Caching Java Heap Objects

Author: Jiangli Zhou (jiangli), Thomas Schatzl (tschatzl)

This page gives some details how the Hotspot VM maintains archived (cached) Java heap objects.

Supported Platforms and Configurations

- Supports 64-bit, non-Windows platforms only.
- Supports G1 GC only.
- Requires UseCompressedOops and UseCompressedClassPointers set.

Types of Pinned G1 Heap Regions

The pinned region type is a super type of all archive region types, which include the open archive type and the closed archive type.

- **00100 0 [8] Pinned Mask**
- **01000 0 [16] Old Mask**
- **10000 0 [32] Archive Mask**
- **11100 0 [56] Open Archive**: ArchiveMask | PinnedMask
- **11100 1 [57] Archive** : ArchiveMask | PinnedMask + 1

Pinned Regions

Objects within the region are 'pinned', which means GC does not move any live objects. GC scans and marks objects in the pinned region as normal, but skips forwarding live objects. Pointers in live objects are updated. Dead objects (unreachable) can be collected and freed.

Archive-type Regions

The archive-types are sub-types of 'pinned'. There are two types of archive region currently, open archive and closed archive. Both can support caching Java heap objects via the CDS (Class Data Sharing) archive.

Open Archive (GC-RW) Regions

Open archive region is GC writable. GC scans & marks objects within the region and adjusts (updates) pointers in live objects the same way as a pinned region. Live objects (reachable) are pinned and not forwarded by GC.

Open archive region does not have 'dead' objects. Unreachable objects are 'dormant' objects. Dormant objects are not collected and freed by GC. Please see Dormant Objects for details.

Caching Java objects with open archive region may help JVM startup and runtime performance.

Open Archive Heap Region Verification

All live objects and dormant objects are verified to make sure they only point to live objects or archived objects.

Adjustable Outgoing Pointers

As GC can adjust pointers within the live objects in open archive heap region, objects can have outgoing pointers to another Java heap region, including closed archive region, open archive region, pinned (or humongous) region, and normal generational region (young, old). When a referenced object is moved by GC, the pointer within the open archive region is updated accordingly.

Closed Archive (GC-RO) Regions

The closed archive region is GC read-only region. GC cannot write into the region. Objects are not scanned and marked by GC. Objects are pinned and not forwarded. Pointers are not updated by GC either. Hence, objects within the archive region cannot have any outgoing pointers to another Java heap region. Objects however can still have pointers to other objects within the closed archive regions (we might allow pointers to open archive regions in the future). That restricts the type of Java objects that can be supported by the archive region.

Supported Java objects:

- Primitive arrays (not changed at runtime once initialized)
- String and the underlying 'value' array
- Closed and runtime 'immutable' object graph
    - Not changed at runtime once initialized
    - No out-going references to non-closed archive GC regions

In JDK 9 we added the support for shared Strings with the (closed) archive regions.

The GC-readonly archive region makes Java heap memory sharable among different JVM processes.

NOTE: synchronization on the objects within the archive heap region can still cause writes to the memory page.

Closed Archive Regions Verification

Objects can only point to objects within the closed Archive region.

Dormant Java Objects

Dormant Java objects are unreachable Java objects residing in the open archive heap region(s).

A Java object in the open archive heap region is a live object if it can be reached during scanning. Some of the Java objects in the open region may not be reachable during scanning. Those objects are considered as dormant, but not dead. For example, a constant pool 'resolved_references' array is reachable via the klass root only if its container klass (shared) is already loaded at the time during the current GC scanning. If a shared klass is not yet loaded, the klass root is not scanned and it's constant pool 'resolved_reference' array (A) in the open archive region is not reachable. Then A is a dormant object.

