



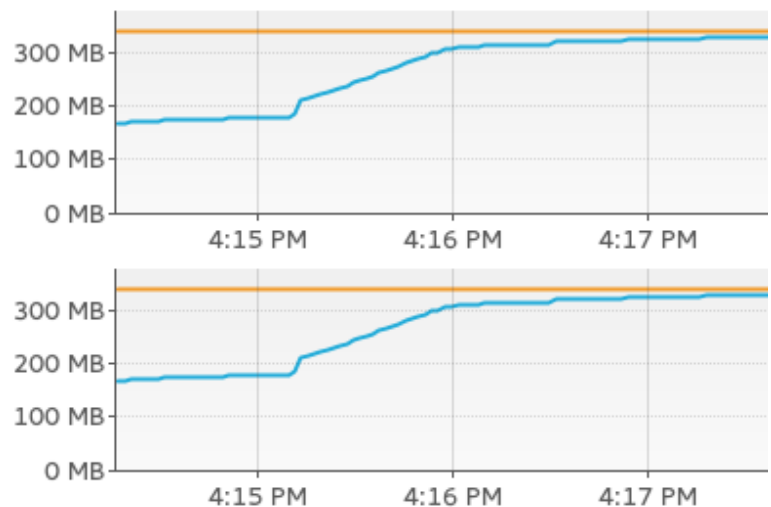
[Nueva guía] Descargue la Guía 2017 de Microservicios: Rom...

[Descargar la guía](#) ▶

# Cómo encontrar y corregir fugas de memoria en su aplicación Java

por Leo Ufimtsev MVB · 12 y 17 de diciembre · Zona de Java

Descargue *Microservices for Java Developers* : una introducción práctica a frameworks y contenedores. Presentado en asociación con Red Hat .



¿Tiene una aplicación Java que funciona bien al principio, pero se ralentiza después de un tiempo, o funciona bien para una pequeña cantidad de archivos, pero el rendimiento se degrada para una gran cantidad de archivos? Tal vez tengas una pérdida de memoria.

Al corregir fugas de memoria, si alguien me pregunta: "Si supieras lo que sabes ahora, ¿qué te dirías a ti mismo?" Bueno, yo diría ...

## Público objetivo

Si bien, en general, el enfoque descrito en este artículo

es independiente de IDE & OS, utilicé Linux [Fedora] y Eclipse [Desarrollo de complementos] en las capturas de pantalla e instrucciones.

## Síntomas de una pérdida de memoria

Funciona rápido al principio, pero se ralentiza con el tiempo.

- Funciona bien con pequeños conjuntos de datos, graves problemas de rendimiento con grandes conjuntos de datos
- Aumento constante del uso de la memoria de la generación anterior en tu JVM
- Errores de montón fuera de memoria en su JVM
- Accidentes espontáneos.

## Fugas de memoria comunes

Una pérdida de memoria en Java (¿quién lo hubiera pensado?) Puede ocurrir si se olvida de cerrar un recurso, o si no se lanza una referencia a un objeto. p.ej

- Los búferes de archivo / texto no están cerrados. (como fue en mi caso )
- Los mapas Hash mantienen las referencias activas si equals() and hashCode() no están implementadas, por ejemplo

```
1
2
3  importar java . util . Mapa ;
4  clase pública MemLeak {
5  clave de cadena pública final ;
6  MemLeak público ( clave de cadena ) {
7      esta . llave = llave ;
8  }
9
10 public static void main ( String args []
11     prueba {
12         Mapa del mapa = Sistema . getProper
13         para (;;) {
```

```

14  mapa . put ( nuevo memLeak ( y amp
15  }
16  } catch ( Exception e ) {
17      e . printStackTrace ();
18  }
19  }
20  }

```

- detalles ...
- Las clases internas que hacen referencia a las clases externas pueden tener fugas. (hacerlos estáticos para evitar).

