Customization of ResourceManager's JVM Interaction

To be Reviewed By: March 31, 2022

Authors: Darrel Schneider

Status: Draft | Discussion | Active | Dropped | Superseded

Superseded by: N/A

Related: N/A

Problem

The Geode ResourceManager needs to obtain information from the JVM about how big the heap is and how much of it is in use. Some JVM vendors may have proprietary APIs that if the ResourceManager used them would cause the ResourceManager to perform its job better. Or Geode users may want to run on a newer release of java that has better ways to obtain this information. But because Geode is built with the standard Java 8 release, it is not able to have calls to APIs that are not part of Java 8.

Anti-Goals

This proposal is not adding support for new garbage collectors or improving how Geode uses the ResourceManager to do eviction or prevent out of memory exceptions.

Solution

The ResourceManager only needs two items from the JVM: the maximum size of the heap and the amount of the heap in use. To allow Geode to call APIs that did not exist in Java 8, we will add a single interface, HeapUsageProvider that a user can implement with a class that must have a zero-argument constructor. The user will specify that it wants to use its class by setting the system property "geode.resource-manager.heap-usage-provider.class-name" to the name of their class. This system property would need to be set on each server that they want it to be used on. Their class will also need to be made loadable on the server by adding it to the server's class path or deploying it to the server. Geode will load the class when the Cache is created. If the class fails to load, or a new instance can not be created, or the instance does not implement HeapUsageProvider, then the Cache creation will fail with an exception describing the problem. The single instance created when the Cache is created will be used by Geode until the Cache is closed at which point the HeapUsageProvider close method will be called.

HeapUsageProvider implementations must return meaningful values when asked for the maximum and used heap amounts. It can optionally send notifications to Geode when the used memory changes.

By default Geode supplies its own internal HeapUsageProvider which will be used if the user does not set the "geode.resource-manager.heap-usage-provider.class-name" system property. Before implementing your own HeapUsageProvider it would be wise to look at the Geode source of its internal HeapUsageProvider to see if it can be used instead.

A draft PR of this solution can be found here: https://github.com/apache/geode/pull/7456

Changes and Additions to Public Interfaces

One new public Interface will be added:
package org.apache.geode.cache.control;
import java.util.function.LongConsumer;

/**
 * This interface can be implemented with a class that is then specified using the
 * @link {@javax.annotation.Resource(name="heap-usage-provider")}
 * system property and will then be used by geode to determine how much heap memory
 * is in use and what the maximum heap size is.
 * This interface provides the heap usage in two ways.
 * It must implement {@link #getBytesUsed()} which is used by geode to directly
 * ask this provider for the current used memory.
 * It can also provide notifications when the usage has changed by calling
 * {@link LongConsumer#accept(long)} on the listener passed to startNotifications.
 * It should only do this after startNotifications has been called and should stop notifications after stopNotifications has been called.
 * Notifications are optional. If the underlying JVM does not support notifications
 * then startNotifications and stopNotifications can have empty implementations.
 */
public interface HeapUsageProvider {
    /**
     * This is the name of a system property but this String must be prefixed with either
     * "geode." or "gemfire.".
     * If set then its value must be the name of a class that
     * implements the {@link HeapUsageProvider} interface and has a zero-arg public constructor.
     * If the class can not be found or an instance can not be constructed than the Cache startup
     * will fail with an exception.
     * Otherwise, a single instance will be created and used by Geode’s ResourceManager
     * to determine if the eviction-heap-threshold or the critical-heap-threshold are exceeded.
     * This single instance will be used until the Geode Cache is closed.
     * @constant string HEAP_USAGE_PROVIDER_CLASS_NAME = "resource-manager.heap-usage-provider.class-name"
     */
    String HEAP_USAGE_PROVIDER_CLASS_NAME = "resource-manager.heap-usage-provider.class-name";
    /**
     * Called by geode when this provider should start providing notifications
     * to the given listener.
     */
    void startNotifications(LongConsumer listener);
    /**
     * Called by geode when this provider should stop providing notifications.
     */
    void stopNotifications();
    /**
     * Called by geode when it wants to know what the maximum amount of heap is.
     * Geode does not expect this value to change, so it tends to call it once and
     * cache the returned value. Provider implementations must implement this method
     * to return a reasonable value.
     * @return the maximum amount of heap memory in bytes
     */
    long getMaxMemory();
    /**
     * Called by geode when it needs to know how many bytes are current used of heap memory.
     * Geode will call this method periodically (by default every 500 milliseconds which can
     * be changed using the "gemfire.heapPollerInterval" system property). Provider implementations must implement this method
     * to return a reasonable value.
     * @return the number of heap memory bytes currently in use
     */
    long getBytesUsed();
    /**
     * Called when this provider will no longer be used by Geode.
     */
    void close();
}

**Performance Impact**
No performance impact is expected.

**Backwards Compatibility and Upgrade Path**
No problem with rolling upgrades or backward compatibility.

It is worth noting that if the user specifies their own HeapUsageProvider then they will not be using the one that Geode uses by default. Part of the default HeapUsageProvider to check for the "gemfire.ResourceManager.HEAP_POOL" system property. So by specifying your own HeapUsageProvider you cause Geode to ignore the HEAP_POOL property.

**Prior Art**
The HEAP_POOL system property was Geode's previous attempt to allow customization of the ResourceManager's interaction with the JVM. The problem is it only allows the memory pool name to be customized. It does now allow customization of the APIs that will be called on that MemoryPoolMXBean.

FAQ

One thing that was considered was to use the standard jdk ServiceLoader to create an instance of this class. But since this solution only wants a single instance of HeapUsageProvider the service solution was going to require both the service definition and some way to specify which of the possible multiple service instances implementing HeapUsageProvider to use. So it seemed better to only allow a single class to be specified.

Errata

What are minor adjustments that had to be made to the proposal since it was approved?