed when Rack changed their behaviour in how they compute content-length. They used to compute the content-length for each body, but that changed in 2.0.8:
https://github.com/rack/rack/commit/8c62821f4a464858a6b6ca3c3966ec308d2bb53e#diff-10b933d2c1fdc82ceecade456c64e1c2L92
https://github.com/rack/rack/issues/1472#issuecomment-574362342

Using Rack::ContentLength is now the method they prefer if you need to compute the content-length. However, Rack::ContentLength

will not try to re-compute the value if that header already exists:
https://github.com/rack/rack/blob/6196377654b7ff7ce7abaecea62bb285d77d53aa/lib/rack/content_length.rb#L21

Should Ruby:

- Do a self.delete 'content-length' in the inflater?
- Compute the content-length itself and update the header? (Hacky example: https://github.com/ruby/ruby/compare/master...jmreid:content-length)

History

#1 - 03/04/2020 10:05 PM - jmreid (Justin Reid)

- Description updated

#2 - 03/04/2020 10:25 PM - ioquatix (Samuel Williams)

Are you using res.content_length? Why can't you just use res.body.to_s.bytesize?

#3 - 03/05/2020 01:40 AM - jmreid (Justin Reid)

ioquatix (Samuel Williams) wrote in #note-2:

> Are you using res.content_length? Why can't you just use res.body.to_s.bytesize?

res.content_length is just the content-length response header from the net/http request:

```
uri = URI('https://storage.googleapis.com/justin-reid-test/test.js')
res = Net::HTTP.get_response(uri)

res.to_hash
```

```
{"x-guploader-uploadid"=>["AEnB2UrJSkEHDPO5iCRug2Qp0bzweRd8pd05d5eRq0fIUtRVZuibaGfQZhLHBN58g0ZW1N-qMcsUiZACutp
DoObHTijYs3_DrUNOy8SH9HA1hhTW0RtIbco"],
 "date"=>["Thu, 05 Mar 2020 01:35:08 GMT"],
 "expires"=>["Fri, 05 Mar 2021 01:35:08 GMT"],
 "last-modified"=>["Wed, 04 Mar 2020 20:53:40 GMT"],
 "etag"=>["\"e5994ce974ae1b8e426810037812e7d5\""],
 "x-goog-generation"=>["1583355220705723"],
 "x-goog-metageneration"=>["2"],
 "x-goog-stored-content-encoding"=>["gzip"],
 "x-goog-stored-content-length"=>["2733"],
 "content-type"=>["application/javascript; charset=utf-8"],
 "x-goog-hash"=>["crc32c=VCx7Dg==", "md5=5ZlM6XSuG45CaBADeBLn1Q=="],
 "x-goog-storage-class"=>["STANDARD"],
 "accept-ranges"=>["bytes"],
 "vary"=>["Accept-Encoding"],
 "content-length"=>["2733"],
 "server"=>["UploadServer"],
 "cache-control"=>["public, max-age=31536000, immutable"],
 "age"=>["6"],
 "alt-svc"=>["quic=\":443\"; ma=2592000; v=\"46,43\",h3-Q050=\":443\"; ma=2592000,h3-Q049=\":443\"; ma=2592000
,h3-Q048=\":443\"; ma=2592000,h3-Q046=\":443\"; ma=2592000,h3-Q043=\":443\"; ma=2592000"]}
```

The issue here is that this "content-length"=>["2733"] value is 2733. The res.body at this point is 9995, so content-length needs to match that or not exist. Otherwise it causes browsers to only partially download the file.

#4 - 03/05/2020 02:19 AM - jeremyevans0 (Jeremy Evans)

jmreid (Justin Reid) wrote in #note-3:

> The issue here is that this "content-length"=>["2733"] value is 2733. The res.body at this point is 9995, so content-length needs to match that or not exist. Otherwise it causes browsers to only partially download the file.

I don't think I agree with that logic. content_length is documented to return the value of the header, not the size of the decompressed body. So the method appears to be operating exactly as documented.

I'm also not sure why you are mentioning browsers in this context. If I open up Chrome and go into Development Tools, when I request https://storage.googleapis.com/justin-reid-test/test.js and look at the response headers, I see 2733 for Content-Length, not 9995.

The only possible interpretation I can think of where browsers come into play is if you were writing a server, using net/http inside a request handler, and taking the response status, headers, and body returned by net/http and returning them directly as the response. In which case the proper fix

would be not decompressing the body automatically:

```
res = Net::HTTP.get_response(uri){|res| res.decode_content = false}
```

#5 - 03/05/2020 02:56 AM - jmreid (Justin Reid)

> So the method appears to be operating exactly as documented.

I'm not saying that method isn't working as intended. It's working as intended and I'm just using it to show the size of the header for the request net/http made. My comment saying that content-length needs to match 9995 meant: The Content-Length header that net/http returns needs to actually match the content length of the body for that request.

> If I open up Chrome and go into Development Tools, when I request https://storage.googleapis.com/justin-reid-test/test.js and look at the response headers, I see 2733 for Content-Length, not 9995.

