Inflate all HTTP Content-Encoding: deflate, gzip, x-gzip responses by default

05/25/2012 07:37 AM - drbrain (Eric Hodel)

Status: Closed
Priority: Normal
Assignee: naruse (Yui NARUSE)
Target version: 2.0.0

Description

This patch moves the compression-handling code from Net::HTTP#get to Net::HTTPResponse to allow decompression to occur by default on any response body. (A future patch will set the Accept-Encoding on all requests that allow response bodies by default.)

Instead of having separate decompression code for deflate and gzip-encoded responses, ([Zlib::Inflate.new(32 + Zlib::MAX_WBITS)]) is used which automatically detects and inflated gzip-wrapped streams which allows for simpler processing of gzip bodies (no need to create a StringIO).

Related issues:

Blocks Ruby master - Feature #6494: Send Accept-Encoding for all HTTP requests

Associated revisions

Revision 36473 - 07/19/2012 10:43 PM - drbrain (Eric Hodel)

- lib/net/http/response.rb: Automatically inflate gzip and deflate-encoded response bodies. [Feature #6942]
- lib/net/http/generic_request.rb: Automatically accept gzip and deflate content-encoding for requests. [Feature #6494]
- lib/net/http/request.rb: Updated documentation for #6494.
- lib/net/http.rb: Updated documentation for #6492 and #6494, removed Content-Encoding handling now present in Net::HTTPResponse.
- test/net/http/test_httpresponse.rb: Tests for #6492
- test/net/http/test_http_request.rb: Tests for #6494
- test/open-uri/test_open-uri.rb (test_content_encoding): Updated test for automatic content-encoding handling.

Revision 36473 - 07/19/2012 10:43 PM - drbrain (Eric Hodel)

- lib/net/http/response.rb: Automatically inflate gzip and deflate-encoded response bodies. [Feature #6942]
- lib/net/http/generic_request.rb: Automatically accept gzip and deflate content-encoding for requests. [Feature #6494]
- lib/net/http/request.rb: Updated documentation for #6494.
- lib/net/http.rb: Updated documentation for #6492 and #6494, removed Content-Encoding handling now present in Net::HTTPResponse.
- test/net/http/test_httpresponse.rb: Tests for #6492
- test/net/http/test_http_request.rb: Tests for #6494
- test/open-uri/test_open-uri.rb (test_content_encoding): Updated test for automatic content-encoding handling.

Revision 36473 - 07/19/2012 10:43 PM - drbrain (Eric Hodel)

- lib/net/http/response.rb: Automatically inflate gzip and deflate-encoded response bodies. [Feature #6942]
- lib/net/http/generic_request.rb: Automatically accept gzip and deflate content-encoding for requests. [Feature #6494]
- lib/net/http/request.rb: Updated documentation for #6494.
- lib/net/http.rb: Updated documentation for #6492 and #6494, removed Content-Encoding handling now present in Net::HTTPResponse.
- test/net/http/test_httpresponse.rb: Tests for #6492
- test/net/http/test_http_request.rb: Tests for #6494
- test/open-uri/test_open-uri.rb (test_content_encoding): Updated test for automatic content-encoding handling.

Revision 36473 - 07/19/2012 10:43 PM - drbrain (Eric Hodel)

- lib/net/http/response.rb: Automatically inflate gzip and deflate-encoded response bodies. [Feature #6942]
- lib/net/http/generic_request.rb: Automatically accept gzip and deflate content-encoding for requests. [Feature #6494]
- lib/net/http/request.rb: Updated documentation for #6494.
- lib/net/http.rb: Updated documentation for #6492 and #6494, removed Content-Encoding handling now present in Net::HTTPResponse.
- test/net/http/test_httpresponse.rb: Tests for #6492
- test/net/http/test_http_request.rb: Tests for #6494
- test/open-uri/test_open-uri.rb (test_content_encoding): Updated test for automatic content-encoding handling.