## ¿Cómo se solucionan?

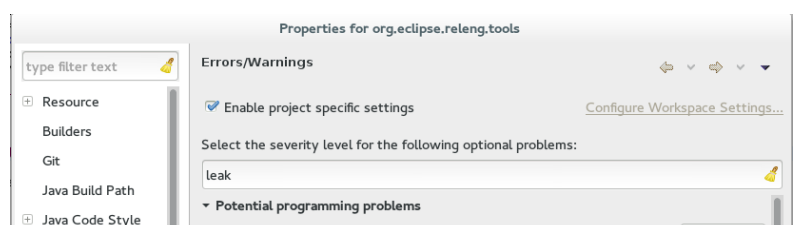
Hay dos enfoques. El primero es un intento de 'solución rápida'. Si eso falla, entonces tendrás que recorrer el largo camino.

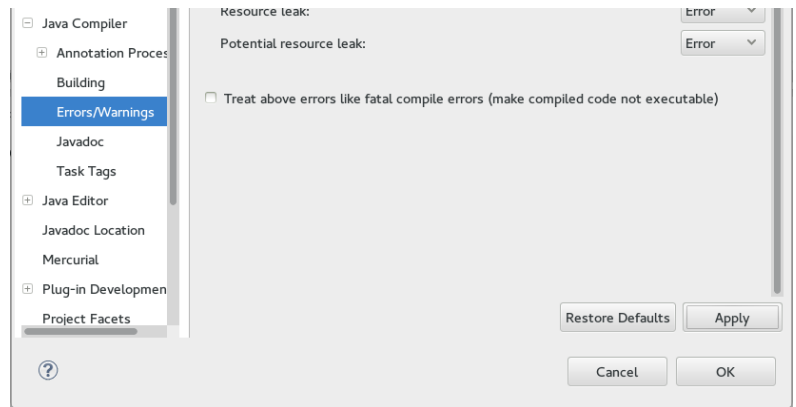
1) Solución rápida: advertencias de pérdida de memoria Eclipse (capta algunas fugas)

2) Deshabilite y habilite manualmente partes de su código y observe el uso de memoria de su JVM usando una herramienta JVM como VisualVM (o Jconsole, o Thermostat).

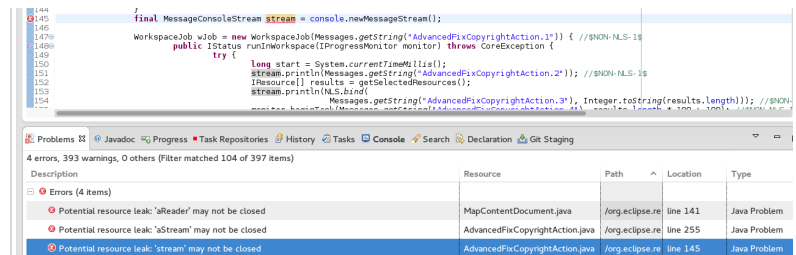
### 1) Solución rápida: advertencia de Eclipse Memory Leak / errores.

Para el código con cumplimiento JDK 1.5+, Eclipse lanzará advertencias y errores para casos obvios de fugas. Para ser más precisos, cualquier elemento que se pueda cerrar (desde 1.5 - por ejemplo, outputstream desde 1.5) lanzará una advertencia sobre usted si su referencia se destruye pero el objeto no está cerrado. Sin embargo, la detección de fugas no siempre está habilitada en los proyectos de Eclipse. Es posible que tengas que encenderlos primero. Vaya a la configuración de su proyecto y habilítelos como se demostró:





Ahora Eclipse describirá las pérdidas de memoria:



Sin embargo, incluso con Eclipse hocus pocus, no se detectan todos los cierres de archivos y fugas. Especialmente cuando se trabaja con código heredado (pre 1.5), es probable que se encuentre con filtraciones porque se escribieron antes de que se implementara el término "cerrable". O a veces las aperturas / cierres de archivos se anidan tan profundamente que el eclipse no los detectará. Si estás en este lugar, es posible que quieras probar el paso 2.

