PermGen Memory Leak

A continuación vamos a analizar los errores que causan ***java.lang.OutOfMemoryError: PermGen space*** síntomas en el seguimiento de la pila.

En primer lugar vamos a ir a través de la conceptos básicos necesarios para entender el tema - y explicar qué objetos, clases, cargadores de clases y el modelo de memoria de JVM son. Si está familiarizado con los conceptos básicos, puede saltar directamente a la sección siguiente,donde voy a describir dos casos típicos para el error en cuestión junto con notas y sugerencias para resolverlo.

Buscando una manera fácil de detectar fugas PermGen? Plumbr detecta fugas en minutos y automáticamente le indica cómo resolverlos.

Objetos, clases y cargadores de clases

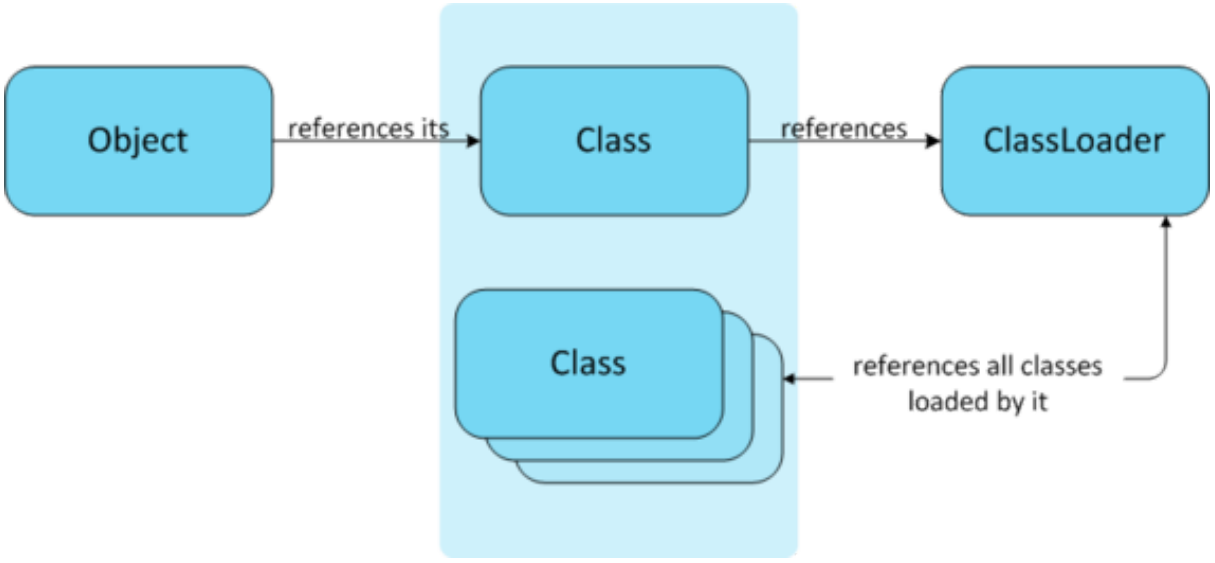
Bueno, no se iniciará con los mismos fundamentos. Creo que si nos ha encontrado, ya debe estar familiarizado con el concepto de que todo en Java es un **objeto.**Y que todos los objetos se especifican por su clase. Por lo que cada objeto tiene una referencia a una instancia de *java.lang.Class* describir la estructura de la clase de ese objeto.

Pero lo que realmente sucede bajo el capó cuando se crea un nuevo objeto en el código? Por ejemplo, si se escribe algo realmente complicado como el

```
jefe persona = new Persona (),
```

la Máquina Virtual de Java (JVM) tiene que entender la estructura del objeto a crear. Para lograr esto, la JVM busca la clase llamada Persona. Y sila *persona* se accede aclase depor primera vez durante esta ejecución particular del programa, tiene que ser *cargado* por la JVM, normalmente desde elcorrespondiente. *Person.class* archivo El proceso de búsqueda para el *Person.class* archivoen el disco, cargarlo en memoria y analizar su estructura se llama *la carga de*clases.Asegurar proceso de carga de clases apropiada es la responsabilidad de un ***classloader****.* Cargadores de clases son instancias de *java.lang.ClassLoader* la clasey todos y cada clase en un programa Java tiene que ser cargado por algún *classloader.* Como resultado, ahora tenemos las siguientes relaciones:

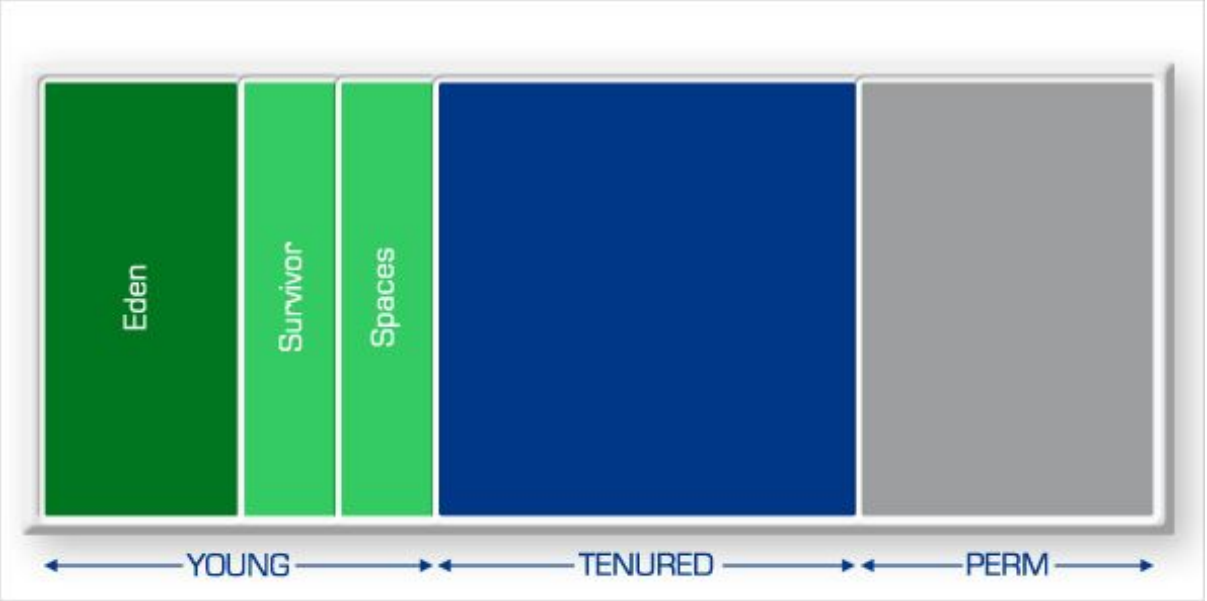
Como se puede ver en el siguiente diagrama de cada classloader tiene referencias a todas las clases se ha cargado. Para el propósito de nuestro artículo de estas relaciones son muy interesantes.



Recuerde que esta imagen, vamos a necesitar más tarde.

Generación Permanente

Casi todas las JVM utiliza hoy en día una región separada de la memoria, llamada la generación permanente (o **PermGen** para abreviar), para mantener representaciones internas de las clases de Java. PermGen también se utiliza para almacenar más información - averiguar los detalles de este post si está interesado - pero para nuestro artículo que es seguro asumir que sólo las definiciones de clase se almacenan en PermGen. El tamaño predeterminado de esta región en mis dos máquinas que funcionan con Java 1.6 no es un 82MB muy impresionante.



Como ya he explicado en una de mis entradas anteriores, una pérdida de memoria en Java es una situación en la que algunos objetos ya no son utilizados por una aplicación, pero el **garbage collector** lo reconoce como no utilizados. Esto lleva a la ***OutOfMemoryError*** si esos objetos no utilizados contribuyen al uso del montón significativamente suficiente como para que la próxima solicitud de asignación de memoria por la aplicación no se puede cumplir.

La causa fundamental de *java.lang.OutOfMemoryError: espacio PermGen* es exactamente el mismo: la JVM debe cargar la definición de una nueva clase, pero no hay suficiente espacio en el PermGen hacerlo - ya hay demasiadas clases almacenados allí. Una posible razón de esto podría ser su servidor de aplicaciones o el uso de demasiadas clases para el tamaño actual de PermGen ser capaz de adaptarse a ellos. Otra razón común podría ser una pérdida de memoria.