Revision 36473 - 07/19/2012 10:43 PM - drbrain (Eric Hodel)
lib/net/http/response.rb: Automatically inflate gzip and deflate-encoded response bodies. [Feature #6942]
lib/net/http/generic_request.rb: Automatically accept gzip and deflate content-encoding for requests. [Feature #6494]
lib/net/http/request.rb: Updated documentation for #6494.
lib/net/http.rb: Updated documentation for #6492 and #6494, removed Content-Encoding handling now present in Net::HTTPResponse.
test/net/http/test_httpresponse.rb: Tests for #6492
test/net/http/test_http_request.rb: Tests for #6494
test/open-uri/test_open-uri.rb (test_content_encoding): Updated test for automatic content-encoding handling.

Revision 36473 - 07/19/2012 10:43 PM - drbrain (Eric Hodel)

lib/net/http/response.rb: Automatically inflate gzip and deflate-encoded response bodies. [Feature #6942]
lib/net/http/generic_request.rb: Automatically accept gzip and deflate content-encoding for requests. [Feature #6494]
lib/net/http/request.rb: Updated documentation for #6494.
lib/net/http.rb: Updated documentation for #6492 and #6494, removed Content-Encoding handling now present in Net::HTTPResponse.
test/net/http/test_httpresponse.rb: Tests for #6492
test/net/http/test_http_request.rb: Tests for #6494
test/open-uri/test_open-uri.rb (test_content_encoding): Updated test for automatic content-encoding handling.

Revision 36477 - 07/20/2012 02:05 AM - drbrain (Eric Hodel)

NEWS: Updated net/http for automatic proxy detection (#6546) and automatic gzip and deflate compression (#6492, #6494).

Revision 36477 - 07/20/2012 02:05 AM - drbrain (Eric Hodel)

NEWS: Updated net/http for automatic proxy detection (#6546) and automatic gzip and deflate compression (#6492, #6494).

Revision 36477 - 07/20/2012 02:05 AM - drbrain (Eric Hodel)

NEWS: Updated net/http for automatic proxy detection (#6546) and automatic gzip and deflate compression (#6492, #6494).

Revision 36477 - 07/20/2012 02:05 AM - drbrain (Eric Hodel)

NEWS: Updated net/http for automatic proxy detection (#6546) and automatic gzip and deflate compression (#6492, #6494).

Revision 36477 - 07/20/2012 02:05 AM - drbrain (Eric Hodel)

NEWS: Updated net/http for automatic proxy detection (#6546) and automatic gzip and deflate compression (#6492, #6494).

Revision 36477 - 07/20/2012 02:05 AM - drbrain (Eric Hodel)

NEWS: Updated net/http for automatic proxy detection (#6546) and automatic gzip and deflate compression (#6492, #6494).

History

#1 - 05/25/2012 09:15 AM - drbrain (Eric Hodel)
- File net.http.inflate_by_default.patch added

Opps, forgot patch.

#2 - 05/25/2012 10:25 AM - naruse (Yui NARUSE)
I agree with the concept of the patch, but

- ensure
  - @socket.finish if Inflater === @socket

When @socket is Socket-like object, the object should behave like a socket. Inflater#finish should be Inflater#shutdown or Inflater#close and this if-clause is not needed.

- def read_chunked(dest)
- def read_chunked(dest, inflater)
    len = nil
    total = 0
    while true

09/19/2021 2/11
len = hexlen.hex
break if len == 0
begin
  @socket.read len, dest
  inflater.read len, dest
ensure
  total += len
  @socket.read 2  # \n
  def
this variable inflater is confusing with the inflater method.
  
  def read clen, dest, ignore_eof = false
  
  temp_dest = ''

  @socket.read clen, temp_dest, ignore_eof

  dest << @inflate.inflate(temp_dest)
  
end

This read method return a string whose length is not clen, this is wrong.
Other IO-like object for example Zlib::GzipReader returns a string whose length is clen.
So Inflater should have a internal buffer and return the string whose length is just clen.

