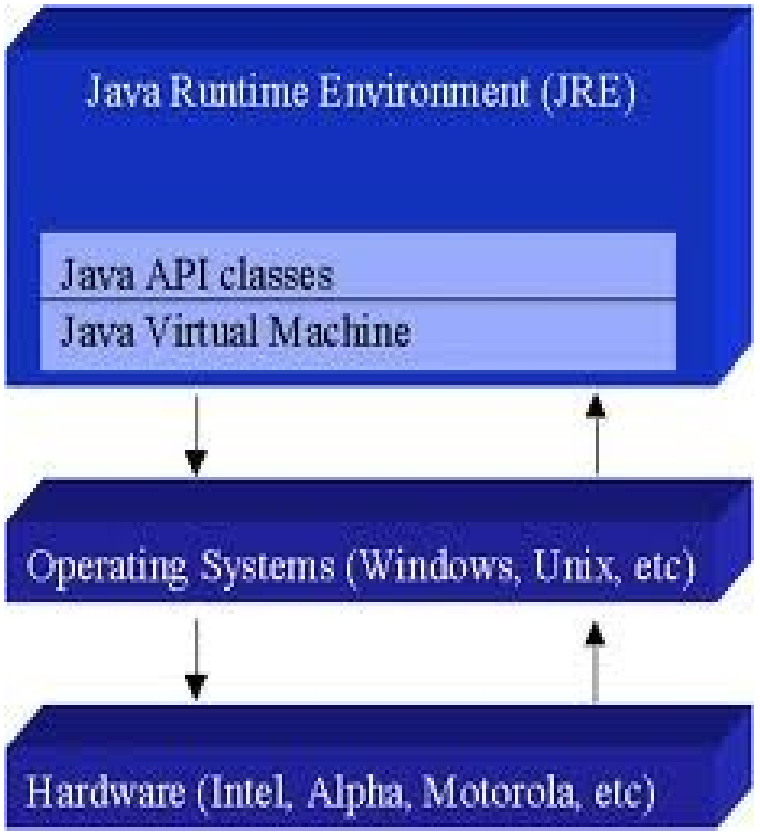
Introducción a la JVM

Carlos Zuluaga

Introducción y conceptos básicos

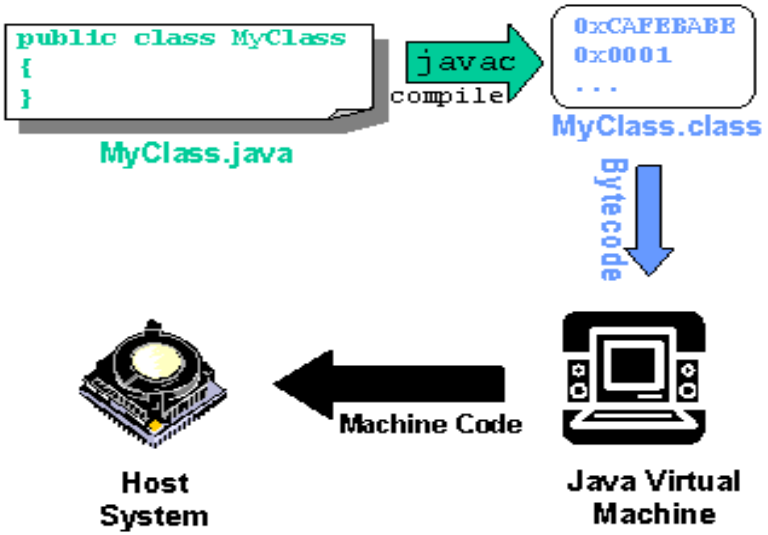
La JVM

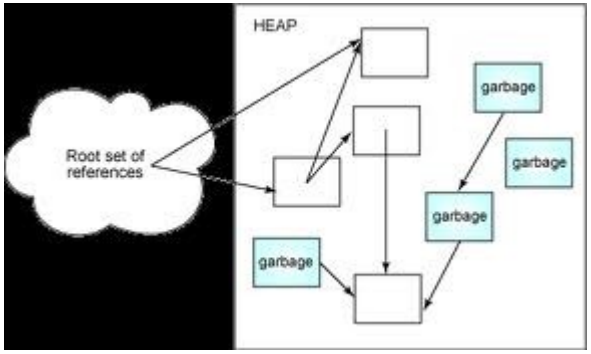
- Es el entorno *cerrado* en el que se ejecutan las aplicaciones y las utilidades de los servidores de aplicaciones.
- Es la intermediaria entre el programa y el sistema operativo para acceder a recursos como:
  - Memoria
  - Procesador
  - Disco
  - Red
- El aspecto más importante por comprender es el modelo de memoria y la recolección de basura.