Object State Transition

All Java objects are initially dormant objects when open archive heap regions are mapped to the runtime Java heap. A dormant object becomes live object when the associated shared class is loaded at runtime. Explicit call to G1SATBCardTableModRefBS::enqueue() needs to be made when a dormant object becomes live. That should be the case for cached objects with strong roots as well, since strong roots are only scanned at the start of GC marking (the initial marking) but not during Remarking/Final marking. If a cached object becomes live during concurrent marking phase, G1 may not find it and mark it live unless a call to G1SATBCardTableModRefBS::enqueue() is made for the object.

Currently, a live object in the open archive heap region cannot become dormant again. This restriction simplifies GC requirement and guarantees all outgoing pointers are updated by GC correctly.

Only objects for shared classes from the builtin class loaders (the boot loader, PlatformClassLoaders, and AppClassLoaders) are supported for caching currently.

New/Updated GC APIs

G1ArchiveAllocator

- G1ArchiveAllocator(G1CollectedHeap* g1h, bool open);
- static G1ArchiveAllocator* G1ArchiveAllocator::create_allocator(G1CollectedHeap* g1h, bool open);
- static inline void G1ArchiveAllocator::set_range_archive(MemRegion range, bool open);
- static inline bool G1ArchiveAllocator::is_closed_archive_object(oop object);
- static inline bool G1ArchiveAllocator::is_open_archive_object(oop object);
- static inline bool G1ArchiveAllocator::is_archive_object(oop object);
- static inline bool G1ArchiveAllocator::in_closed_archive_range(oop object);
- static inline bool G1ArchiveAllocator::in_open_archive_range(oop object);

G1CollectedHeap

- void G1CollectedHeap::begin_archive_alloc_range(bool open = false);

HeapRegionType

- bool HeapRegionType::is_archive();
- bool HeapRegionType::is_open_archive();

Types of Cached Java Objects

In Closed Archive Heap Region

- Interned Java.lang.String objects and the underlying 'value' objects
- Primitive box caches and the cached objects (JDK-8209120, JDK-8213033)

In Open Archive Heap Region

- Class mirror objects (Java.lang.Class objects)
- Class constant pool resolved_references arrays
- Module Graph (Sub-graph of Java Heap Objects)
- System Module Boot layer Configuration (JDK-8207263)

Caching Java Objects at Archive Dump Time

The closed archive and open archive regions are allocated at the top (or near the top) of the dump time Java heap. Archived Java objects are copied into the designated archive heap regions. For example, String objects and the underlying 'value' arrays are copied into the closed archive regions. All references to the archived objects (from shared class metadata, string table, etc) are set to the new heap locations. A hash table is used to keep track of all archived Java objects during the copying process to make sure Java object is not archived more than once if reached from different roots. It also makes sure references to the same archived object are updated using the same new address location.

Caching Constant Pool resolved_reference Array

The 'resolved_references' is an array that holds references of resolved constant pool entries including Strings, mirrors and methodTypes, etc. Each loaded class has one 'resolved_references' array (in ConstantPoolCache). The 'resolved_references' arrays are copied into the open archive regions during dump process. Prior to copying the 'resolved_references' arrays, JVM iterates through constant pool entries and resolves all JVM_CONSTANT_String entries to existing interned Strings for all archived classes. When resolving, JVM only looks up the string table and finds existing interned Strings without inserting new ones. If a string entry cannot be resolved to an existing interned String, the constant pool entry remain as unresolved. That prevents memory waste if a constant pool string entry is never used at runtime.

All String objects referenced by the string table are copied first into the closed archive regions. The string table entry is updated with the new location when each String object is archived. The JVM updates the resolved constant pool string entries with the new object locations when copying the 'resolved_references' arrays to the open archive regions. References to the 'resolved_references' arrays in the ConstantPoolCache are also updated.

At runtime as part of ConstantPool::restore_unshareable_info() work, call G1SATBCardTableModRefBS::enqueue() to let GC know the 'resolved_references' is becoming live. A handle is created for the cached object and added to the loader_data's handles.

Runtime Java Heap With Cached Java Objects

The closed archive regions (the string regions) and open archive regions are mapped to the runtime Java heap at the same offsets as the dump time offsets from the runtime Java heap base.