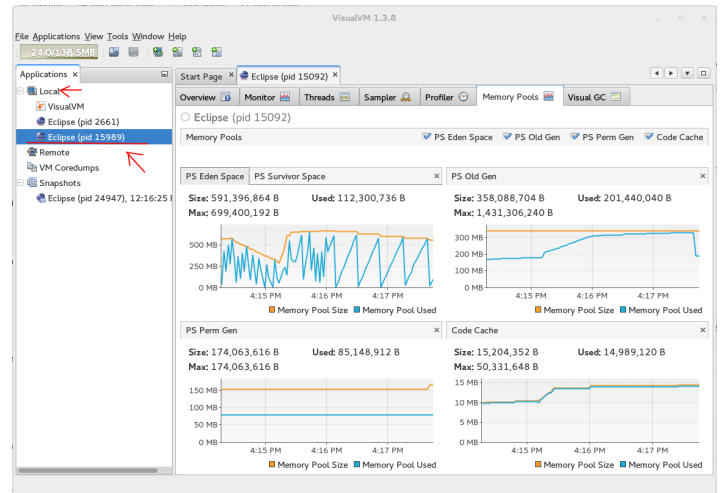
## 2) Deshabilite y habilite manualmente partes de su código y observe el uso de la memoria de su JVM usando una herramienta JVM como VisualVM.

Si llegaste tan lejos, entonces tendrás que arremangarte y hacer un poco de trabajo manual. Puede leer todo su código e intentar comprender dónde ocurre la fuga. Para ayudarlo con este proceso: Recomendando probar usando una herramienta como VisualVM . (Pero el termostato y MAT también funcionan).

### Configurar VisualVM

1. Descargar la herramienta
2. Abra la terminal, vaya a ... / visualvm\_xyz / bin ejecute el script de shell './visualvm' (o visualvm.exe en Windows).
3. Deberías ver la ventana principal. Si expandes

'local' y haces doble clic en tu aplicación en ejecución (un eclipse infantil en mi caso), puedes ver sus propiedades.



**4. Solución de problemas de VisualVM en Fedora (omite esto si funciona bien:** para mí, inicialmente no podía conectarme a mi JVM, no podía tomar volcados de memoria y los perfiles tampoco funcionaban. Estos son algunos pasos que pueden ayudar:

- Asegúrese de ejecutarlo como su propio usuario y no sudo.
- Realice una actualización completa de su sistema (`sudo yum update`).
- Reiniciar ayuda
- Intenta cerrar todas las aplicaciones Java en ejecución. Inicie VisualVM y vuelva a probar suerte.

**5. Agrega algunos complementos. B ntes**

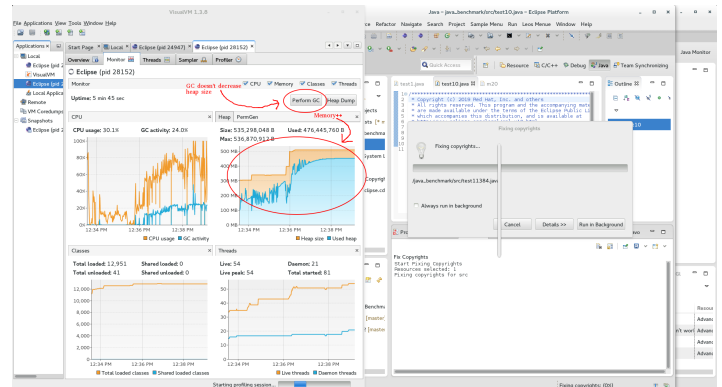
VisualVM convirtió útil para mí, primero tuve que añadir algunos plugins. Vaya a Herramientas -> Complementos -> 'Complementos disponibles'. Seleccione estos complementos (puede navegar y agregar más si lo desea):

