

IMAGE LOADING METHOD AND DEVICE

TECHNICAL FIELD

This application relates to the field of image processing technologies, and in particular, to an image loading method and device.

BACKGROUND

For an Android device, when an image is being loaded, because decoding the image needs to occupy a large amount of a Java heap memory for an application process, and the size of an application process in an Android system is limited, the Java heap memory easily becomes insufficient when usage of the Java heap memory for the application process is relatively high, thereby triggering out of memory (OOM).

To solve this problem, currently, the times of decoding an image can be reduced by caching decoded image data for reuse, and thus reduce the usage of the Java heap memory. For example, after a bitmap object is decoded from a network or a local resource, an Least Recently Used (LRU) cache management can be implemented by using LinkedHashMap (a Map data structure implemented based on link and hash). That is, recently used bitmap objects are saved by using strong reference (e.g., saved to the LinkedHashMap). When the amount of cached data reaches a predetermined value, bitmap objects not frequently used are deleted thereby achieving reuse of the bitmap objects, and improving user application experience.

However, when the existing image cache manner is used to achieve the reuse of decoded image data, a bitmap object is still decoded at the layer of a Java framework, and the decoded bitmap object is cached in the Java heap memory. That is, the memory occupied by the cached image data is still the Java heap memory. Consequently, the Java heap memory is still under great usage pressure, and when decoding images, the heap memory still easily becomes insufficient, thereby triggering OOM.

SUMMARY

Embodiments of this application provide an image loading method and device, to solve the problem that OOM is very easily triggered because pressure of the Java heap memory is still relatively high when decoded image data is reused in an existing image cache manner.

According to one aspect, an embodiment of this application provides an image loading method, including:

- when loading an image, determining whether a native heap memory has decoded data of the image cached therein; and

- if the native heap memory has the decoded data of the image cached therein, obtaining the decoded data of the image from the native heap memory, and loading the image based on the obtained decoded data;

- if the native heap memory does not have the decoded data of the image cached therein, obtaining and loading the image from a storage space storing the image, and caching, into the native heap memory, decoded data of the image obtained in the process of loading the image.

According to another aspect, an embodiment of this application provides an image loading device, including:

- a determining unit, configured to: when loading an image, determine whether a native heap memory has decoded data of the image cached therein; and

- an execution unit, configured to: if it is determined that the native heap memory has the decoded data of the image cached therein, obtain the decoded data of the image from the native heap memory, and load the image based on the obtained decoded data; if it is determined that the native heap memory does not have the decoded data of the image cached therein, obtain and load the image from a storage space storing the image, and cache, into the native heap memory, decoded data of the image obtained in the process of loading the image.

Advantageous effects of the present application are as follows:

The embodiments of this application provide an image loading method and device, where decoded image data may be cached in a native heap memory for reuse during image loading. In this way, because the native heap memory is not counted into a Java heap memory for an application process, pressure of the Java heap memory can be effectively reduced, and probability of triggering OOM is reduced. In addition, image caching can be better implemented by using a heap memory to improve image loading efficiency and reduce system garbage collection (GC).

BRIEF DESCRIPTION OF THE DRAWINGS

To describe the technical solutions in the embodiments of this application more clearly, accompanying drawings are briefly described below. Apparently, the accompanying drawings in the following description show merely some embodiments of this application,

and a person of ordinary skill in the art may still derive other drawings from these accompanying drawings without creative efforts.

FIG. 1 is a possible schematic flowchart of an image loading method according to Embodiment 1 of this application;

FIG. 2 is another possible schematic flowchart of an image loading method according to Embodiment 1 of this application; and

FIG. 3 is a possible schematic structural diagram of an image loading device according to Embodiment 2 of this application.

DETAILED DESCRIPTION OF THE INVENTION

For an Android system, a heap memory may include a Java heap memory and a native heap memory, and the native heap memory is not counted into the Java heap memory. In the embodiments of this application, decoded image data can be cached in the native heap memory based thereupon, for reuse during image loading. In this way, because the native heap memory is not counted into the Java heap memory for application processes, pressure of the heap memory can be effectively reduced, and probability of triggering OOM is reduced. In addition, an image can be cached better by using the heap memory. That is, the decoded data of the image can be reused better, to improve image loading efficiency, reduce system GC, and further improve system performance.

