| Print | Contents |

Garbage collection and finalization in Java

Abstract

This article outlines how Java's garbage collection system works and how to explicitly make objects available for collection by using the finalize method.

Garbage collection

In Java, the runtime system performs memory management tasks for you, using a facility known as garbage collection. The Java garbage collector is a daemon thread – a thread that runs for the benefit of other threads – that automatically reclaims dynamically allocated memory that's no longer needed. A thread is a form of process within a program. A program may contain more than one thread running concurrently. For example, one thread might be doing a calculation while another is searching through a database.

The Java garbage collector runs as a low-priority thread, waiting for higher-priority threads to relinquish the processor.

Garbage collection algorithms

Java, in versions before 1.3, used a "mark & sweep" garbage collection algorithm. In the first phase, it tracks through all the object references, marking any objects that are referred to. In a second phase, it collects up any objects that were left unmarked. This default algorithm was not suitable for all applications. Java, since version 1.3, now has several different algorithms available, which have been greatly expanded in version 1.4.1 to include the "copying collector" algorithm, which stops all application threads until the garbage collection is complete. This algorithm uses one thread to complete the collection. Other algorithms include the "parallel copying collector" that stops all application threads but uses multiple threads to complete the collection. Another algorithm is the "parallel scavenge collector" – this too stops all application threads but is used for gigabyte heaps.

Controlling garbage collection

If you want to ensure that an object is garbage collected, you can set its object reference to `null`, as in the following:

```
myObject theObject = new myObject();
theObject = null;
```

The object can then be collected the next time the garbage collector runs, as long as no other references to it exist in other code. In fact, the garbage collector is guaranteed to only ever run for objects with no references.

The garbage collector runs synchronously – at set, regular intervals – or asynchronously –

at irregular intervals – depending on the situation and the system on which you are running Java. The garbage collector runs synchronously when the system runs out of memory or if you request it to do so. You can try to invoke the garbage collector at any time by calling the System class's `gc` method:

```
System.gc();
```

The above statement schedules the garbage collector to execute as soon as it can. However, asking garbage collector to run in this way does not guarantee that objects will be collected immediately.

Although the garbage collector runs by default when the memory allocation runs out, it is still implementation-dependent.

You can prevent the garbage collector ever running by using the `java -noasyncgc` command to run an application. However, this should only be used with caution and only in a test environment, as it is bound to cause your application to use up all available memory resources eventually.

Finalization

Each class in Java can have a `finalize` method that helps return resources to the system. In effect, finalization means that before an object is garbage collected, the Java runtime system gives the object a chance to clean up after itself. The code of a typical `finalize` method is as follows:

```
public void finalize() throws Throwable {
    firstName = null ;
    lastName = null ;
    super.finalize() ;
}
```

Java calls the `finalize` method at some time after the system has determined that an object is a candidate for return and before the object is garbage collected. This method can be used to free up any non-memory resources used by the object, for instance, socket connections or open files. You should note that the `finalize` method in Java is completely different to the deceptively similar destructor method in C++. The `finalize` method is never guaranteed to run in Java because objects may not be garbage collected. For essential destruction behavior, you should use an explicit delete method.

Any `finalize` method that you create must override the empty `finalize` method provided by the parent `Object` class, so it must

- be called `finalize`
- have no parameters
- return `void`
- have a `throws Throwable` clause

A class can have only one `finalize` method – finalizer overloading is not permitted in Java and simple classes usually have none.

You can force object finalization by calling the `runFinalization` method in Java's System class as follows:

```
System.runFinalization() ;
```

This method frees up system resources by calling the `finalize` methods for all objects awaiting garbage collection.

The `finalize` method will only be called once, if called at all. The `finalize` method should be written to call its superclasses' `finalize` method explicitly, using the call `super.finalize()`. Although the `finalize` method is a legitimate Java tool, you cannot rely on this process to free resources at a particular time because it is dependent on the garbage collection procedure, which itself is not predictable. You should use `finalize()` only when its failure to execute predictably will not cause problems for your application.

Summary

The Java garbage collector is a daemon thread – a thread that runs for the benefit of other threads. It is a mark-sweep facility that scans Java's dynamic memory areas for objects, marking those that are referenced. When Java determines that there are no longer any references to an object, it marks the object for eventual garbage collection. The "mark and sweep" garbage collection algorithm is not suitable for all applications. The new version of Java has been expanded to contain several new garbage collection algorithms. These include the "copying collector", the "parallel copying collector" and the "parallel scavenge collector". All these new algorithms stop all application threads until the garbage collection is complete.

Java's automatic garbage collector eliminates many of the memory leaks that can occur in C and C++. It runs as a low-priority thread, waiting for higher-priority threads to relinquish the processor. You can use a `finalize` method in a class to help return resources to the system. The `finalize` method runs automatically when the system runs out of memory or when runtime ends, but you can also invoke it at other times using the `System.runFinalization` method.

Table of Contents