CDS Archived Heap Improvements

1. Overview

In the JDK 16 and 17 timeframe, we plan to enhance the CDS Archived Heap to implement the following:

- Fix GC bug in G1 (JDK-8253081)to support archiving the full module graph (JDK-8244778)
    - This requires the Archived Heap to maintain a **Root List**.
- Support CDS Archived Heap for other collectors
    - Phase 1: ParallelGC and SerialGC (which support compressed oops)
    - Phase 2: ZGC (which requires non-compressed oops)

As shown below, the Root List makes it easy to support other collectors.

2. Archived Heap in JDK 15

This section describes the implementation, and the problems, of the Archived Heap in JDK 15.

In JDK 15, the Archived Heap is supported only by G1 with compressed oops. Other types of GC are not supported. Uncompressed oops are not supported.

The Archived Heap is divided into two parts

- **Closed Region**: Objects in this region can point to only other objects in this region. Today this region is used to store the archived String objects and their corresponding character arrays.
    - Objects in this region are alive forever. They are never collected.
    - Objects in this region are immutable.
- **Open Region**: All other archived objects are stored in this region.
    - Objects in this region can reference any objects (including those that are outside of the Archived Heap).
    - Objects in this region are alive forever. They are never collected.

The Open Region is problematic:

- It contains objects that are not reachable at VM bootstrap, but may become reachable later.
    - For example, archived mirror objects (instances of java.lang.Class) are stored in the Open Region. An archived mirror become reachable only when its corresponding C++ Klass object is loaded by the SystemDictionary.
- As a result, G1 cannot conclude that an archived object is garbage just because it's unreachable.
- Moreover, there's no way for G1 to distinguish between unreachable objects that
    - **Type A**: will later become reachable
    - **Type B**: were once reachable, but have become unreachable

Luckily, in JDK 15, all unreachable objects in the Open Region are of **Type A**. I.e., once we have made an archived object reachable, it will always be reachable.

3. Challenges in JDK 16

In JDK 16, we have implemented JDK-8244778 (Archive full module graph in CDS). With this change, we have introduced objects of **Type B** into the Open Region:

- The full module graph contains HashMaps (of the currently loaded packages, etc) which are backed by Object[] arrays.
    - Initially, all of these backing arrays are in the Open Region.
- When new classes are loaded, these HashMaps are updated
    - As a result, a non-archived object may be stored in to an archived backing array **ARR**.
    - i.e., Object OBJ = new Object(); ARR[n] = OBJ;
- Under some circumstances, ARR becomes garbage (e.g., when the HashMap is expanded)
    - ARR is now a **Type B** object.
    - Later, a GC happens, and OBJ is relocated due to GC
    - However, ARR[n] is not relocated (because ARR is not reachable??). As a result, ARR[n] becomes invalid (points to garbage)
    - Under some circumstances (GC verification – and possibly other situations as well??), ARR[n] is dereferenced and we would get an assert or crash in the GC.

4. Root List - Distinguishing Live and Dead Objects in Open Region

In JDK 16, with JDK-8253081, we will implement a Root List in the Archived Heap to allow the GC to reliably determine whether an archived object is alive or dead:

- Prototype: http://cr.openjdk.java.net/~iklam/jdk16/8253081-open-archive-root-list/
- Start at HeapShared::roots(). This is where the Root List is implemented

At VM bootstrap, the Root List contains all objects in the Open Region that **may** be referenced in the future, including

- mirrors of classes that have not yet been loaded.
- ConstantPool::resolved_references() of classes that have not yet been loaded.
- Objects in ArchivedKlassSubGraphInfoRecords that have not been initialized.
- java.lang.Module objects that have not yet been added to the module graph.

When a mirror M becomes referenced by a newly loaded class K, M will be removed from the Root List, and K will be responsible for keeping M alive.

Other types of roots are also similarly removed from the Root List when they become referenced.

The Root List is implemented as a ObjArray stored inside a global OopHandle, so it will always be scanned by the GC. Therefore, the GC can use reachability to tell whether an object in the Open Region is live or dead. In fact, the Open Region now becomes just like other regular G1 regions. We should be able to remove most of G1's special handling code for the Open Region.

In the example above:

- At VM bootstrap, the ARR array is referenced indirectly by an ArchivedKlassSubGraphInfoRecord, whose root object is stored in the Root List, so it's reachable.
- Later, when the ArchivedKlassSubGraphInfoRecord is initialized, its root object is removed from the Root List and stored into Java static field (e.g., ArchivedBootLayer. archivedBootLayer), and ARR will be indirectly referenced from there.
- Later, when ARR is discarded by the HashMap due to table expansion, it is no longer reachable. G1 knows that ARR is dead and will never dereference ARR[n].

5. Support Archived Heap for other collectors

With the Root List, it should be fairly easy to support Archived Heap for SerialGC and ParallelGC. See JDK-8234679 - Support CDS shared heap in non-G1 garbage collectors.

- At VM bootstrap, reserve enough space in the old generation
- Copy objects in the Open Region and Closed Region into the old generation
  - Relocation may be necessary, since the Open/Closed regions assume that they are mapped to a pre-determined address, which may be different than the location of the old generation
  - There's already code for doing this kind of relocation (see HeapShared::patch_archived_heap_embedded_pointers)
  - The Open/Closed regions may be divided into at most 4 blocks with gaps in between. The easiest way to handle the gaps is to fill them with a dummy array.
- Even with the copying/patching, the start up time should still be much faster than running without the Archived Heap. Without the Archived Heap, we need to create many expensive data structures (such as the module graph) from scratch, which is very time consuming.
  - G1 with Archived Heap = 32ms
  - G1 without Archived Heap = 57ms
  - Copying/patching = less than 1ms.

6. Support Uncompressed Oops for the Archived Heap

In JDK 15, the class metadata references archived heap objects using narrowOop (e.g. Klass::_archived_mirror). With the Root List, these references are changed to be integer indices. As a result, most of the Archived Heap code should be agnostic to oop encoding. This will make it easier to support uncompressed oops in Archived Heap.