#3 - 05/25/2012 03:15 PM - drbrain (Eric Hodel)
naruse (Yui NARUSE) wrote:

  I agree with the concept of the patch, but
  
  • ensure
  • @socket.finish if Inflater === @socket

  When @socket is Socket-like object, the object should behave like a socket.
  Inflater#finish should be Inflater#shutdown or Inflater#close and this if-clause is not needed.

I considered this, but calling #close would also terminate a persistent connection which is undesirable.

I don't see a way to cleanly finish the inflate stream without an if-clause.

  [...]

I will submit a new patch that includes fixes for the rest of your comments.

#4 - 05/25/2012 03:38 PM - naruse (Yui NARUSE)
drbrain (Eric Hodel) wrote:

naruse (Yui NARUSE) wrote:

  I agree with the concept of the patch, but
  
  • ensure
  • @socket.finish if Inflater === @socket

  When @socket is Socket-like object, the object should behave like a socket.
  Inflater#finish should be Inflater#shutdown or Inflater#close and this if-clause is not needed.

I considered this, but calling #close would also terminate a persistent connection which is undesirable.
I don’t see a way to cleanly finish the inflate stream without an if-clause.

I see,

If Inflater's @socket.read returns nil or a string shorter than clen, it means the input is finished and @inflate can finish. So at that time, you can call @inflate.finish.

But on persistent connection current simple read all may eat another content, mustn’t it? I suspect they must see content body.

#5 - 05/26/2012 02:06 AM - drbrain (Eric Hodel)

naruse (Yui NARUSE) wrote:

If Inflater's @socket.read returns nil or a string shorter than clen, it means the input is finished and @inflate can finish. So at that time, you can call @inflate.finish.

I hadn't thought of that, I will implement it.

But on persistent connection current simple read all may eat another content, mustn’t it? I suspect they must see content body.

The response must contain Content-Length or Transfer-Encoding: chunked to be persistent, so this is OK. Net::HTTP already handles this.

#6 - 05/30/2012 08:45 AM - drbrain (Eric Hodel)

- File net.http.inflate_by_default.2.patch added

I've updated this patch. Upon working with the code again and looking at RFC 2616, I have made the following changes:

naruse (Yui NARUSE) wrote:

If Inflater's @socket.read returns nil or a string shorter than clen, it means the input is finished and @inflate can finish. So at that time, you can call @inflate.finish.

I hadn't thought of that, I will implement it.

Due to read_chunked, and persistent connections I don't see how to make this work.

When reading the body's Content-Length or Content-Range this strategy would work, but read_chunked reads multiple chunks of the compressed body and indicates the input to inflate is finished with a terminating "0\r\n\n" on the raw socket. Adding this communication between the raw socket and Inflater seems worse.

When the connection is persistent, #read should only return nil when the connection was abnormally terminated in which case we will throw away the body.

For #read_all, this would work.

Due to all the special cases, I changed Net::HTTPResponse#inflater to yield the Inflater and automatically clean it up. This keeps the special information about cleanup out of #read_body_0

this variable inflater is confusing with the inflater method.

In Net::HTTPResponse#read_chunked, the confusing "inflater" variable has been replaced with "chunk_data_io" which comes from RFC 2616 section 3.6.1.

This read method return a string whose length is not clen, this is wrong. Other IO-like object for example Zlib::GzipReader returns a string whose length is clen. So Inflater should have an internal buffer and return the string whose length is just clen.

Upon review, I think this is OK.

RFC 2616 specifies that Content-Length and Content-Range (which are used for clen) refer to the transferred bytes and are used to read the correct amount of data from the response to maintain the persistent connection. Net::HTTPResponse#read_body doesn't allow the user to specify the amount of bytes they wish to read, so returning more data to the user is OK.

I have made an additional change beyond your review:
I've added a Net::ReadAdapter to the Inflater to stream of the encoded response body through inflate without buffering it all. This will reduce memory consumption for large responses.

