DZone

Java Memory Leaks: Tools, Fixes, and More

by Angela Stringfellow MVB · Aug. 14, 17 · Java Zone · Tutorial

Memory management is Java's strongest suit, and one of the many reasons why developers choose Java over other platforms and programming languages. On paper, you create objects and Java would deploy its Garbage Collector to allocate and free up memory. But that's not to say Java is flawless; in real life, memory leaks do happen, and it happens a lot in Java applications. **We put together this guide to arm you with the know-how to detect, avoid, and fix memory leaks in Java.**

Should You Worry About Memory Leaks?

Memory leaks often involve small amounts of memory resources, and you would probably not have problems with it. But when your application returns a java.lang.OutOfMemoryError, then your first and most likely suspect would be a memory leak.

Memory leaks are often an indicator of badly written programs, and if you are the type of programmer that wants everything to be perfect, then you should investigate any memory leak that occurs. As a Java programmer, there is no way for you to know when Java virtual machine would run the garbage collector. This is true, even if you specify System.gc(). The garbage collector will most probably run when memory runs low, or when the available memory is lesser than what your program needs. If the garbage collector does not free up enough memory resources, your program will get memory from your operating system.

A Java memory leak is not always serious as compared to memory leaks that happen in C++ and other programming languages. According to Jim Patrick of IBM Developer Works, for the most part, you should be concerned with a memory leak considering two factors: the size of the leak and the program's lifetime.

A small Java application might have a memory leak, but if the JVM will have enough memory to run your program, then it will not really matter. But if your Java application runs constantly, then memory leaks would be a problem. A continuously running program will eventually run out of memory resources.

Another area where memory leaks might be a problem is when the program calls for a lot of temporary objects that use up large amounts of memory. When these memory-hogging objects are not de-referenced, then the program would soon have an available memory that is lower than what it needs.

How to Avoid Java Memory Leaks

To avoid memory leaks, you need to pay attention to how you write your code. Here are specific methods to help you stamp out memory leaks.

1. Use Reference Objects to Avoid Memory Leaks

Raimond Reichert at JavaWorld writes that you can use reference objects to get rid of memory leaks.

Using the java.lang.ref package, you can work with the garbage collector in your program. This allows you to avoid directly referencing objects, but use special reference objects that are easily cleared by the garbage collector. The special subclasses allow you to refer to objects indirectly. For instance, Reference has three subclasses: PhantomReference, SoftReference, and WeakReference.

A referent, or an object that is referenced by these subclasses, can be accessed using that reference object's get method. The advantage of using this method is that you can clear a reference easily by setting it to null and that the reference is pretty much immutable, or it cannot be changed. How does garbage collector act with each type of referent?

- SoftReference object: Garbage collector is required to clear all SoftReference objects when memory runs low.
- WeakReference object: When garbage collector senses a weakly referenced object, all references to it are cleared and ultimately taken out of memory.
- PhantomReference object: Garbage collector would not be able to automatically clean up PhantomReference objects, you would need to clean it up manually by clearing all references to it.

Using reference objects, you can work with the garbage collector to automate the task of removing listeners that are weakly reachable. WeakReference objects, especially with a cleanup thread, can help you avoid memory errors.

2. Avoid Memory Leaks Related to a WebApp Classloader

If you are using Jetty 7.6.6. or higher, you can prevent WebApp classloader pinning. When your code keeps referring to a webapp classloader, memory leaks can easily happen. There are two types of leaks in this case: **daemon threads** and **static fields**.

- **Static fields** are started with the classloader's value. Even as Jetty stops deploying and then redeploys your webapp, the static reference persists and so the object cannot be cleared from memory.
- **Daemon threads** that are started outside the lifecycle of a Web application are prone to memory leaks because these threads have references to the classloader that started the threads.