El Compilador

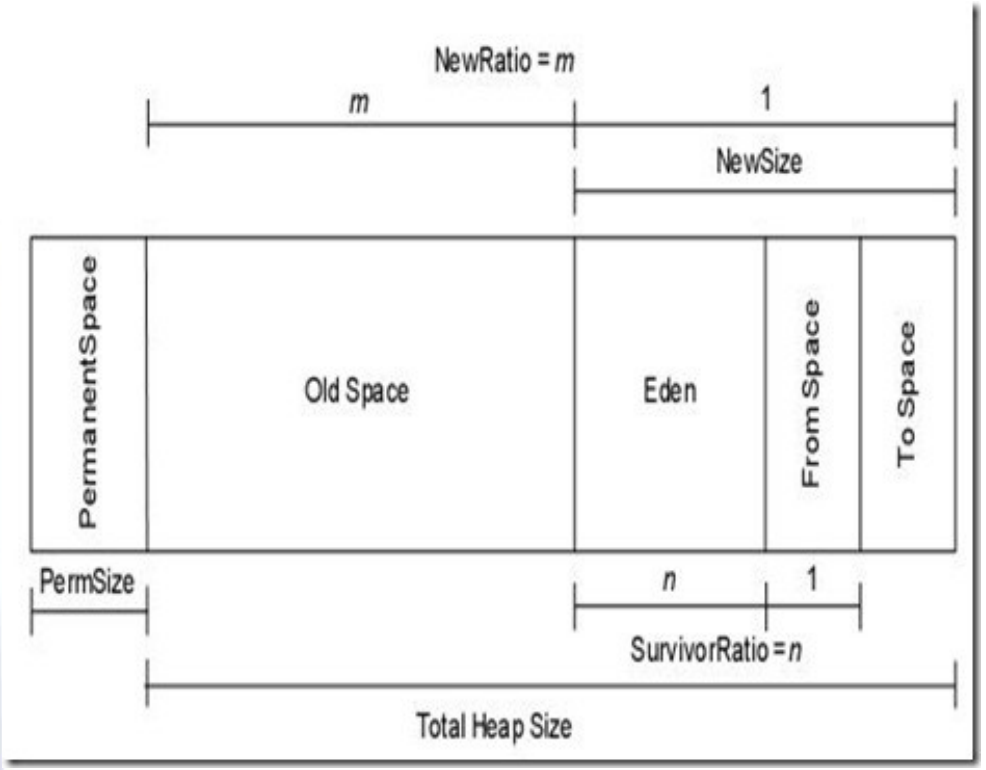
- La JVM tiene dos componentes principales: el JRE que es el entorno de ejecución y el JDK que es el kit de desarrollo usado para compilar el código fuente.
- Los archivos .java son compilados y transformados en código intermedio o bytecode, este luego es ejecutado por el JRE.
- Todos los objetos son apuntadores.





Modelo de Memoria

Estructura de Memoria



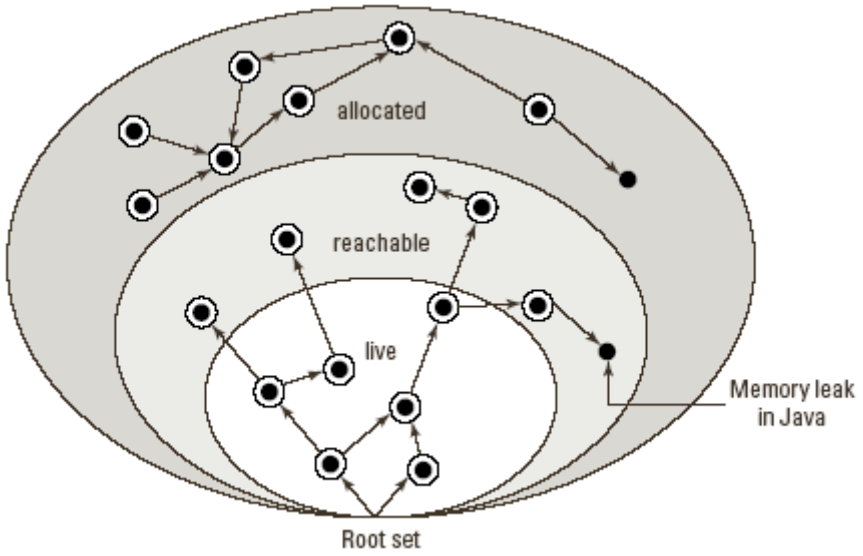
- Eden: memoria joven, en este espacio se crean los objetos.
- From y To space: secciones para objetos que sobreviven una recolección.
- Old Space: memoria para objetos que han sobrevivido varias recolecciones.
- Permanent Space: Para cargar definiciones de clases y el stack.

