Duplicated Encoded Image Data

hajimehoshi@chromium.org
Last Updated: 2016-07-11

Summary

This document aims to explain the CL https://codereview.chromium.org/2054643003/ to remove duplicated encoding image data. ImageResource has a blink::Image object in m_image and encoded image data in m_data as SharedBuffer. The problem is that there is also encoded image data in m_image's DeferredImageDecoder in SkRWBuffer. We found that this duplication consumes much memory. In theory, decoded data can be recreated from that in SkRWBuffer when needed. This CL tries to remove encoded image data in SharedBuffer by changing lifetimes of ImageResource's m_image and m_data.

What the CL does

- Changes the lifetime of ImageResource::m_data to be cleared soon after m_image is created.
- Changes the lifetime of ImageResource::m_image not to be discarded even when pruning.
- Changes ImageResource::resourceBuffer to return Image::data when m_data is not present.
- Overrides BitmapImage::data() to return its ImageSource::data().
- Adds DeferredImageDecoder::data() to return SharedBuffer generated from SkRWBuffer.

Objects

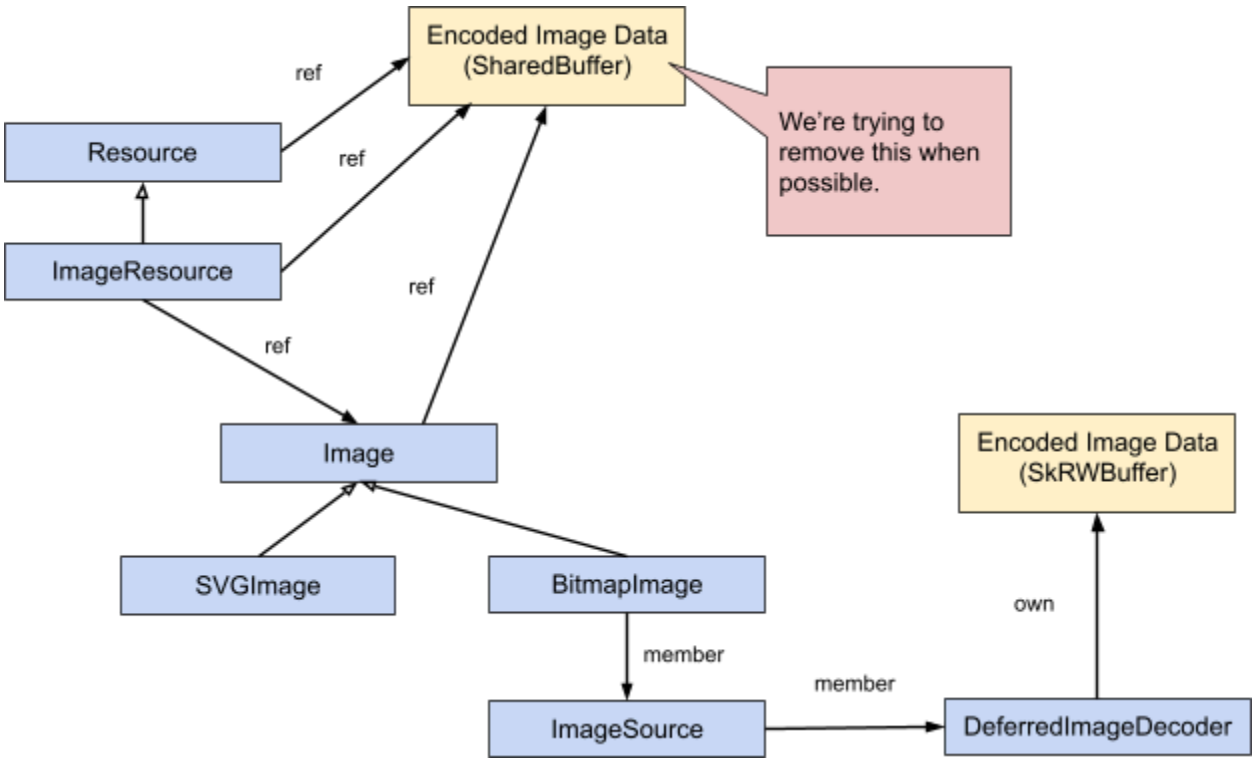
Resource: A class for a HTTP response. One Resource instance exists for one HTTP response.
ImageResource: A subclass of Resource and represents an image resource.