To make the objectives, technical solutions, and advantages of this application clearer, the following further describes this application in detail with reference to the accompanying drawings. Apparently, the described embodiments are merely a part rather than all of the embodiments of this application. All other embodiments obtained by a person of ordinary skill in the art based on the embodiments of this application without creative efforts shall fall within the protection scope of this application.

Embodiment 1

Embodiment 1 of this application provides an image loading method. The image loading method can be applied to any client or terminal device based on an Android system, and this is not limited in this application. As shown in FIG. 1, the image loading method may include the following steps.

Step 101: when loading an image, determine whether native heap memory has decoded data of the image cached therein; and if a result of the determination is yes, perform step 102; otherwise, perform step 103.

Optionally, the decoded data of the image usually is pixel array data of a bitmap object obtained by decoding the image. In addition, in this embodiment of this application, when decoded data of any image is cached in the native heap memory, corresponding record data is usually generated in cache record information of the native heap memory (NativeCacheInfo). Therefore, the determining whether native heap memory has cached decoded data of the image may include:

determining whether the record data corresponding to the decoded data of the image can be obtained from the NativeCacheInfo, and if a result of the determination is yes, determining that the native heap memory has the decoded data of the image cached therein; otherwise, determining that the native heap memory does not have the decoded data of the image cached therein.

Each piece of record data of the NativeCacheInfo may include attribute information such as a width, a height, a native heap memory pointer (“a native pointer,” used to indicate a storage address of decoded data corresponding to the piece of record data in the native heap memory), Bitmap.Config (used to indicate of the amount of memory occupied by each pixel of the image), and a data length of decoded data of an image corresponding to the piece of record data. In addition, for convenience of searching and recording, each piece of record data usually may be marked with identification information that can uniquely identify the image corresponding to the piece of record data (or with identification information that can uniquely identify decoded data of the image corresponding to the piece of record data), and details are not described herein.

Furthermore, because memory occupied by the NativeCacheInfo is usually relatively small, the NativeCacheInfo may be stored in the Java heap memory (certainly, to further reduce storage pressure of the Java heap memory, the NativeCacheInfo may alternatively be stored in the native heap memory). In addition, record data in the NativeCacheInfo may be managed by using LRU, to maintain the record data in the NativeCacheInfo and cached image data in a heap memory in real time, and synchronously release memory occupied by decoded data corresponding to a piece of record data in the native heap memory when the piece of record data is deleted, to further reduce the pressure of the heap memory and reduce probability of triggering OOM.

In addition, it should be noted that in the embodiments of this application, other manners can be used to determine whether the native heap memory has the decoded data of the image cached therein. For example, data cached in the native heap memory can be scanned piece by piece to determine whether the native heap memory has the decoded data of the image cached

therein. This is not limited in this application.

Step 102: obtain the decoded data of the image from the native heap memory and load the image based on the obtained decoded data.

Optionally, as shown in FIG. 2, step 102 of obtaining the decoded data of the image from the native heap memory and loading the image based on the obtained decoded data may include the following steps.

Step 102A: determine whether a bitmap object currently displayed on an image view satisfies a reuse condition for the decoded data of the image; and if the bitmap object currently displayed on the image view satisfies the reuse condition for the decoded data of the image, perform step 102B; if the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image, perform step 102C.

Optionally, for a system with a version lower than Android 4.4, when the width, the height, and the Bitmap.Config of an image corresponding to the bitmap object currently displayed on the image view are completely consistent with the width, the height, and the Bitmap.Config of an image corresponding to the decoded data of the image, and an attribute of the bitmap object the currently displayed on the image view is mutable (that is, modifiable), it is considered that the bitmap object currently displayed on the image view satisfies the reuse condition for the decoded data of the image, and otherwise, it is considered that the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image.

For a system with a version of Android 4.4 or higher, when a product of the width of an image corresponding to the bitmap object currently displayed on the image view, the height of the image, and the amount of memory occupied by each pixel of the image is not smaller than a data length of the decoded data of the image, it may be considered that the bitmap object currently displayed on the image view satisfies the reuse condition for the decoded data of the image, and otherwise, it is considered that the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image.