With Jetty, you can use preventers to help you address problems associated with WebApp classloaders. For instance, app context leak preventer, such as appcontext.getappcontext() helps you keep the static references within the context classloader. Other preventers you can use are the following:

- AWT leak preventer
- DOM leak preventer
- Driver manager leak preventer
- GC thread leak preventer

- Java2D leak preventer
- LDAP leak preventer
- Login configuration leak preventer
- Security provider leak preventer

3. Other Specific Steps

BurnIgnorance.com also lists several ways to prevent memory leaks in Java, including:

- Release the session when it is no longer needed. Use the HttpSession.invalidate() to do this.
- Keep the time-out time low for each session.
- Store only the necessary data in your HttpSession.
- Avoid using string concatenation. Use StringBuffer's append() method because the string is an unchangeable object while string concatenation creates a lot of unnecessary objects. A large number of temporary objects will slow down performance.
- As much as possible, you **should not** create HttpSession in your jsp page. You can do this by using the page directive <%@page session="false"%>.
- If you are writing a query that is frequently executed, use PreparedStatement object rather than using Statement object. Why? PreparedStatement is precompiled while Statement is compiled every time your SQL statement is transmitted to the database.
- When using JDBC code, avoid using "*" when you write your query. Try to use the corresponding column name instead.
- If you are going to use stmt = con.prepareStatement(sql query) within a loop, then be sure to close it inside that particular loop.
- Be sure to close the Statement and ResultSet when you need to reuse these.
- Close the ResultSet, Connection, PreparedStatement, and Statement in the finally block.

What to Do When You Suspect Memory Leaks

If you find it takes longer to execute your application or if you notice a considerable slowdown, then it is time to check for memory leaks.

How do you know your program has a memory leak? A very common telltale sign is the java.lang.OutOfMemoryError error. This error has several detail messages that would allow you to determine if there is a memory leak or not:

- Java heap space: Means that memory resources could not be allocated for a particular object in the Java heap. This can mean several things, including a memory leak, or the specified heap size is lower than what the application needs. It could also mean that your program is using a lot of finalizers.
- PermGen space: This means that the permanent generation area is already full. This area is where the method

- PermGen space: This means that the permanent generation area is already full. This area is where the method and class objects are stored. You can easily correct this by increasing the space via -XX:MaxPermSize.
- Requested array size exceeds VM limit: This means that the program is trying to assign an array that is > than the heap size.
- Request <size> bytes for <reason>. Out of swap space?: This means that an allocation using the local heap did not succeed, or the native heap is close to being all used up.
- <Reason> <stack trace> (Native method): This means that a native method was not allocated the required memory.

There are times when your application just crashes without returning an OutOfMemoryError message. When it does crash before giving you an error message, it makes it more difficult to diagnose and correct memory leaks. The good news is that you can check the fatal log error or the crash dump to see what went wrong.

Moreover, there are a lot of monitoring and diagnostic tools that you can use to help you identify and correct memory leaks. Stackify's Daryn Howard has identified Java profilers as a good way to track down memory leaks and be able to run the garbage collector manually. You can use Java profilers to review how memory is being used that can easily show you which processes and classes are using too much memory when they should not be. You can also use JVM Performance Metrics, which gives you tons of data on garbage collection, thread counts, and memory usage.

A quick word about Java profilers. Java profiling helps you monitor different JVM parameters including object creation, thread execution, method execution, and, yes, garbage collection.

There are times when you have ruled out memory leaks as the reason for your application's slow down, so you can use Java profiling tools to get a closer view of how your application is utilizing memory and other resources. Instead of going over your code to find the problems, you can just use these tools to do just that. It saves you the effort and the hours needed to ensure that your code is up to par.

These tools give you a comprehensive set of statistics and other information that you can use to trace your coding mistakes. They can also help you find what are really causing performance slowdown, multi-threading problems, and memory leaks. In short, they give you a more stable and scalable application. And the best part – Java profiling tools can give you a fine-grained analysis of every problem and how to solve them.

If you use these tools early into your project, and you use them regularly – particularly when you use them in conjunction with other Java performance tools – you can create efficient, high-performing, fast, and stable applications. It can also help you know critical issues before you deploy your app.

Some of the metrics you can find out using Java profiling tools include:

- A method's CPU time
- Memory utilization
- Information on method calls
- What objects are created
- What objects are removed from memory or garbage collected

You also have:

Memory Analyzer (MAT): It allows you to analyze Java heap to search for memory look and lower memory use. You can easily analyze heap dumps even when there are millions of objects living in them, and see the sizes of each object and why Garbage Collector is not deleting it from memory. You get a nifty report on these objects, helping you narrow down suspected memory leaks.

Java Flight Recorder: The Java Flight Recorder is basically a diagnostic and profiling tool that gives you more information about a running application. It gives you better data than those provided by other tools, allows APIs created by third party services, and lowers your total cost of ownership. A commercial feature of Oracle Java SE, Java Flight Recorder gives you an easy way to detect memory leaks, find the classes responsible for these leaks, and locate the leak in order to correct it.

