| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/MemoryPoolMXBean.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/management/MemoryNotificationInfo.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/management/MemoryType.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/management/MemoryPoolMXBean.html)    [**NO FRAMES**](http://docs.google.com/MemoryPoolMXBean.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#2s8eyo1) | DETAIL: FIELD | CONSTR | [METHOD](#17dp8vu) |

## **java.lang.management**

Interface MemoryPoolMXBean

public interface **MemoryPoolMXBean**

The management interface for a memory pool. A memory pool represents the memory resource managed by the Java virtual machine and is managed by one or more [memory managers](http://docs.google.com/java/lang/management/MemoryManagerMXBean.html).

A Java virtual machine has one or more instances of the implementation class of this interface. An instance implementing this interface is an [MXBean](http://docs.google.com/ManagementFactory.html#MXBean) that can be obtained by calling the [ManagementFactory.getMemoryPoolMXBeans()](http://docs.google.com/java/lang/management/ManagementFactory.html#getMemoryPoolMXBeans()) method or from the [platform MBeanServer](http://docs.google.com/java/lang/management/ManagementFactory.html#getPlatformMBeanServer()) method.

The ObjectName for uniquely identifying the MXBean for a memory pool within an MBeanServer is:

[java.lang:type=MemoryPool](http://docs.google.com/java/lang/management/ManagementFactory.html#MEMORY_POOL_MXBEAN_DOMAIN_TYPE),name=*pool's name*

#### Memory Type

The Java virtual machine has a heap for object allocation and also maintains non-heap memory for the method area and the Java virtual machine execution. The Java virtual machine can have one or more memory pools. Each memory pool represents a memory area of one of the following types:

* [heap](http://docs.google.com/java/lang/management/MemoryType.html#HEAP)
* [non-heap](http://docs.google.com/java/lang/management/MemoryType.html#NON_HEAP)

#### Memory Usage Monitoring

A memory pool has the following attributes:

* [Memory usage](#_3znysh7)
* [Peak memory usage](#_2et92p0)
* [Usage Threshold](#_tyjcwt)
* [Collection Usage Threshold](#_4d34og8) (only supported by some *garbage-collected* memory pools)

#### 1. Memory Usage

The [getUsage()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsage()) method provides an estimate of the current usage of a memory pool. For a garbage-collected memory pool, the amount of used memory includes the memory occupied by all objects in the pool including both *reachable* and *unreachable* objects.

In general, this method is a lightweight operation for getting an approximate memory usage. For some memory pools, for example, when objects are not packed contiguously, this method may be an expensive operation that requires some computation to determine the current memory usage. An implementation should document when this is the case.

#### 2. Peak Memory Usage

The Java virtual machine maintains the peak memory usage of a memory pool since the virtual machine was started or the peak was reset. The peak memory usage is returned by the [getPeakUsage()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getPeakUsage()) method and reset by calling the [resetPeakUsage()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#resetPeakUsage()) method.

#### 3. Usage Threshold

Each memory pool has a manageable attribute called the *usage threshold* which has a default value supplied by the Java virtual machine. The default value is platform-dependent. The usage threshold can be set via the [setUsageThreshold](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#setUsageThreshold(long)) method. If the threshold is set to a positive value, the usage threshold crossing checking is enabled in this memory pool. If the usage threshold is set to zero, usage threshold crossing checking on this memory pool is disabled. The [isUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isUsageThresholdSupported()) method can be used to determine if this functionality is supported.

A Java virtual machine performs usage threshold crossing checking on a memory pool basis at its best appropriate time, typically, at garbage collection time. Each memory pool maintains a [usage threshold count](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsageThresholdCount()) that will get incremented every time when the Java virtual machine detects that the memory pool usage is crossing the threshold.

This manageable usage threshold attribute is designed for monitoring the increasing trend of memory usage with low overhead. Usage threshold may not be appropriate for some memory pools. For example, a generational garbage collector, a common garbage collection algorithm used in many Java virtual machine implementations, manages two or more generations segregating objects by age. Most of the objects are allocated in the *youngest generation* (say a nursery memory pool). The nursery memory pool is designed to be filled up and collecting the nursery memory pool will free most of its memory space since it is expected to contain mostly short-lived objects and mostly are unreachable at garbage collection time. In this case, it is more appropriate for the nursery memory pool not to support a usage threshold. In addition, if the cost of an object allocation in one memory pool is very low (for example, just atomic pointer exchange), the Java virtual machine would probably not support the usage threshold for that memory pool since the overhead in comparing the usage with the threshold is higher than the cost of object allocation.