#7 - 05/31/2012 11:10 PM - mame (Yusuke Endoh)
- Status changed from Open to Assigned
- Assignee set to naruse (Yui NARUSE)

#8 - 06/07/2012 04:08 AM - naruse (Yui NARUSE)

drbrain (Eric Hodel) wrote:

I've updated this patch. Upon working with the code again and looking at RFC 2616, I have made the following changes:

naruse (Yui NARUSE) wrote:

If Inflater's @socket.read returns nil or a string shorter than clen, it means the input is finished and @inflate can finish. So at that time, you can call @inflate.finish.

I hadn't thought of that, I will implement it.

Due to read_chunked, and persistent connections I don't see how to make this work.

Yeah, I thought adding an another layer, transport encoding decoder, but it is just an idea and I don't suggest it.

this variable inflater is confusing with the inflater method.

In Net::HTTPResponse#read_chunked, the confusing "inflater" variable has been replaced with "chunk_data_io" which comes from RFC 2616 section 3.6.1.

This read method return a string whose length is not clen, this is wrong. Other IO-like object for example Zlib::GzipReader returns a string whose length is clen. So Inflater should have a internal buffer and return the string whose length is just clen.

Upon review, I think this is OK.

RFC 2616 specifies that Content-Length and Content-Range (which are used for clen) refer to the transferred bytes and are used to read the correct amount of data from the response to maintain the persistent connection. Net::HTTPResponse#read_body doesn't allow the user to specify the amount of bytes they wish to read, so returning more data to the user is OK.

Your patch hides content-encoding layer. Content-Length and Content-Range are the member of the layer.

Net::HTTPRequest#read is on the layer. A user of net/http can't know whether a request used content-encoding or not. On such situation, it can't be a reason why hidden Content-Encoding layer effects the behavior of read method.

I have made an additional change beyond your review:

I've added a Net::ReadAdapter to the Inflater to stream of the encoded response body through inflate without buffering it all. This will reduce memory consumption for large responses.

ok.

#9 - 06/08/2012 09:48 AM - drbrain (Eric Hodel)

naruse (Yui NARUSE) wrote:

drbrain (Eric Hodel) wrote:

Due to read_chunked, and persistent connections I don't see how to make this work.

Yeah, I thought adding an another layer, transport encoding decoder, but it is just an idea and I don't suggest it.

I had this idea too, but it would be a larger change. I hope we can create something simpler, but also usable.
This read method return a string whose length is not clen, this is wrong. Other IO-like object for example Zlib::GzipReader returns a string whose length is clen. So Inflater should have a internal buffer and return the string whose length is just clen.

Upon review, I think this is OK.

RFC 2616 specifies that Content-Length and Content-Range (which are used for clen) refer to the transferred bytes and are used to read the correct amount of data from the response to maintain the persistent connection. Net::HTTPResponse#read_body doesn't allow the user to specify the amount of bytes they wish to read, so returning more data to the user is OK.

Your patch hides content-encoding layer.
Content-Length and Content-Range are the member of the layer.

Net::HTTPRequest#read is on the layer.

I'm confused.

There is Net::BufferedIO#read, but no Net::HTTPResponse#read. There is Net::HTTPResponse#read_body which lets you read the entire body or chunks of unknown size.

I don't see a way for the user to say "read 10 bytes of the response body" without manually buffering:

```ruby
require 'net/http'
req = Net::HTTP::Get.new '/'
body_part = Net::HTTP.start 'localhost' do |http|
  buffer = ''
  target_size = 10
  http.request req do |res|
    res.read_body do |chunk|
      break if buffer.length == target_size
      buffer << chunk[0, target_size - buffer.length]
    end
  end
  buffer
end
p body_part
```

Since Net::HTTPResponse is not usable as an IO I don't think IO-like behavior should apply to Net::HTTPResponse::Inflater.

A user of net/http can't know whether a request used content-encoding or not.

Do you mean "a user of net/http must be able to tell content-encoding was present"?

When this patch is combined with #6494 they will not be able to know whether a request used content-encoding or not. I think this is good, the user should not have to worry about how the bytes were transported. (This behavior matches Net::HTTP#get).

If we want the user to be able to handle content-encoding themselves I think adding a Net::HTTP#compression = false (which will disable both #6492 and #6494) would be best. We can also add an indicator on Net::HTTPResponse that decompression was performed.