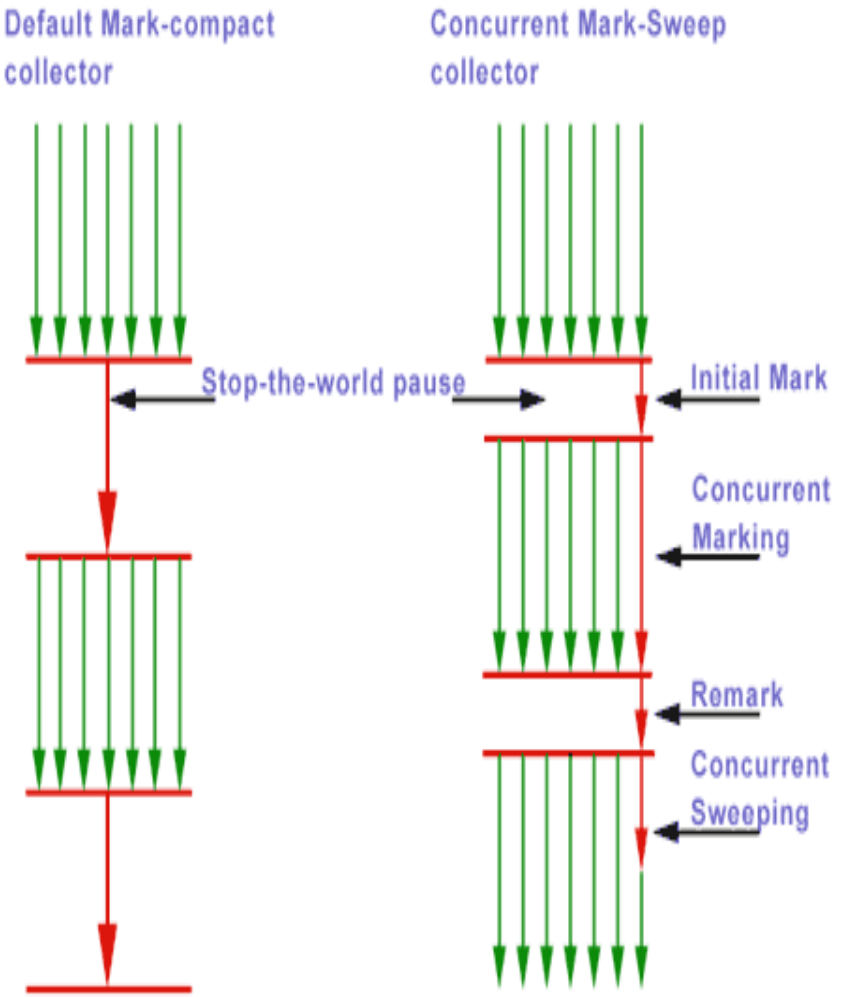
Estructura de Memoria

- -Xms: Tamaño mínimo del *Heap*.
- -Xmx: Tamaño máximo del *Heap*.
- -XX:NewSize: Tamaño mínimo del *Eden*.
- -XX:MaxNewSize: Tamaño máximo del *Eden*.
- -XX:PermSize: Tamaño mínimo del espacio permanente.
- -XX:MaxPermSize: Tamaño máximo del espacio permanente.
- Ejemplo:
- -Xms256m -Xmx512m -XX:MaxNewSize=220m -XX:PermSize=128m

Recolección de basura

Recolección de basura





VisualVM


Spaces
Perm
Old
Eden
S0
S1
Graphs
Compile Time: 3562 compiles - 1m 17,138s
Class Loader Time: 20725 loaded, 7629 unloaded - 11,946s
GC Time: 40767 collections, 20m 11,874s Last Cause: Allocation Failure
Eden Space (99,875M, 49,688M): 10,250M, 40355 collections, 14m 2,226s
Survivor 0 (33,312M, 23,688M): 2,115M
Survivor 1 (33,312M, 23,938M): 0
Old Gen (800,000M, 800,000M): 420,568M, 412 collections, 6m 9,648s
Perm Gen (160,000M, 160,000M): 74,830M