The memory usage of the system can be monitored using [polling](#3dy6vkm) or [threshold notification](#1t3h5sf) mechanisms.

1. **Polling**  
   An application can continuously monitor its memory usage by calling either the [getUsage()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsage()) method for all memory pools or the [isUsageThresholdExceeded()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isUsageThresholdExceeded()) method for those memory pools that support a usage threshold. Below is example code that has a thread delicated for task distribution and processing. At every interval, it will determine if it should receive and process new tasks based on its memory usage. If the memory usage exceeds its usage threshold, it will redistribute all outstanding tasks to other VMs and stop receiving new tasks until the memory usage returns below its usage threshold.  
      
    // Assume the usage threshold is supported for this pool.  
    // Set the threshold to myThreshold above which no new tasks  
    // should be taken.  
    pool.setUsageThreshold(myThreshold);  
    ....  
     
    boolean lowMemory = false;  
    while (true) {  
    if (pool.isUsageThresholdExceeded()) {  
    // potential low memory, so redistribute tasks to other VMs  
    lowMemory = true;  
    redistributeTasks();  
    // stop receiving new tasks  
    stopReceivingTasks();  
    } else {  
    if (lowMemory) {  
    // resume receiving tasks  
    lowMemory = false;  
    resumeReceivingTasks();  
    }   
    // processing outstanding task  
    ...  
    }  
    // sleep for sometime  
    try {  
    Thread.sleep(sometime);  
    } catch (InterruptedException e) {  
    ...  
    }  
    }  
    The above example does not differentiate the case where the memory usage has temporarily dropped below the usage threshold from the case where the memory usage remains above the threshould between two iterations. The usage threshold count returned by the [getUsageThresholdCount()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsageThresholdCount()) method can be used to determine if the memory usage has returned below the threshold between two polls.  
   Below shows another example that takes some action if a memory pool is under low memory and ignores the memory usage changes during the action processing time.  
    // Assume the usage threshold is supported for this pool.  
    // Set the threshold to myThreshold which determines if   
    // the application will take some action under low memory condition.  
    pool.setUsageThreshold(myThreshold);  
     
    int prevCrossingCount = 0;  
    while (true) {  
    // A busy loop to detect when the memory usage   
    // has exceeded the threshold.  
    while (!pool.isUsageThresholdExceeded() ||   
    pool.getUsageThresholdCount() == prevCrossingCount) {  
    try {  
    Thread.sleep(sometime)  
    } catch (InterruptException e) {  
    ....  
    }  
    }  
     
    // Do some processing such as check for memory usage  
    // and issue a warning  
    ....  
     
    // Gets the current threshold count. The busy loop will then  
    // ignore any crossing of threshold happens during the processing.  
    prevCrossingCount = pool.getUsageThresholdCount();  
    }

1. **Usage Threshold Notifications**  
   Usage threshold notification will be emitted by [MemoryMXBean](http://docs.google.com/java/lang/management/MemoryMXBean.html). When the Java virtual machine detects that the memory usage of a memory pool has reached or exceeded the usage threshold the virtual machine will trigger the MemoryMXBean to emit an [usage threshold exceeded notification](http://docs.google.com/java/lang/management/MemoryNotificationInfo.html#MEMORY_THRESHOLD_EXCEEDED). Another usage threshold exceeded notification will not be generated until the usage has fallen below the threshold and then exceeded it again.  
   Below is an example code implementing the same logic as the first example above but using the usage threshold notification mechanism to detect low memory conditions instead of polling. In this example code, upon receiving notification, the notification listener notifies another thread to perform the actual action such as to redistribute outstanding tasks, stop receiving tasks, or resume receiving tasks. The handleNotification method should be designed to do a very minimal amount of work and return without delay to avoid causing delay in delivering subsequent notifications. Time-consuming actions should be performed by a separate thread. The notification listener may be invoked by multiple threads concurrently; so the tasks performed by the listener should be properly synchronized.  
      
    class MyListener implements javax.management.NotificationListener {  
    public void handleNotification(Notification notification, Object handback) {  
    String notifType = notification.getType();  
    if (notifType.equals(MemoryNotificationInfo.MEMORY\_THRESHOLD\_EXCEEDED)) {  
    // potential low memory, notify another thread  
    // to redistribute outstanding tasks to other VMs  
    // and stop receiving new tasks.  
    lowMemory = true;  
    notifyAnotherThread(lowMemory);  
    }  
    }  
    }  
     
    // Register MyListener with MemoryMXBean   
    MemoryMXBean mbean = ManagementFactory.getMemoryMXBean();  
    NotificationEmitter emitter = (NotificationEmitter) mbean;  
    MyListener listener = new MyListener();  
    emitter.addNotificationListener(listener, null, null);  
     
    // Assume this pool supports a usage threshold.  
    // Set the threshold to myThreshold above which no new tasks  
    // should be taken.  
    pool.setUsageThreshold(myThreshold);  
      
    // Usage threshold detection is enabled and notification will be   
    // handled by MyListener. Continue for other processing.  
    ....  
     