Resource::m_data: A SharedBuffer representing HTTP response body content. In general this is not purged as long as no error happens.
ImageResource::m_data: Same as Resource::m_data. If the response is multipart, m_data represents one part of them. m_data is discarded soon after m_image is created and is recreated each time when another part is received. Note that m_data's lifetime is much different between non-multipart and multipart. This is shared by blink::Image as m_encodedImageData.

ImageResource::m_image: A blink::Image. This is destroyed when pruning and revived when updateImage is called when the resource is used again.

m_data is the original data and m_image is a cache.

blink::Image: Represents an image. This has encoded image data and a reference to a decoder, and might have decoded image data.
blink::BitmapImage: a subclass of blink::Image for bitmap images. This has encoded image data in DeferredImageDecoder's SkRWBuffer and might have decoded image data there. Decoded image data can be destroyed when pruning.
blink::SVGImage: a subclass of blink::Image for SVG image. SVGImage is not affected by the CL and this means encoded image data in SVG is still duplicated.
blink::Image::m_encodedImageData: Represents an encoded image data (SharedBuffer). This is passed from ImageResource::m_data.
blink::Image::m_source: An ImageSource.
blink::ImageSource: Has DeferredImageDecoder as a member.
blink::DeferredImageDecoder: Has encoded image data (SkRWBuffer) and a reference to an actual decoder and might have decoded image data.