Step 102B: copy pixel array data of a bitmap object cached in the native heap memory and obtained by decoding the image into the bitmap object currently displayed on the image view to obtain a new bitmap object, and render the new bitmap object into the image view to load the image.

That is, when the image view has displayed an image A (Bitmap-A), if an image B (Bitmap-B) needs to be displayed, the image A displayed on the image view does not need to be replaced with the image B, and instead, the image B only needs to be bound with the

image view, and pixel array data of the image B is copied into the image A, to improve image loading efficiency.

Optionally, to prevent old data in the image view from interfering with display of new data, before the pixel array data of the bitmap object of the image is copied into the bitmap object currently displayed on the image view, the pixel array data of the bitmap object currently displayed on the image view may be removed first, and details are not described herein.

Step 102C: obtain, from a reusable bitmap pool (a bitmap file pool), a bitmap object satisfying the reuse condition for the decoded data of the image, or create a bitmap object satisfying the reuse condition for the decoded data of the image, copy pixel array data of a bitmap object cached in the native heap memory and obtained by decoding the image into the obtained or created bitmap object, to obtain a new bitmap object, and render the new bitmap object into the image view, to load the image.

Optionally, as shown in FIG. 2, before step 102C is performed, the bitmap object currently displayed on the image view may be first stored into the reusable bitmap pool (which corresponds to step 102C'), for reuse during another image loading (because although the current bitmap object does not necessarily satisfy a reuse condition for a current to-be-loaded image, the current bitmap object is very likely to satisfy a reuse condition for another image), to further improve the image loading efficiency and reduce system GC.

The reusable bitmap pool may usually be located in the Java heap memory, and may be referenced through soft reference so that when memory is sufficient, an original resource may be effectively used to reduce GC, and when a system needs memory, the memory may be released in time without causing OOM. In addition, to reduce storage pressure of the heap memory, each bitmap object stored in the reusable bitmap pool may alternatively be a bitmap object with pixel array data removed, that is, a bitmap object framework that can be reused by another image.

Further, it should be noted that, in a process of performing step 102C, it may be determined first whether a bitmap object satisfying the reuse condition for the decoded data of the image can be obtained from the reusable bitmap pool. If the bitmap object satisfying the reuse condition for the decoded data of the image can be obtained from the reusable bitmap pool, the operation of creating a bitmap object satisfying the reuse condition for the decoded data of the image does not need to be performed (that is, when the bitmap object satisfying the reuse condition for the decoded data of the image cannot be obtained from the reusable bitmap pool, the operation of creating a bitmap object satisfying the reuse condition

for the decoded data of the image is performed), to further improve the image loading efficiency and reduce the system GC.

Optionally, usually, a bitmap object satisfying the reuse condition for the decoded data of the image can be created by invoking a system application programming interface (API) and inputting a width, a height, and Bitmap.Config of a to-be-displayed image, and details are not described herein.

Furthermore, it should be noted that, to prevent old data from interfering with display of new data, before the pixel array data of the bitmap object of the image is copied into the obtained bitmap object or the created bitmap object, if it is determined that pixel array data in the obtained bitmap object or the created bitmap object is not empty, the pixel array data in the obtained bitmap object or the created bitmap object may be cleared first, and details are not described herein.

Further, optionally, it can be learned from FIG. 2 that, before the decoded data of the image is obtained from the native heap memory, the method may further include the following step:

Step 104: determine whether the decoded data of the image cached in the native heap memory is valid, and determine that a determining result is yes.

That is, usually, when it is determined that the decoded data of the image cached in the native heap memory is valid, the decoded data of the image is then obtained from the native heap memory.

Optionally, the determining whether the decoded data of the image cached in the native heap memory is valid may include:

determining whether a native pointer in record data that is in the NativeCacheInfo and corresponds to the decoded data of the image is a valid pointer, and if a result of the determination is yes, determining that the decoded data of the image cached in the native heap memory is valid.

In addition, it can be learned from FIG. 2 that, if it is determined that the decoded data of the image cached in the native heap memory is invalid, the method may directly jump to step 103, that is, the operation of obtaining and loading the image from a storage space storing the image and caching decoded data of the image obtained in the process of loading the image into the native heap memory is performed.