      
   There is no guarantee about when the MemoryMXBean will emit a threshold notification and when the notification will be delivered. When a notification listener is invoked, the memory usage of the memory pool may have crossed the usage threshold more than once. The [MemoryNotificationInfo.getCount()](http://docs.google.com/java/lang/management/MemoryNotificationInfo.html#getCount()) method returns the number of times that the memory usage has crossed the usage threshold at the point in time when the notification was constructed. It can be compared with the current usage threshold count returned by the [getUsageThresholdCount()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsageThresholdCount()) method to determine if such situation has occurred.

#### 4. Collection Usage Threshold

Collection usage threshold is a manageable attribute only applicable to some garbage-collected memory pools. After a Java virtual machine has expended effort in reclaiming memory space by recycling unused objects in a memory pool at garbage collection time, some number of bytes in the memory pools that are garbaged collected will still be in use. The collection usage threshold allows a value to be set for this number of bytes such that if the threshold is exceeded, a [collection usage threshold exceeded notification](http://docs.google.com/java/lang/management/MemoryNotificationInfo.html#MEMORY_THRESHOLD_EXCEEDED) will be emitted by the [MemoryMXBean](http://docs.google.com/java/lang/management/MemoryMXBean.html). In addition, the [collection usage threshold count](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getCollectionUsageThresholdCount()) will then be incremented.

The [isCollectionUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isCollectionUsageThresholdSupported()) method can be used to determine if this functionality is supported.

A Java virtual machine performs collection usage threshold checking on a memory pool basis. This checking is enabled if the collection usage threshold is set to a positive value. If the collection usage threshold is set to zero, this checking is disabled on this memory pool. Default value is zero. The Java virtual machine performs the collection usage threshold checking at garbage collection time.

Some garbage-collected memory pools may choose not to support the collection usage threshold. For example, a memory pool is only managed by a continuous concurrent garbage collector. Objects can be allocated in this memory pool by some thread while the unused objects are reclaimed by the concurrent garbage collector simultaneously. Unless there is a well-defined garbage collection time which is the best appropriate time to check the memory usage, the collection usage threshold should not be supported.

The collection usage threshold is designed for monitoring the memory usage after the Java virtual machine has expended effort in reclaiming memory space. The collection usage could also be monitored by the polling and threshold notification mechanism described above for the [usage threshold](#_tyjcwt) in a similar fashion.

**Since:** 1.5 **See Also:** [JMX Specification.](http://docs.google.com/javax/management/package-summary.html),  [Ways to Access MXBeans](http://docs.google.com/package-summary.html#examples)