This is because Chrome reports the size of the file over the network and not the decompressed size in dev tools. If you find the resource in the network panel and hover over the "size" column, you'll see that "resource size" is the uncompressed size.

It's best to use curl to test these URLs.

My core concern is that Ruby shouldn't be leaving an incorrect header value around after it mutates the response. I can't see a reason why leaving Content-Length: 2733 makes sense when the actual body is 9995.

> The only possible interpretation I can think of where browsers come into play is if you were writing a server, using net/http inside a request handler, and taking the response status, headers, and body returned by net/http and returning them directly as the response. In which case the proper fix would be not decompressing the body automatically.

There are valid reasons to let net/http inflate the body. Just disabling gzip inflation because Ruby passes incorrect values in a mutated response doesn't seem like a proper fix.

#6 - 03/05/2020 04:04 AM - jeremyevans0 (Jeremy Evans)

jmreid (Justin Reid) wrote in #note-5:

> > So the method appears to be operating exactly as documented.
>
> I'm not saying that method isn't working as intended. It's working as intended and I'm just using it to show the size of the header for the request net/http made. My comment saying that content-length needs to match 9995 meant: The Content-Length header that net/http returns needs to actually match the content length of the body for that request.

I think I understand your reasoning better now. net/http is currently deleting the Content-Encoding header, so deleting or modifying the Content-Length header makes sense in that light. Modifying the Content-Length header is tricky because you do not know the decoded size until after decoding. Anyway, the current implementation, by deleting Content-Encoding and not changing Content-Length, is inconsistent.

I don't think net/http should be modifying the response headers. I think the response headers should remain exactly as sent by the server.

The deletion of the Content-Encoding header has been present since the initial support was merged in ff7f462bf49a1199b1657de6a73a0dc91deae1fa back in 2007. My guess as to why is so that existing callers that decoded bodies based on the value of the Content-Encoding header would not attempt to decode an already decoded body after the support was merged. And that does seem a reasonable way of keeping backwards compatibility while adding transparent decoding of bodies. I'm not sure how much code still exists that uses net/http and manually decodes bodies based on the Content-Encoding header, but maybe we can consider whether to remove the automatic deletion of the Content-Encoding header when decoding, possibly indicating whether decoding happened using a separate method.

#7 - 03/05/2020 04:42 AM - jmreid (Justin Reid)

jeremyevans0 (Jeremy Evans) wrote in #note-6:

> I don't think net/http should be modifying the response headers. I think the response headers should remain exactly as sent by the server.

I do agree in general. However, in this case net/http is actually mutating the response itself. To me that feels like a time when modifying the headers should happen as well to match those changes.

Deleting Content-Encoding makes sense because the encoding is no longer gzip. Same logic should be applied to Content-Length, in my opinion.

I do understand that net/http shouldn't be modifying headers for just any reason, but again, changing the actual response contents necessitates changing these two headers.

> maybe we can consider whether to remove the automatic deletion of the Content-Encoding header when decoding, possibly indicating whether decoding happened using a separate method.

This is definitely another option, but feels like it might set people up for more misunderstanding. If net/http inflates the body but leaves Content-Encoding: gzip, that opens the door for even more confusion in my opinion.

> Modifying the Content-Length header is tricky because you do not know the decoded size until after decoding

This is true, but Zlib::Inflate does have a total_out method that could be used (just a hacky patch I made while testing this locally): https://github.com/ruby/ruby/compare/master...jmreid:content-length

#8 - 03/05/2020 05:54 AM - jeremyevans0 (Jeremy Evans)

jmreid (Justin Reid) wrote in #note-7:

> > Modifying the Content-Length header is tricky because you do not know the decoded size until after decoding
>
> This is true, but Zlib::Inflate does have a total_out method that could be used (just a hacky patch I made while testing this locally): https://github.com/ruby/ruby/compare/master...jmreid:content-length

total_out doesn't give you the full size of the output until after the input is fully processed. So if there is an exception or other early exit from the block passed to inflater before the body is fully inflated, you can end up with an incorrect result. Also, the Content-Length header inside the block would still be wrong. You could remove it before the block and only set it on success after the block, that's probably the best way to handle it if you want to modify it.

```
s = Zlib::Deflate.deflate('a'* 100)
s.bytesize # => 12
io = Zlib::Inflate.new
io.total_out # => 0
io << s.byteslice(0,6)
io.total_out # => 2
io << s.byteslice(6,7)
io.total_out # => 100
```

#9 - 03/05/2020 01:59 PM - jmreid (Justin Reid)

jeremyevans0 (Jeremy Evans) wrote in #note-8:

> total_out doesn't give you the full size of the output until after the input is fully processed. So if there is an exception or other early exit from the block passed to inflater before the body is fully inflated, you can end up with an incorrect result. Also, the Content-Length header inside the block would still be wrong. You could remove it before the block and only set it on success after the block, that's probably the best way to handle it if you want to modify it.

Ah, understood!