Generación de PermGen Memory Leaks

Como en todas las fugas de memoria, solo es posible usando o mediante testing, sin embargo este tipo de fugas se pueden crear mediante el deploy/undeploy. En caso de una aplicación web Java desplegado en un servidor de aplicaciones todas esas clases en su EAR / WAR pierden su valor cuando la aplicación está undeployed. La JVM continúa funcionando como servidor de aplicaciones está todavía vivo, pero un montón de definiciones de clases no están en uso más. Y deben ser removidos de PermGen. Si no, vamos a tener pérdida de memoria en el área PermGen.

Leaking Threads

Un posible escenario para una fuga del classloader es a través de Threads de larga duración. Esto sucede cuando su aplicación o una biblioteca de 3 ª parte que utiliza su

aplicación se inicia un poco de hilo de larga duración. Un ejemplo de esto podría ser un hilo temporizador cuyo trabajo consiste en ejecutar algún código periódicamente.

Si la vida útil prevista de este hilo no es fijo, nos dirigimos directamente en un problema. Cuando cualquier parte de su aplicación ***cada vez inicia un subproceso, debe asegurarse de que no va a sobrevivir a la aplicación***. En los casos típicos el desarrollador o bien no es consciente de esta responsabilidad o simplemente se olvida de escribir el código de limpieza.

De lo contrario, si algún hilo sigue funcionando después de la aplicación es desplegada por lo general tendrá una referencia a un classloader de la aplicación web que se inició por, denominado *contexto classloader*.Que a su vez significa que todas las clases de la aplicación sin desplegar siguen detenidos en la memoria. Para siolucionarlo, es la aplicación que inicia nuevos threads, debe cerrarlas durante el repliegue utilizando un Listener de contexto de servlet. Si se trata de una biblioteca tercera parte, debe buscar su propio gancho de cierre específica. O un informe de error si no hay ninguno.

Leaking Drivers

Otro caso típico de una fuga puede ser causada por los drivers de bases de datos. Destaquemos algunas cosas que suceden cuando esta aplicación se implementa en el servidor.

- El servidor crea una nueva instancia de *java.lang.Classloader* y empieza a cargar las clases de la aplicación de usarlo.
- Dado que el PetClinic utiliza una base de datos HSQL, se carga el controlador JDBC correspondiente, *org.hsqldb.jdbcDriver*
- esta clase, de ser un buen conductor de modales JDBC, registra a sí mismo con *java.sql.DriverManager* durante la inicialización, como se requiere por la especificación JDBC. Este registro incluye el almacenamiento en el interior de un campo estático de DriverManager una referencia a una instancia de org.hsqldb.jdbcDriver.

Ahora, cuando la aplicación está sin desplegar desde el servidor de aplicaciones, el *java.sql.DriverManager* será todavía mantienen esa referencia, ya que no hay código en la biblioteca HSQLDB ni en el marco de la primavera ni en la aplicación para eliminar eso! Como se explicó anteriormente, un *jdbcDriver* objeto todavía contiene una referencia a una *org.hsqldb.jdbcDriver,* clase de que a su vez contiene una referencia a la instancia de *java.lang.Classloader* utiliza para cargar la aplicación. Este classloader ahora sigue haciendo referencia a todas las clases de la aplicación. En el caso de nuestra aplicación de demostración en particular, durante el inicio de aplicaciones casi 2000 clases se cargan, que ocupa aproximadamente 10 MB en PermGen. Lo que significa que se tarda unos 5-10 vuelve a desplegar para llenar el PermGen con el tamaño por defecto para llegar a la *java.lang.OutOfMemoryError:PermGen* accidente espacio.

Para arreglar esto, una posibilidad es escribir un listener contexto de servlet, que de-registra el conductor HSQLDB de *DriverManager* durante el cierre de la aplicación. Esto es bastante sencillo. Pero recuerde - usted tendrá que escribir el código correspondiente en cada aplicación utilizando el controlador.

Conclusión

Hay muchas razones por las que su aplicación puede encontrarse con un ***java.lang.OutOfMemoryError: PermGen space***. La causa fundamental de la mayoría de ellos es alguna referencia a un objeto o una clase cargada por el *Classloader* de la aplicación que ha muerto después de eso. O un enlace directo al *Classloader*. Las acciones que debe tomar para el remedio son bastante similares para la mayoría de los casos de fuga. En primer lugar, averiguar dónde se llevará a cabo esa referencia. En segundo lugar, añadir un shutdown hook para eliminar la referencia durante undeployment. Usted puede hacer eso, ya sea usando un listener contexto servlet o mediante el uso de la API proporcionada por la biblioteca de tercera parte.