For Content-Length with Content-Encoding, the Content-Length will be invalid. I think this is OK because RFC 2616 doesn't contain an indicator of the decoded length and the user is most likely interested in the decoded body.

Content-Range with Content-Encoding requires special handling. The compressed stream may start anywhere in the underlying block. (For a deflate-based stream the user would need to manually reconstruct the complete response in order to inflate it.) I think such users should disable compression.

On such situation, it can't be a reason why hidden Content-Encoding layer effects the behavior of read method.

I agree that in RFC 2616 that Content-Encoding, Content-Length and Content-Range are all on the same layer, but without an IO-like interface for the Net::HTTPResponse body I don't think a restriction on the behavior of the read method should apply. Since this API is entirely internal, I think it is OK if a future addition to the API needs to add buffering to be IO-like.
drbrain (Eric Hodel) wrote:

naruse (Yui NARUSE) wrote:

Due to read_chunked, and persistent connections I don't see how to make this work.

Yeah, I thought adding another layer, transport encoding decoder, but it is just an idea and I don't suggest it.

I had this idea too, but it would be a larger change. I hope we can create something simpler, but also usable.

This read method return a string whose length is not clen, this is wrong.
Other IO-like object for example Zlib::GzipReader returns a string whose length is clen.
So Inflater should have a internal buffer and return the string whose length is just clen.

Upon review, I think this is OK.

RFC 2616 specifies that Content-Length and Content-Range (which are used for clen) refer to the transferred bytes and are used to read the correct amount of data from the response to maintain the persistent connection. Net::HTTPResponse#read_body doesn't allow the user to specify the amount of bytes they wish to read, so returning more data to the user is OK.

Your patch hides content-encoding layer.
Content-Length and Content-Range are the member of the layer.

Net::HTTPRequest#read is on the layer.

I'm confused.

(snip)
Since Net::HTTPResponse is not usable as an IO I don't think IO-like behavior should apply to Net::HTTPResponse::Inflater.

Ah, yes, you are correct, it is only for Net::HTTPResponse::Inflater.

A user of net/http can't know whether a request used content-encoding or not.

I am unsure what you mean by "can't".

Do you mean "a user of net/http must be able to tell content-encoding was present"?

Yes.

When this patch is combined with #6494 they will not be able to know whether a request used content-encoding or not. I think this is good, the user should not have to worry about how the bytes were transported. (This behavior matches Net::HTTP#get).

Yeah, I think it is acceptable.

If we want the user to be able to handle content-encoding themselves I think adding a Net::HTTP#compression = false (which will disable both #6492 and #6494) would be best. We can also add an indicator on Net::HTTPResponse that decompression was performed.

Someone may want such function, but it is not required until someone actually request.

For Content-Length with Content-Encoding, the Content-Length will be invalid. I think this is OK because RFC 2616 doesn't contain an indicator of the decoded length and the user is most likely interested in the decoded body.

It is OK for an HTTP client itself because Content-Length is for the reader of socket.
A client can know how many bytes should it read by Content-Length.
For this purpose, Content-Length must show the compressed size.

For Content-Range with Content-Encoding requires special handling. The compressed stream may start anywhere in the underlying block. (For a deflate-based stream the user would need to manually reconstruct the complete response in order to inflate it.) I think such users should disable compression.
So on range response with content-encoding, the response won't uncompress the body and keep Content-Encoding field, right?
If so, I agree.

On such situation, it can't be a reason why hidden Content-Encoding layer effects the behavior of read method.

I agree that in RFC 2616 that Content-Encoding, Content-Length and Content-Range are all on the same layer, but without an IO-like interface for the Net::HTTPResponse body I don't think a restriction on the behavior of the read method should apply. Since this API is entirely internal, I think it is OK if a future addition to the API needs to add buffering to be IO-like.

OK, I agree if the read method has a comment which explain it breaks the manner of IO-like object because it is internal API.