Further, before the decoded data of the image is obtained from the native heap memory, the method may further include the following step:

if it is determined that the image corresponding to the bitmap object currently displayed

on the image view is not loaded for the first time, obtaining the pixel array data of the bitmap object currently displayed on the image view, and caching the pixel array data into the native heap memory for reuse during the next time of display.

In addition, as described above, when the pixel array data of the bitmap object currently displayed on the image view is cached into the native heap memory, corresponding record data may be generated in the NativeCacheInfo, and details are not described herein.

Step 103: obtain and load the image from the storage space storing the image, and cache, into the native heap memory, decoded data of the image obtained in the process of loading the image.

Optionally, the storage space storing the image may be a local disk cache, a local file, a network, or the like. In addition, loading the obtained image may include operations such as decoding the obtained image to obtain the bitmap object, and rendering the bitmap object.

In addition, as described above, when the decoded data of the image obtained in the process of loading the image is cached into the native heap memory, corresponding record data may be generated in the NativeCacheInfo.

Thereby, the loading operation of the current to-be-loaded image can be completed.

It can be learned from the content of this embodiment of this application that, in the embodiments of this application, decoded image data may be cached in the native heap memory for reuse during image loading. In this way, because the native heap memory is not counted into the Java heap memory for an application process, pressure of the Java heap memory can be effectively reduced and probability of triggering OOM is reduced. In addition, an image can be cached better by using a heap memory to improve image loading efficiency and reduce system GC.

Embodiment 2

Based on the same inventive concept as that of the image loading method in Embodiment 1 of this application, Embodiment 2 of this application provides an image loading device. For details about implementations of the image loading device, reference may be made to related descriptions in Method Embodiment 1, and overlapping details are not repeated herein. As shown in FIG. 3, the image loading device may include:

a determining unit 31, configured to: when loading an image, determine whether native heap memory has decoded data of the image cached therein; and

an execution unit 32, configured to: if it is determined that the native heap memory has the decoded data of the image cached therein, obtain the decoded data of the image from the native heap memory, and load the image based on the obtained decoded data; if it is

determined that the native heap memory does not have the decoded data of the image cached therein, obtain and load the image from a storage space storing the image, and cache, into the native heap memory, decoded data of the image obtained in the process of loading the image.

Optionally, the determining unit 31 may be configured to: determine whether record data corresponding to the decoded data of the image can be obtained from cache record information of the native heap memory, and if a result of the determining is yes, determine that the native heap memory has the decoded data of the image cached therein.

Optionally, the execution unit 32 may be configured to: when it is determined that the decoded data of the image cached in the native heap memory is valid, obtain the decoded data of the image from the native heap memory.

Optionally, the execution unit 32 may be configured to: determine whether a native heap memory pointer in the record data that is in the cache record information of the native heap memory and corresponds to the decoded data of the image is a valid pointer, and if a result of the determining is yes, determine that the decoded data of the image cached in the native heap memory is valid;

where a native heap memory pointer in each piece of record data in the cache record information of the native heap memory is used to indicate a storage address of decoded data corresponding to the record data in the native heap memory.

In addition, it should be noted that, the native heap memory cache record information may be located in Java heap memory, and the record data in the cache record information of the native heap memory is managed by using LRU.

Optionally, the decoded data of the image usually is pixel array data of a bitmap object obtained by decoding the image.

The execution unit 32 may be configured to: determine whether a bitmap object currently displayed by an image view satisfies a reuse condition for the decoded data of the image; and if the bitmap object currently displayed by the image view satisfies the reuse condition for the decoded data of the image, copy pixel array data of a bitmap object cached in the native heap memory and obtained by decoding the image into the bitmap object currently displayed by the image view to obtain a new bitmap object, and render the new bitmap object into the image view to load the image;

if the bitmap object currently displayed by the image view does not satisfy the reuse condition for the decoded data of the image, obtain, from a bitmap file pool, a bitmap object satisfying the reuse condition for the decoded data of the image, or create a bitmap object satisfying the reuse condition for the decoded data of the image, copy pixel array data of a

bitmap object cached in the native heap memory and obtained by decoding the image into the obtained or created bitmap object, to obtain a new bitmap object, and render the new bitmap object into the image view, to load the image;

where the bitmap file pool is located in a Java heap memory, and its reference manner is soft reference.