Other tools you should know include:

- **NetBeans Profiler**: Can support Java SE, Java FX, EJB, mobile applications, and Web applications, and could be used to monitor memory, threads, and CPU resources.
- **JProfiler**: A thread, memory, and CPU profiling tool that can also be used to analyze memory leaks and other performance bottlenecks.
- **GC Viewer**: An open-source tool that allows you to easily visualize information produced by JVM. You can use this to see performance metrics related to garbage collection, including accumulated pauses, longest pauses, and throughput. Aside from enabling you to run garbage collection, you can also use this tool to set up the preliminary heap size.
- **VisualVM**: Based on the NetBeans platform, VisualVM is an easily extensible tool using a variety of plugins. You get detailed data on your applications, and these data can be used to monitor both remote and local apps. You can get memory profiling and manually run the garbage collector using this tool.
- **Patty in action**: Another open source tool that you can use as a profiling tool that can give you target and drilled down profiling. You can use this tool to analyze heaps.
- **JRockit**: A proprietary solution from Oracle, JRockit is for Java SE applications that may be used to predict latency and allows you to visualize garbage collection and sort through memory-related issues.
- **GCeasy**: GCeasy is a tool that analyzes logs related to garbage collection. It is an easy way to detect memory leak problems as it goes through and analyze garbage collection logs. Another reason to use GCeasy is that it is available online; there is no need for you to install it on your machine to use it.

Java Memory Leaks: Solutions

Now that you know your program has memory leaks, you can use these tools to help fix these leaks when they become a problem, or even before they become an issue.

Using Tools That Can Detect Memory Leaks

For this example, we are going to use VisualVM, but other tools described above may also be utilized.

Once you have downloaded and configured VisualVM, you can analyze your code by running your application with VisualVM attached to it. Then perform the task that slows down your application and looks at the “monitor” and “memory pools” tabs. What do you need to look out for? See spikes in memory usage in the Monitor tab. When this happens, press on the “Perform GC” button, which will activate garbage collection. This should help decrease the amount of memory used.

If that does not work, switch to “memory pools” and look at the Old Gen section. If there are objects leaking, you would see it here. Remember that active objects are placed in “Eden” and will then be moved to “Survivor.” Meanwhile, older objects are found in the ‘Old Gen’ pool.

At this point, you can go back to your code and comment out the irrelevant parts, up to the point where you notice that there is performance slow down or where it just stops. Repeat all these steps until you have stopped all the leaks.

Enable some parts of your code to check memory usage, and if you find another leak, it is time to get into the method that caused these leaks to help plug it. Keep on narrowing it down until you only have a single class or method left. Validate all file buffers to see if these are closed. Also, check all hashmaps to see if you are using these properly.

Using Heap Dumps

If you find the above-mentioned method too tedious, you might be able to reduce the time you spend on fixing memory leaks by using heap dumps. Heap dumps allow you to see the number of instances open and how much space these instances take up. If there is a specific instance that you want to investigate further, you can just double click on that particular instance and see more information. Heap dumps are useful in knowing just how many objects are generated by your application.

Using Eclipse Memory Leak Warnings

Another way to save time is to rely on Eclipse memory leak warnings. If you have a code that is compliant with JDK 1.5 or higher, you can use Eclipse to warn you when a reference is ended but the object persists and is not closed. Just be sure to enable leak detection in your project settings. This might not be a comprehensive solution, though. This is because Eclipse does not detect all leaks and may miss some file closures, especially when you have code that is not JDK 1.5 (or higher) compliant. Another reason why Eclipse does not always work is that these file closures and openings are nested very deeply.

Additional Resources and Tutorials

Get more insights and information on how to avoid, detect, and rectify memory leaks from the following resources and tutorials:

- How to find and fix memory leaks in your Java application
- Hunting Memory Leaks in Java
- Java Tip 79: Interact with garbage collector to avoid memory leaks
- Preventing Memory Leaks

- How to create/avoid memory leak in Java and .NET?
- What you need to know about Android app memory leaks

Memory leaks are certainly a concern for Java developers, but they're not always the end of the world. Arm yourself with the know-how to prevent them before they occur and address them when they arise.

Topics: JAVA , MEMORY LEAKS , JAVA PERFORMANCE , JAVA TOOLS , TUTORIAL