#11 - 06/09/2012 05:48 AM - drbrain (Eric Hodel)
naruse (Yui NARUSE) wrote:

drbrain (Eric Hodel) wrote:

naruse (Yui NARUSE) wrote:

A user of net/http can't know whether a request used content-encoding or not.

I am unsure what you mean by "can't".
Do you mean "a user of net/http must be able to tell content-encoding was present"?

Yes.
When this patch is combined with #6494 they will not be able to know whether a request used content-encoding or not. I think this is good, the user should not have to worry about how the bytes were transported. (This behavior matches Net::HTTP#get).

Yeah, I think it is acceptable.

Ok.

If we want the user to be able to handle content-encoding themselves I think adding a Net::HTTP#compression = false (which will disable both #6492 and #6494) would be best. We can also add an indicator on Net::HTTPResponse that decompression was performed.

Someone may want such function, but it is not required until someone actually request.

I will make a future patch, I need such a feature to handle broken deflate streams in mechanize.

Content-Range with Content-Encoding requires special handling. The compressed stream may start anywhere in the underlying block. (For a deflate-based stream the user would need to manually reconstruct the complete response in order to inflate it.) I think such users should disable compression.

So on range response with content-encoding, the response won't uncompress the body and keep Content-Encoding field, right?
If so, I agree.

Ok, I will update the patch

On such situation, it can't be a reason why hidden Content-Encoding layer effects the behavior of read method.

I agree that in RFC 2616 that Content-Encoding, Content-Length and Content-Range are all on the same layer, but without an IO-like interface for the Net::HTTPResponse body I don't think a restriction on the behavior of the read method should apply. Since this API is entirely internal, I think it is OK if a future addition to the API needs to add buffering to be IO-like.

OK, I agree if the read method has a comment which explain it breaks the manner of IO-like object because it is internal API.
Ok, I will update the patch

#12 - 06/09/2012 06:47 AM - drbrain (Eric Hodel)
- File net.http.inflate_by_default.3.patch added

This patch adds a comment to Net::HTTPResponse::Inflater#read discussing returning more bytes than clen and a link to this ticket.

This patch ignores Content-Encoding when Content-Range is present.

#13 - 06/09/2012 09:29 AM - normalperson (Eric Wong)
"drbrain (Eric Hodel)" drbrain@segment7.net wrote:

Feature #6492: Inflate all HTTP Content-Encoding: deflate, gzip, x-gzip responses by default
https://bugs.ruby-lang.org/issues/6492

I like Net::HTTP being able to inflate compressed responses.

However, I think doing this by default is exploitable by an evil server. A server could compress a huge file of zeroes to trigger an out-of-memory conditions in existing code that uses Net::HTTP.

100M compresses to 100K for me:

$ dd if=/dev/zero bs=1M count=100 | gzip -c | wc -c
101791

#14 - 06/09/2012 10:23 AM - drbrain (Eric Hodel)
On Jun 8, 2012, at 5:28 PM, Eric Wong normalperson@yhbt.net wrote:

I like Net::HTTP being able to inflate compressed responses.

However, I think doing this by default is exploitable by an evil server. A server could compress a huge file of zeroes to trigger an out-of-memory conditions in existing code that uses Net::HTTP.

Net::HTTP#get does this by default already, this patch (and #6494) make this the default for all requests.

If you aren't using the API to handle a compressed 100MB request (Net::HTTPResponse#read_body with a block) you probably can't handle an raw 100MB response, so what is the difference besides bandwidth cost of the server?

#15 - 06/09/2012 02:23 PM - normalperson (Eric Wong)
Eric Hodel drbrain@segment7.net wrote:

On Jun 8, 2012, at 5:28 PM, Eric Wong normalperson@yhbt.net wrote:

I like Net::HTTP being able to inflate compressed responses.

However, I think doing this by default is exploitable by an evil server. A server could compress a huge file of zeroes to trigger an out-of-memory conditions in existing code that uses Net::HTTP.

Net::HTTP#get does this by default already, this patch (and #6494) make this the default for all requests.

I've always considered Net::HTTP#get (or anything where slurping is done blindly) dangerous when talking to untrusted servers regardless of gzip.