Further, optionally, the execution unit 32 may be further configured to: if it is determined that the bitmap object currently displayed by the image view does not satisfy the reuse condition for the decoded data of the image, store the bitmap object currently displayed by the image view into the bitmap file pool.

Persons skilled in the art should understand that, embodiments of this application may be provided as methods, apparatuses (devices), or computer program products. Therefore, this application may take the form of total hardware embodiments, total software embodiments, or embodiments combining software and hardware. In addition, this application may take the form of a computer program product implemented on one or more computer-usable storage media (including, but not limited to, a magnetic storage device, a CD-ROM, and an optical storage device) including computer-usable program code.

This application is described with reference to flowcharts and/or block diagrams of the method, the apparatus (the device), and the computer program product in the embodiments of this application. It should be understood that computer program instructions may be used to implement each process and/or each block in the flowcharts and/or the block diagrams and a combination of a process and/or a block in the flowcharts and/or the block diagrams. These computer program instructions may be provided for a general-purpose computer, a dedicated computer, an embedded processor, or a processor of any other programmable data processing device to generate a machine, so that the instructions executed by a computer or a processor of any other programmable data processing device generate an apparatus for implementing a specific function in one or more processes in the flowcharts and/or in one or more blocks in the block diagrams.

These computer program instructions may also be stored in a computer readable memory that can instruct a computer or any other programmable data processing device to work in a specific manner, so that the instructions stored in the computer readable memory generate an artifact that includes an instruction apparatus. The instruction apparatus implements a specific function in one or more processes in the flowcharts and/or in one or more blocks in the block diagrams.

These computer program instructions may also be loaded onto a computer or another

programmable data processing device, so that a series of operations and steps are performed on the computer or the another programmable device, thereby generating computer-implemented processing. Therefore, the instructions executed on the computer or the another programmable device provide steps for implementing a specific function in one or more processes in the flowcharts and/or in one or more blocks in the block diagrams.

Although preferable embodiments of this application have been described, once persons skilled in the art know the basic creative concept, they can make additional changes and modifications to these embodiments. Therefore, the following claims are intended to be construed as to cover the preferable embodiments and all changes and modifications falling within the scope of this application.

Obviously, persons skilled in the art can make various modifications and variations to this application without departing from the spirit and scope of this application. In this case, if the modifications and variations made to this application fall within the scope of the claims of this application and their equivalent technologies, this application is intended to include these modifications and variations.

CLAIMS

What is claimed is:

1. An image loading method, comprising:
 - when loading an image, determining whether a native heap memory has decoded data of the image cached therein;
 - if the native heap memory has the decoded data of the image cached therein, obtaining the decoded data of the image from the native heap memory and loading the image based on the obtained decoded data; and
 - if the native heap memory does not have the decoded data of the image cached therein, obtaining and loading the image from a storage space storing the image, and caching, into the native heap memory, decoded data of the image obtained in the process of loading the image.
2. The method according to claim 1, wherein the determining whether native heap memory has decoded data of the image cached therein comprises:
 - determining whether record data corresponding to the decoded data of the image can be obtained from cache record information of the native heap memory, and if a result of the determining is yes, determining that the native heap memory has the decoded data of the image cached therein.
3. The method according to claim 1 or 2, wherein when determining that the decoded data of the image cached in the native heap memory is valid, obtaining the decoded data of the image from the native heap memory.
4. The method according to claim 3, wherein the determining whether the decoded data of the image cached in the native heap memory is valid comprises:
 - determining whether a native heap memory pointer in the record data that is in the cache record information of the native heap memory and that corresponds to the decoded data of the image is a valid pointer, and if a result of the determining is yes, determining that the decoded data of the image cached in the native heap memory is valid,
 - wherein a native heap memory pointer in each piece of record data in the cache record information of the native heap memory is used to indicate a storage address of the decoded data corresponding to the record data in the native heap memory.
5. The method according to claim 2, wherein the cache record information of the native heap memory is located in a Java heap memory, and each piece of record data in the cache record information of the native heap memory is managed by using an Least Recently Used (LRU) method.