| **Method Summary** | |
| --- | --- |
| [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) | [**getCollectionUsage**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getCollectionUsage())()            Returns the memory usage after the Java virtual machine most recently expended effort in recycling unused objects in this memory pool. |
| long | [**getCollectionUsageThreshold**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getCollectionUsageThreshold())()            Returns the collection usage threshold value of this memory pool in bytes. |
| long | [**getCollectionUsageThresholdCount**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getCollectionUsageThresholdCount())()            Returns the number of times that the Java virtual machine has detected that the memory usage has reached or exceeded the collection usage threshold. |
| [String](http://docs.google.com/java/lang/String.html)[] | [**getMemoryManagerNames**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getMemoryManagerNames())()            Returns the name of memory managers that manages this memory pool. |
| [String](http://docs.google.com/java/lang/String.html) | [**getName**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getName())()            Returns the name representing this memory pool. |
| [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) | [**getPeakUsage**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getPeakUsage())()            Returns the peak memory usage of this memory pool since the Java virtual machine was started or since the peak was reset. |
| [MemoryType](http://docs.google.com/java/lang/management/MemoryType.html) | [**getType**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getType())()            Returns the type of this memory pool. |
| [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) | [**getUsage**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsage())()            Returns an estimate of the memory usage of this memory pool. |
| long | [**getUsageThreshold**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsageThreshold())()            Returns the usage threshold value of this memory pool in bytes. |
| long | [**getUsageThresholdCount**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#getUsageThresholdCount())()            Returns the number of times that the memory usage has crossed the usage threshold. |
| boolean | [**isCollectionUsageThresholdExceeded**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isCollectionUsageThresholdExceeded())()            Tests if the memory usage of this memory pool after the most recent collection on which the Java virtual machine has expended effort has reached or exceeded its collection usage threshold. |
| boolean | [**isCollectionUsageThresholdSupported**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isCollectionUsageThresholdSupported())()            Tests if this memory pool supports a collection usage threshold. |
| boolean | [**isUsageThresholdExceeded**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isUsageThresholdExceeded())()            Tests if the memory usage of this memory pool reaches or exceeds its usage threshold value. |
| boolean | [**isUsageThresholdSupported**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isUsageThresholdSupported())()            Tests if this memory pool supports usage threshold. |
| boolean | [**isValid**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isValid())()            Tests if this memory pool is valid in the Java virtual machine. |
| void | [**resetPeakUsage**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#resetPeakUsage())()            Resets the peak memory usage statistic of this memory pool to the current memory usage. |
| void | [**setCollectionUsageThreshold**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#setCollectionUsageThreshold(long))(long threhsold)            Sets the collection usage threshold of this memory pool to the given threshold value. |
| void | [**setUsageThreshold**](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#setUsageThreshold(long))(long threshold)            Sets the threshold of this memory pool to the given threshold value if this memory pool supports the usage threshold. |

| **Method Detail** |
| --- |

### getName

[String](http://docs.google.com/java/lang/String.html) **getName**()

Returns the name representing this memory pool.

**Returns:**the name of this memory pool.

### getType

[MemoryType](http://docs.google.com/java/lang/management/MemoryType.html) **getType**()

Returns the type of this memory pool.

**MBeanServer access**:

The mapped type of MemoryType is String and the value is the name of the MemoryType.

**Returns:**the type of this memory pool.

### getUsage

[MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) **getUsage**()

Returns an estimate of the memory usage of this memory pool. This method returns null if this memory pool is not valid (i.e. no longer exists).

This method requests the Java virtual machine to make a best-effort estimate of the current memory usage of this memory pool. For some memory pools, this method may be an expensive operation that requires some computation to determine the estimate. An implementation should document when this is the case.

This method is designed for use in monitoring system memory usage and detecting low memory condition.

**MBeanServer access**:

The mapped type of MemoryUsage is CompositeData with attributes as specified in [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html#from(javax.management.openmbean.CompositeData)).

**Returns:**a [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) object; or null if this pool not valid.

### getPeakUsage

[MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) **getPeakUsage**()

Returns the peak memory usage of this memory pool since the Java virtual machine was started or since the peak was reset. This method returns null if this memory pool is not valid (i.e. no longer exists).

**MBeanServer access**:

The mapped type of MemoryUsage is CompositeData with attributes as specified in [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html#from(javax.management.openmbean.CompositeData)).

**Returns:**a [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) object representing the peak memory usage; or null if this pool is not valid.

### resetPeakUsage

void **resetPeakUsage**()

Resets the peak memory usage statistic of this memory pool to the current memory usage.

**Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a security manager exists and the caller does not have ManagementPermission("control").

### isValid

boolean **isValid**()

Tests if this memory pool is valid in the Java virtual machine. A memory pool becomes invalid once the Java virtual machine removes it from the memory system.

**Returns:**true if the memory pool is valid in the running Java virtual machine; false otherwise.

### getMemoryManagerNames

[String](http://docs.google.com/java/lang/String.html)[] **getMemoryManagerNames**()

Returns the name of memory managers that manages this memory pool. Each memory pool will be managed by at least one memory manager.

**Returns:**an array of String objects, each is the name of a memory manager managing this memory pool.

### getUsageThreshold

long **getUsageThreshold**()

Returns the usage threshold value of this memory pool in bytes. Each memory pool has a platform-dependent default threshold value. The current usage threshold can be changed via the [setUsageThreshold](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#setUsageThreshold(long)) method.

**Returns:**the usage threshold value of this memory pool in bytes. **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a usage threshold.**See Also:**[isUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isUsageThresholdSupported())

### setUsageThreshold

void **setUsageThreshold**(long threshold)

Sets the threshold of this memory pool to the given threshold value if this memory pool supports the usage threshold. The usage threshold crossing checking is enabled in this memory pool if the threshold is set to a positive value. The usage threshold crossing checking is disabled if it is set to zero.