If you aren't using the API to handle a compressed 100MB request (Net::HTTPResponse#read_body with a block) you probably can't handle an raw 100MB response, so what is the difference besides bandwidth cost of the server?

With your patch, I'm getting 16M chunks from read_body. Maybe on newer systems, 16M is "safe" to slurp in memory. I think it's too big, but I may also be a dinosaur :)

09/19/2021
Also, HTTP servers may blindly send Content-Encoding: gzip data regardless of whether the client requested with Accept-Encoding: gzip or not. I seem to recall reading of a major website that forces gzip on visitors regardless of their Accept-Encoding:

```
------------------------------ 8< -----------------------------
require 'uri'
require 'net/http'

# feel free to use this URL for testing
uri = URI('http://yhbt.net/x')

Net::HTTP.start(uri.host, uri.port) do |http|
  request = Net::HTTP::Get.new(uri.request_uri)
  request['Accept-Encoding'] = 'gzip'
  http.request request do |response|
    response.read_body do |chunk|
      p [chunk.bytesize]
    end
  end
end
------------------------------ 8< -----------------------------
```

I only used "gzip -9" to generate this test example, I'm not sure if there are ways to use zlib to compress even more aggressively.

Achieving bzip2-level compression ratios would be very scary :)

```
dd if=/dev/zero bs=1M count=1000 | bzip2 -c -9 | wc -c
=> 753
```

#16 - 06/15/2012 05:45 AM - naruse (Yui NARUSE)
How about adding a new attribute which limits the size of reading body?

#17 - 06/15/2012 08:23 AM - normalperson (Eric Wong)
"naruse (Yui NARUSE)" naruse@airemix.jp wrote:

  How about adding a new attribute which limits the size of reading body?

Maybe. I'm not familiar with zlib internals, but does it need to allocate that memory internally even if it's not returned to the callers?

Perhaps adding an attribute to toggle transparent inflate and documenting it with a warning about possible memory usage is the way to go.

#18 - 06/15/2012 10:26 AM - naruse (Yui NARUSE)
normalperson (Eric Wong) wrote:

  "naruse (Yui NARUSE)" naruse@airemix.jp wrote:

    How about adding a new attribute which limits the size of reading body?

Maybe. I'm not familiar with zlib internals, but does it need to allocate that memory internally even if it's not returned to the callers?

Yeah, currently you are correct.
zlib itself has such mechanism, but ext/zlib doesn't.
I'm inspecting deeper now.

#19 - 07/11/2012 05:29 AM - drbrain (Eric Hodel)
- File net.http.inflate_by_default.4.patch added

This patch uses streaming zlib from #6612 which limits the size of a chunk to 16KB (the maximum size of the zlib buffer) for streaming output without the potential for DOS due to large memory growth.

running the test from note 15 shows all 16KB chunk byte sizes.
This patch uses streaming zlib from #6612 which limits the size of a chunk to 16KB (the maximum size of the zlib buffer) for streaming output without the potential for DOS due to large memory growth.

running the test from note 15 shows all 16KB chunk byte sizes.

OK, commit it!

#21 - 07/20/2012 08:17 AM - drbrain (Eric Hodel)
- Status changed from Assigned to Closed

Committed in r36473

Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.http.inflate_by_default.patch</td>
<td>7.28 KB</td>
<td>05/25/2012</td>
<td>drbrain (Eric Hodel)</td>
</tr>
<tr>
<td>net.http.inflate_by_default.2.patch</td>
<td>8.32 KB</td>
<td>05/30/2012</td>
<td>drbrain (Eric Hodel)</td>
</tr>
<tr>
<td>net.http.inflate_by_default.3.patch</td>
<td>9.36 KB</td>
<td>06/09/2012</td>
<td>drbrain (Eric Hodel)</td>
</tr>
<tr>
<td>net.http.inflate_by_default.4.patch</td>
<td>9.42 KB</td>
<td>07/11/2012</td>
<td>drbrain (Eric Hodel)</td>
</tr>
</tbody>
</table>