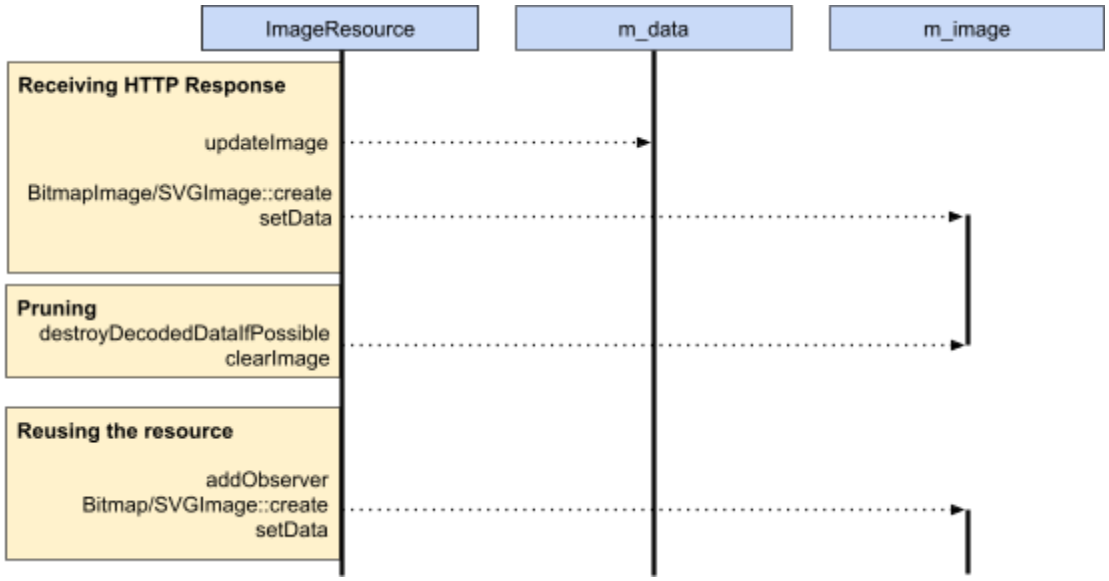


Lifetimes

Before the CL

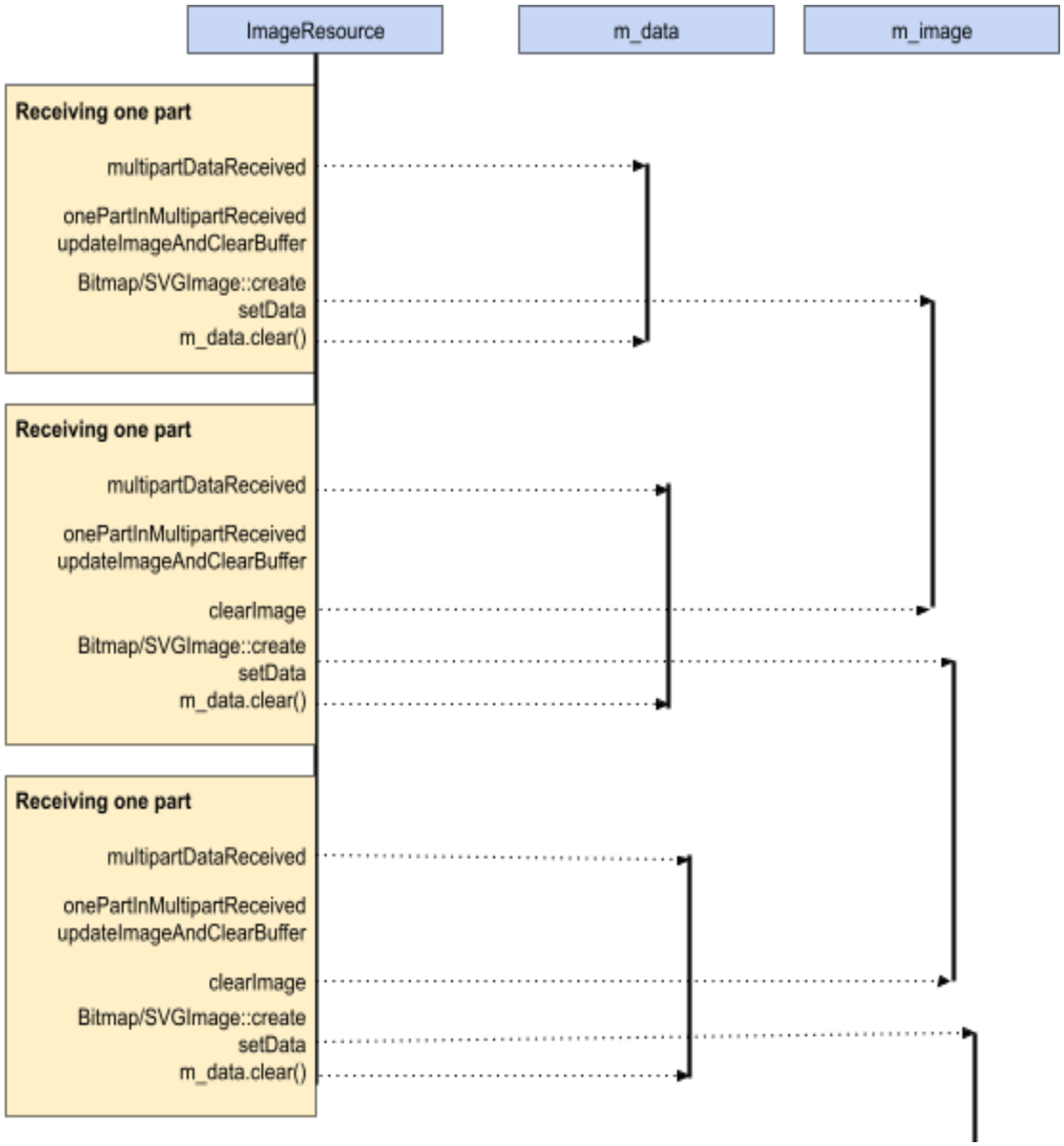
Note that black bold lines in the below figures indicate the state of reference by ImageResource. Even after m_data is derefed from an ImageResource, that data might live because other objects might have a reference to it.

Non-multipart



ImageResource::m_data is basically not discarded but m_image is. m_image is recreated from m_data when necessary.