Algoritmos de recolección - HotSpot

Algoritmo	Características	Recomendado en
Serial Collector (-XX: +UseSerialGC)	Usa sólo una CPU.	Máquinas con una sola CPU, algoritmo por defecto en clientes.
Parallel Collector (-XX: +UseParallelGC)	- Usa múltiples CPUs para recolección de memoria young. - Sólo una CPU para recolecciones old. - Las recolecciones de memoria old en ocasiones pueden tener un gran impacto en el desempeño.	Máquinas con múltiples CPUs que realicen procesos batch, facturación, pagos, calculos científicos.
Parallel Compact Collector (-XX:+UseParallelOldGC)	- Usa el mismo algoritmo del Parallel Collector para recolecciones young. - En las recolecciones old agrega las fases marking, summary y compaction. - Usa múltiples CPUs en todas las recolecciones. - Se le pude indicar a la JVM que haga una selección automática basada en uno de estos parámetros: -XX:MaxGCPauseMillis=n → Máximo tiempo de las pausas. -XX:GCTimeRatio=n → 1/(1+n) Porcentaje total de tiempo para las recolecciones.	Máquinas con múltiples CPUs. Puede no funcionar adecuadamente en aplicaciones que comparten recursos.
***Concurrent Mark-Sweep (CMS) Collector* (-XX: +UseConcMarkSweepGC)**	**- Conocido también como *low-latency* collector. - Las recolecciones jóvenes se hacen con el mismo algoritmo de los recolectores Parallel. - Las recolecciones de objetos viejos se hacen casi todas en paralelo con la ejecución de la aplicación. - Reduce los tiempos de recolecciones viejas con el precio de recolecciones jóvenes un poco más largas y requerimientos extras de memoria Heap. - Cuando se cuenta con un número relativamente pequeño de procesadores (1, 2) se recomienda activar el modo incremental (-XX: +CMSIncrementalMode) - Es el único algoritmo que no tiene la etapa *compactar*.**	**Aplicaciones ejecutándose en máquinas multiprocesador y que tienen un número relativamente grande de objetos de larga duración, por ejemplo un servidor Web.**

Algoritmos de recolección - IBM

Algoritmo	Características	Recomendado en
-Xgcpolicy:optthruput	- Optimizado para un gran throughput. - Recolecciones de basura completas pueden tener tiempos muy altos con el agravante del STW.	Procesos batch.
-Xgcpolicy:gencon	**- Generacional concurrente.** **- Optimizado para aplicaciones con un gran número de muertes jóvenes.** **- Divide la memoria en secciones para objetos jóvenes y viejos.**	**Sistemas transaccionales.**
-Xgcpolicy:optavgpause	- Algoritmo optimizado para aplicaciones donde el tiempo de respuesta es importante. - Reduce los tiempos STW.	IBM lo recomienda para su portal o para sistemas transaccionales.
-Xgcpolicy:subpools	- Optimizado para sistemas multiprocesador.	Máquinas con 16 procesadores o más.

Algoritmos de recolección - JRockit

Algoritmo	Características	Recomendado en
Estrategia estática: gencon	- Usa dos generaciones de objetos y la mayoría de recolecciones concurrentes.	- Aplicaciones sensibles ante grandes pausas por GC. - Muchos objetos de corta duración o muerte joven.
Estrategia estática: singlecon	- Usa sólo una generación y la mayoría de recolecciones concurrentes.	- Aplicaciones sensibles ante grandes pausas por GC. - Pocos objetos de corta duración.
Estrategia estática: singlepar	- Usa una sola generación de objetos y recolector paralelo.	- No hay problema con pausas altas por GC. - Pocos objetos de corta duración.
Estrategia estática: genpar	- Usa dos generaciones de objetos y el recolector paralelo.	- No hay problema con pausas altas por GC. - Muchos objetos de corta duración.
Throughput mode -XgcPrio:throughput (jrockit)	- Asigna toda la CPU posible a la ejecución de la aplicación. - Tiene pocas pausas pero pueden ser largas.	- Aplicaciones que requierene máximo rendimiento pero no son sensibles a altos tiempos de GC. - Batch
Pausetime Mode -XgcPrio:pausetime (jrockit)	- Balanceo entre el máximo throughput posible con un número bajo de pausas por GC. - Se puede usar en combinación con el parámetro: -XpauseTarget:<tiempo en ms> - Por defecto el tiempo de la pausa es de 500ms.	- Aplicaciones donde el tiempo de respuesta es importante. - Sistemas transaccionales.
Deterministic -XgcPrio:deterministic	- Usado para asegurar tiempos de recolección de basura muy cortos. - Tenga cuidado cuando lo use con JRockit Mission Control Client. - Las pausas son por defecto de 30 milisegundos y se puede modificar con el parámetro -XpauseTarget:<tiempo en ms>	- Sistemas donde es muy importante tener una baja latencia. - Transaccionales.

Referencias

- http://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/Connectors_on_JBoss-Configuring_JDBC_DataSources.html
- http://community.jboss.org/wiki/ConfigDataSources
- http://community.jboss.org/wiki/Ejb3DisableSfsbPassivation
- http://java.sun.com/performance/reference/whitepapers/6_performance.html
- http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140228.html
- http://redstack.wordpress.com/2011/01/06/visualising-garbage-collection-in-the-jvm/
- http://java.sun.com/developer/technicalArticles/J2SE/monitoring/
- Joines, Stacy. *Performance Analysis for Java Web Sites*.
- Molyneaux, Ian. *The Art of Application Performance Testing*.
- Haines, Steven. *Pro Java EE 5, Performance Management and Optimization*.
- http://heinsohn.wikidot.com/talleres:pruebas-carga-optimizacion