**Parameters:**threshold - the new threshold value in bytes. Must be non-negative. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if threshold is negative or greater than the maximum amount of memory for this memory pool if defined. [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a usage threshold. [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a security manager exists and the caller does not have ManagementPermission("control").**See Also:**[isUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isUsageThresholdSupported()), [Usage threshold](#_tyjcwt)

### isUsageThresholdExceeded

boolean **isUsageThresholdExceeded**()

Tests if the memory usage of this memory pool reaches or exceeds its usage threshold value.

**Returns:**true if the memory usage of this memory pool reaches or exceeds the threshold value; false otherwise. **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a usage threshold.

### getUsageThresholdCount

long **getUsageThresholdCount**()

Returns the number of times that the memory usage has crossed the usage threshold.

**Returns:**the number of times that the memory usage has crossed its usage threshold value. **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a usage threshold.

### isUsageThresholdSupported

boolean **isUsageThresholdSupported**()

Tests if this memory pool supports usage threshold.

**Returns:**true if this memory pool supports usage threshold; false otherwise.

### getCollectionUsageThreshold

long **getCollectionUsageThreshold**()

Returns the collection usage threshold value of this memory pool in bytes. The default value is zero. The collection usage threshold can be changed via the [setCollectionUsageThreshold](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#setCollectionUsageThreshold(long)) method.

**Returns:**the collection usage threshold of this memory pool in bytes. **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a collection usage threshold.**See Also:**[isCollectionUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isCollectionUsageThresholdSupported())

### setCollectionUsageThreshold

void **setCollectionUsageThreshold**(long threhsold)

Sets the collection usage threshold of this memory pool to the given threshold value. When this threshold is set to positive, the Java virtual machine will check the memory usage at its best appropriate time after it has expended effort in recycling unused objects in this memory pool.

The collection usage threshold crossing checking is enabled in this memory pool if the threshold is set to a positive value. The collection usage threshold crossing checking is disabled if it is set to zero.

**Parameters:**threhsold - the new collection usage threshold value in bytes. Must be non-negative. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if threshold is negative or greater than the maximum amount of memory for this memory pool if defined. [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a collection usage threshold. [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a security manager exists and the caller does not have ManagementPermission("control").**See Also:**[isCollectionUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isCollectionUsageThresholdSupported()), [Collection usage threshold](#_4d34og8)

### isCollectionUsageThresholdExceeded

boolean **isCollectionUsageThresholdExceeded**()

Tests if the memory usage of this memory pool after the most recent collection on which the Java virtual machine has expended effort has reached or exceeded its collection usage threshold. This method does not request the Java virtual machine to perform any garbage collection other than its normal automatic memory management.

**Returns:**true if the memory usage of this memory pool reaches or exceeds the collection usage threshold value in the most recent collection; false otherwise. **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a usage threshold.

### getCollectionUsageThresholdCount

long **getCollectionUsageThresholdCount**()

Returns the number of times that the Java virtual machine has detected that the memory usage has reached or exceeded the collection usage threshold.

**Returns:**the number of times that the memory usage has reached or exceeded the collection usage threshold. **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this memory pool does not support a collection usage threshold.**See Also:**[isCollectionUsageThresholdSupported()](http://docs.google.com/java/lang/management/MemoryPoolMXBean.html#isCollectionUsageThresholdSupported())

### getCollectionUsage

[MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) **getCollectionUsage**()

Returns the memory usage after the Java virtual machine most recently expended effort in recycling unused objects in this memory pool. This method does not request the Java virtual machine to perform any garbage collection other than its normal automatic memory management. This method returns null if the Java virtual machine does not support this method.

**MBeanServer access**:

The mapped type of MemoryUsage is CompositeData with attributes as specified in [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html#from(javax.management.openmbean.CompositeData)).

**Returns:**a [MemoryUsage](http://docs.google.com/java/lang/management/MemoryUsage.html) representing the memory usage of this memory pool after the Java virtual machine most recently expended effort in recycling unused objects; null if this method is not supported.

### isCollectionUsageThresholdSupported

boolean **isCollectionUsageThresholdSupported**()

Tests if this memory pool supports a collection usage threshold.

**Returns:**true if this memory pool supports the collection usage threshold; false otherwise.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/MemoryPoolMXBean.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/management/MemoryNotificationInfo.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/management/MemoryType.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/management/MemoryPoolMXBean.html)    [**NO FRAMES**](http://docs.google.com/MemoryPoolMXBean.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#2s8eyo1) | DETAIL: FIELD | CONSTR | [METHOD](#17dp8vu) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).