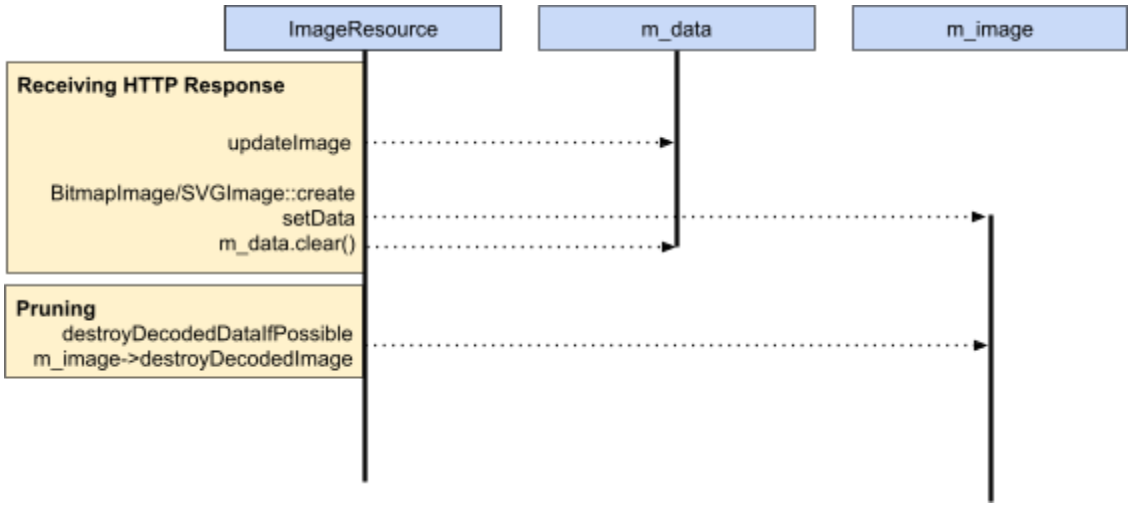
Multipart



ImageResource::m_data is discarded aggressively after each part of the multipart image is received.

After the CL

Non-multipart



In multipart, lifetimes of the objects are not changed. For good or bad, m_data is purged soon after m_image is created.

Misc

Errors

After any error occurs in an ImageResource, m_data and m_image should be cleared (= de-refed from the ImageResource). The CL doesn't change this behavior.

Revalidation

Revalidation can run when the Resource is cached and the new HTTP Response status code is not '304 not modified'. In ImageResource, only failure case of revalidation should be considered and in this case m_data and m_image must be cleared. The CL doesn't change the lifetimes of objects regarding the revalidation.

LoFi

ImageResource can be reloaded when the first loading is LoFi. In this case, Chromium can try to load high-quality image.

1. ResourceFetcher::reloadLoFiImages is called
2. ImageResource::reloadIfLoFi is called. m_image is recreated here if necessary. m_data is cleared after m_image recreation.
3. ResourceFetcher::startLoad is called.
4. ResourceFetcher::didFinishLoading is called.
5. ImageResource::finish is called and m_image is updated with new m_data.

After the CL, m_image is always present and no need to make sure that m_image exists.