6. The method according to claim 1 or 2, wherein the decoded data of the image is pixel array data of a bitmap object obtained by decoding the image; and

the obtaining the decoded data of the image from the native heap memory and loading the image based on the obtained decoded data comprises:

determining whether a bitmap object currently displayed on an image view satisfies a reuse condition for the decoded data of the image;

if the bitmap object currently displayed on the image view satisfies the reuse condition for the decoded data of the image, copying the pixel array data of the bitmap object cached in the native heap memory and obtained by decoding the image into the bitmap object currently displayed on the image view to obtain a new bitmap object, and rendering the new bitmap object into the image view to load the image; and

if the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image, obtaining, from a bitmap file pool, a bitmap object satisfying the reuse condition for the decoded data of the image, or creating a bitmap object satisfying the reuse condition for the decoded data of the image, copying the pixel array data of the bitmap object cached in the native heap memory and obtained by decoding the image into the obtained or created bitmap object to obtain a new bitmap object, and rendering the new bitmap object into the image view to load the image;

wherein

the bitmap file pool is located in a Java heap memory, and a reference manner of the bitmap file pool is soft reference.

7. The method according to claim 6, wherein if it is determined that the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image, the method further comprises:

storing the bitmap object currently displayed on the image view into the bitmap file pool.

8. An image loading device, comprising:

a determining unit configured to: when loading an image, determine whether a native heap memory has decoded data of the image cached therein; and

an execution unit configured to: if it is determined that the native heap memory has the decoded data of the image cached therein, obtain the decoded data of the image from the native heap memory, and load the image based on the obtained decoded data; if it is determined that the native heap memory does not have the decoded data of the image cached therein, obtain and load the image from a storage space storing the image, and cache, into the

native heap memory, decoded data of the image obtained in the process of loading the image.

9. The device according to claim 8, wherein

the determining unit is configured to: determine whether record data corresponding to the decoded data of the image can be obtained from cache record information of the native heap memory, and if a result of the determining is yes, determine that the native heap memory has the decoded data of the image cached therein.

10. The device according to claim 8 or 9, wherein

the execution unit is configured to: when determining that the decoded data of the image cached in the native heap memory is valid, obtain the decoded data of the image from the native heap memory.

11. The device according to claim 10, wherein

the execution unit is configured to: determine whether a native heap memory pointer in the record data that is in the cache record information of the native heap memory and corresponds to the decoded data of the image is a valid pointer, and if a result of the determining is yes, determine that the decoded data of the image cached in the native heap memory is valid, wherein

a native heap memory pointer in each piece of record data in the cache record information of the native heap memory is used to indicate a storage address of the decoded data corresponding to the record data in the native heap memory.

12. The device according to claim 9, wherein the cache record information of the native heap memory is located in a Java heap memory, and each piece of record data of the cache record information of the native heap memory is managed by using an Least Recently Used (LRU) method.

13. The device according to claim 9 or 10, wherein the decoded data of the image is pixel array data of a bitmap object obtained by decoding the image; and

the execution unit is configured to: determine whether a bitmap object currently displayed on an image view satisfies a reuse condition for the decoded data of the image; and if the bitmap object currently displayed on the image view satisfies the reuse condition for the decoded data of the image, copy the pixel array data of the bitmap object cached in the native heap memory and obtained by decoding the image into the bitmap object currently displayed on the image view to obtain a new bitmap object, and render the new bitmap object into the image view to load the image; and

if the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image, obtain, from a bitmap file pool, a bitmap object

satisfying the reuse condition for the decoded data of the image, or create a bitmap object satisfying the reuse condition for the decoded data of the image, copy the pixel array data of the bitmap object cached in the native heap memory and obtained by decoding the image into the obtained or created bitmap object to obtain a new bitmap object, and render the new bitmap object into the image view to load the image, wherein

the bitmap file pool is located in a Java heap memory, and a reference manner of the bitmap file pool is soft reference.

14. The device according to claim 13, wherein

the execution unit is further configured to: if it is determined that the bitmap object currently displayed on the image view does not satisfy the reuse condition for the decoded data of the image, store the bitmap object currently displayed on the image view into the bitmap file pool.