- Piscinas de memoria
- GC Visual
- Matar la aplicación

## Analizar el código de ejecución con Visualvm

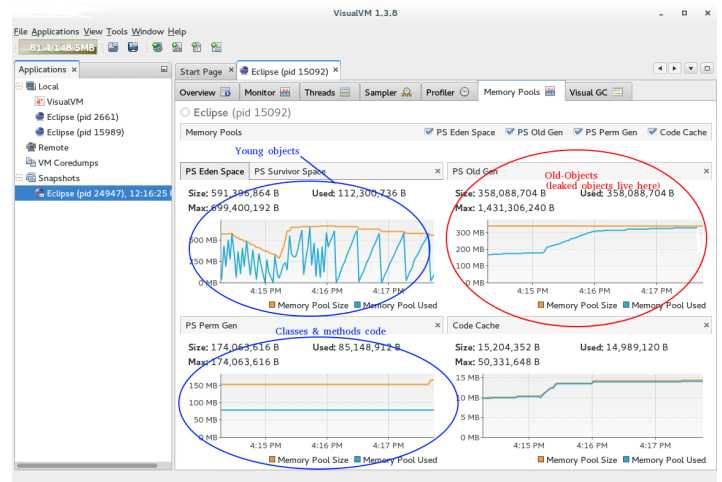
1. Ahora ejecuta tu aplicación Java,
2. Adjunte VisualVM a su aplicación.
3. Realice la operación que causa el rendimiento lento

4. Inspeccione el '*Monitor*' y la pestaña '*pools de memoria*'. Si ve que su memoria aumenta en la pestaña '*Monitor*', intente presionar '*Realizar GC*' (recolección de basura) y vea si eso disminuye el uso de memoria.



Si no...

5. Luego cambie a la pestaña '*pools de memoria*' e inspeccione la '*Vieja Generación*'. (Primera quedarse en "Edén", entonces la transición a través de espacios superviviente, los objetos más antiguos se mueven en la piscina 'Old Gen'. Si algo fugas, que va a ser en la piscina Viejo-Gen. Detalles )



6. Ahora regrese y comente la mayor parte del código de su programa hasta el punto en que la aplicación simplemente comienza y se detiene.
7. Repita hasta que la aplicación no se escape en absoluto.
8. Luego, a través de varias iteraciones, vuelva a habilitar partes de su código e inspeccione el uso de la memoria VisualVM. Cuando su aplicación comience a filtrarse nuevamente, ingrese al método que causó fugas de memoria y reduzca aún más.
9. Eventualmente, reducirá el problema a una sola clase. tal vez incluso a un solo método. Una vez que

esté allí, valide cuidadosamente que todos los búferes de archivos estén cerrados y que los Hashmaps se usen correctamente.

## Benchmarking su código

A veces es difícil saber si su nuevo código brillante es mejor que el código anterior. En este caso, es posible que desee comparar el rendimiento de su aplicación. Aquí hay un código que puede insertar en el lugar correcto para obtener información sobre el tiempo de ejecución y la cantidad de ejecuciones de recolección de basura:

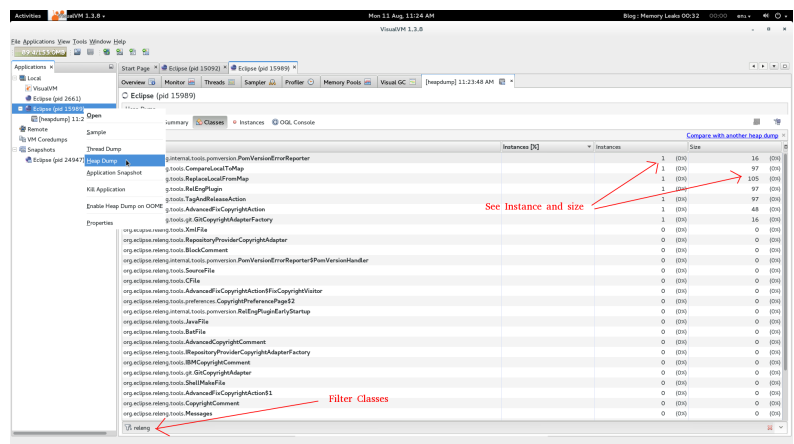
```

1      inicio prolongado = Sistema . currentTimeMilli
2      ..
3      //tu codigo
4      ..
5      extremo largo = Sistema . currentTimeMillis ()
6      Sistema . a cabo . println ( y amp ; quot ; Corre
7      Sistema . a cabo . println ( printGCStats ());
8
9      public static String printGCStats () {
10         long totalGarbageCollections = 0 ;
11         long garbageCollectionTime = 0 ;
12         para ( GarbageCollectorMXBean gc : ManagementFa
13             recuento largo = gc . getCollectionCount ();
14
15         if ( count & amp ; gt ; = 0 ) {
16             totalGarbageCollections += count ;
17         }
18
19         tiempo largo = gc . getCollectionTime ();
20
21         if ( tiempo & amp ; gt ; = 0 ) {
22             garbageCollectionTime += tiempo ;
23         }
24     }
25     regresar & amp ; quot ; Recolecciones de basura
26     &erio ; quot ; Tiempo de recolección de basura (
27 }
```

Como nota, si prueba en su Eclipse principal, se recomienda probar en un Eclipse hijo 'limpio'. O en alguna instancia 'limpia' de su Eclipse, porque otros complementos se interpondrán en el tiempo preciso.

## Ayuda adicional: Heap Dumps

Personalmente no he usado tanto, pero la gente habla maravillas de 'Heap-dumps'. En cualquier momento, puede tomar un montón de archivos y luego ver cuántas instancias de clases están abiertas y cuánto espacio utilizan. Puede hacer doble clic en ellos para ver su contenido. Esto es útil si desea ver cuántos objetos genera su aplicación.



## Espera, mi aplicación no tiene fugas, pero todavía es lento?

Es posible que no tenga ninguna fuga en su código, pero todavía es muy lento. En este caso, tendrá que perfilar su código. La creación de perfiles de código está fuera del alcance de este artículo, pero hay una gran conferencia de YouTube que explica cómo puede crear un perfil con Eclipse utilizando perfiles gratuitos y pagos.

## ¿A dónde ir desde aquí?

En esta etapa, trate de pasar uno o dos días solucionando la fuga de memoria. Si aún tiene problemas, intente leer algunos de estos artículos:

- Caza de fugas de memoria
- Fugas de memoria en clases internas y esto también es útil
- El perfil de la conferencia de YouTube mencionada anteriormente




- [Lea más de la guía JVM GC de Oracle](#)

---

Descargar Building Reactive Microservices en Java :  
diseño de aplicaciones asíncronas y basadas en  
eventos. Presentado en asociación con Red Hat .

---

Temas: JAVA, PÉRDIDAS DE MEMORIA

Publicado en DZone con el permiso de Leo Ufimtsev ,  
DZone MVB . [Vea el artículo original aquí.](#)   
Las opiniones expresadas por los contribuidores de  
DZone son suyas.

# Obtenga lo mejor de Java en su bandeja de entrada.

Manténgase actualizado con DZone's Bi-weekly Java  
Newsletter. [VER UN EJEMPLO](#)

SUSCRIBIR

## Recursos para socios de Java

---

Comience con Spring Boot, OAuth 2.0 y Okta

Okta



Garantizando la visibilidad en microservicios y contenedores

AppDynamics



Comandos avanzados de Linux [Cheat Sheet]

Programa de desarrollo de Red Hat



Patrones de diseño de Java EE modernos: creación de una  
arquitectura escalable para el desarrollo empresarial sostenible

Programa de desarrollo de Red Hat







