Performance Concern

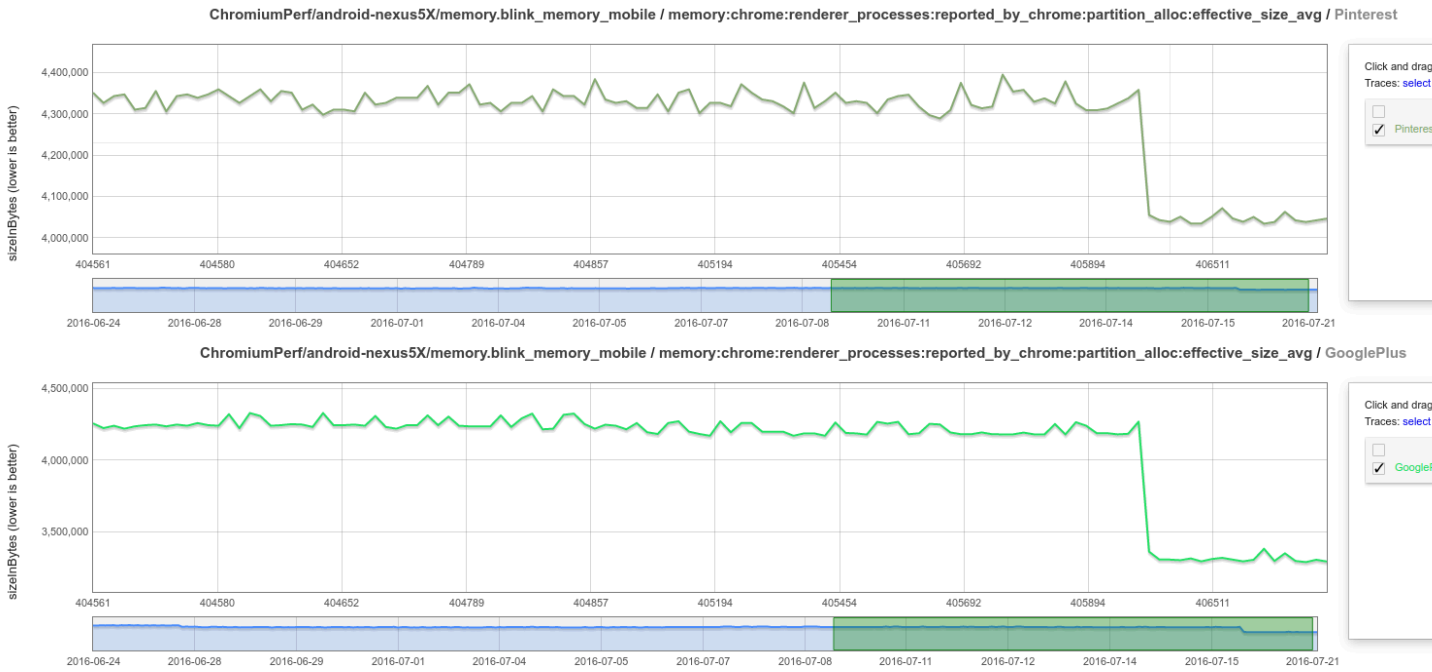
After the CL, BitmapImage::data() returns SharedBuffer generated by SkRWBuffer and this might cause performance regression. This can be used by clipboard, MHTML serialization and WebGL. I tested WebGL tests with telemetry (result). It's a little difficult to say there is no performance regression with the results, and I couldn't get a more stable result on my local machine. There seem to be regressions for some tests (e.g. Earth.html), but as the code says, the tests don't access encoded image data so often.

Result

There are some significant improvements in PartitionAlloc memory usage:

1. https://drive.google.com/file/d/0BwW8PrCcts4WT1ctTFhTVmNZNWc (desktops on Tumblr, Pinterest and Instagram)

2. https://drive.google.com/file/d/0BwW8PrCcts4WT1ctTFhTVmNZNWc (mobiles on Pinterest and Google+)


ChromiumPerf/android-nexus5X/memory.blink_memory_mobile / memory:chrome:renderer_processes:reported_by_chrome:partition_alloc:effective_size_avg / Pinterest
sizeInBytes (lower is better)
4,400,000
4,300,000
4,200,000
4,100,000
4,000,000
404561
404580
404652
404789
404857
405194
405454
405692
405894
406511
2016-06-24
2016-06-28
2016-06-29
2016-07-01
2016-07-04
2016-07-05
2016-07-07
2016-07-08
2016-07-11
2016-07-12
2016-07-14
2016-07-15
2016-07-21
Click and drag
Traces: select
Pinteres
ChromiumPerf/android-nexus5X/memory.blink_memory_mobile / memory:chrome:renderer_processes:reported_by_chrome:partition_alloc:effective_size_avg / GooglePlus
sizeInBytes (lower is better)
4,500,000
4,000,000
3,500,000
404561
404580
404652
404789
404857
405194
405454
405692
405894
406511
2016-06-24
2016-06-28
2016-06-29
2016-07-01
2016-07-04
2016-07-05
2016-07-07
2016-07-08
2016-07-11
2016-07-12
2016-07-14
2016-07-15
2016-07-21
Click and drag
Traces: select
GoogleP