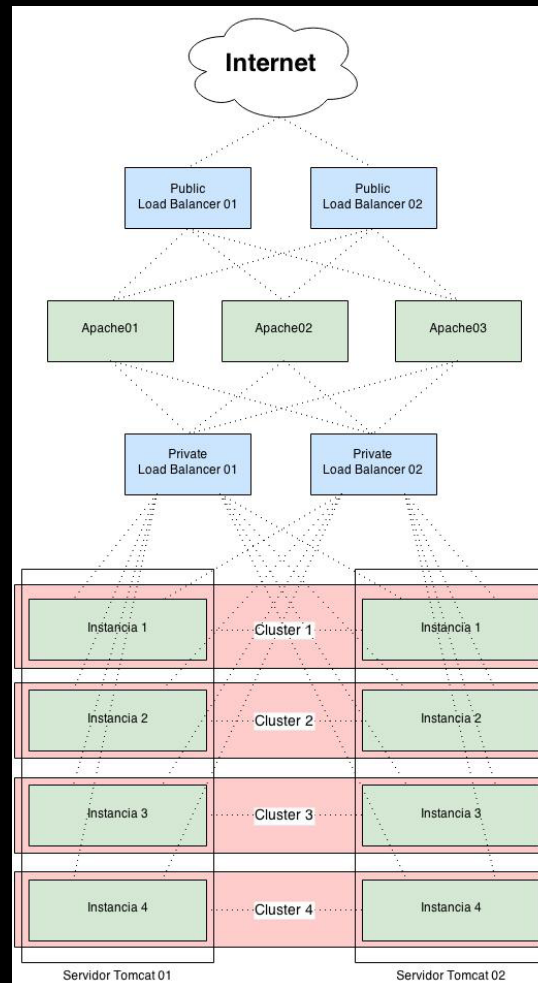


CONFIGURACIÓN AVANZADA DE TOMCAT



CONFIGURACIÓN Y ADMINISTRACIÓN DE UN CLUSTER



CONFIGURACIÓN Y ADMINISTRACIÓN DE UN CLUSTER

- Balanceador de carga
- Definir mecanismo para usar las sticky-session
- Configuración de los workers
- Consideraciones a tener en cuenta
- Configurar la aplicación para trabajar en cluster
- Configurar el cluster de servidores tomcat
- Donde situar las aplicaciones para el despliegue
- Establecer configuración para evitar los fallos del sistema



CONFIGURACIÓN Y ADMINISTRACIÓN DE UN CLUSTER

Tomcat tiene integrado soporte para **cluster**, **balanceo de carga**, y la **persistencia de sesiones**. Esto significa que el sistema es escalable en caso de que lo necesitemos.

Trabajar en Cluster permite además tener una alta disponibilidad.



CONFIGURACIÓN Y ADMINISTRACIÓN DE UN CLUSTER

Escalabilidad

Con mucha frecuencia se confunde el trabajo en Clusters con la escalabilidad, pero se trata de un metodo de escalar y no el concepto en si mismo.

Escalar tiene que ver con la capacidad de adaptar nuestro servidor a una demanda cambiante de peticiones sin que eso afecte al rendimiento o el tiempo que se tarda en servir las respuestas.

Un cluster puede proporcionarnos escalabilidad ya que en el caso de cambiar la demanda podemos cambiar el número de workers dedicados a nuestra aplicación.



CONFIGURACIÓN Y ADMINISTRACIÓN DE UN CLUSTER

Balanceo de carga

El balanceo de carga es un grupo de tecnologías destinadas a la distribución de carga de solicitudes a través de un grupo de servidores. El balanceo de carga es un componente clave de una solución de clustering.

Balanceador de carga

Tomcat está preparado tanto para balanceadores de carga por hardware como por software:

- **Hardware** Estos dispositivos utilizan procesadores diseñados específicamente para la distribución de grandes volúmenes de carga de manera eficiente, y también por lo general incluyen una variedad de compresión, cacheing, y las opciones de cola y seguridad.

Son equipos muy costosos y en el caso de requerir alta disponibilidad tendremos que contar con varios de ellos. Sus características y funcionamiento varían según el fabricante.



Balanceador de carga

- **Software** De forma homologa a lo anterior servidor dedicado o un grupo de servidores actuando como un proxy pueden distribuir la carga.

Apache Existen modulos como mod_proxy y mod_jk que conectan Apache con Tomcat. Si usamos dichos conectores para distribuir la carga entre varios Tomcat constituiriamos un balanceador de cargas. Otras soluciones como **HAProxy** y **Nginx** son soluciones más modernas y eficientes.



Definir mecanismo para usar las sticky-session

Con el uso de un cluster tendremos solución tanto para la escalabilidad como para la alta disponibilidad pero nos surge un nuevo problema, la duplicación de sesiones.

Podemos implementar **sesiones pegajosas** para que todas las peticiones de un mismo cliente hagan llamada al mismo servidor. También podemos **replicar las sesiones** en los demás servidores



Configuración de los workers

instancia-01

```
1 <Server port="8005" shutdown="SHUTDOWN">  
2   Connector port="8080" protocol="HTTP/1.1"  
3   <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

instancia-02

```
1 <Server port="8105" shutdown="SHUTDOWN">  
2   Connector port="8180" protocol="HTTP/1.1"  
3   <Connector port="8109" protocol="AJP/1.3" redirectPort="8443" />
```

instancia-03

```
1 <Server port="8205" shutdown="SHUTDOWN">  
2   Connector port="8280" protocol="HTTP/1.1"  
3   <Connector port="8209" protocol="AJP/1.3" redirectPort="8443" />
```

instancia-04

```
1 <Server port="8305" shutdown="SHUTDOWN">  
2   Connector port="8380" protocol="HTTP/1.1"  
3   <Connector port="8309" protocol="AJP/1.3" redirectPort="8443" />
```



Configuración de los workers

```
1 vi /etc/init.d/tomcat7-01
2
3 #!/bin/sh
4 ### BEGIN INIT INFO
5 # Provides: Tomcat
6 # Required-Start: $network
7 # Required-Stop: $network
8 # Default-Start: 2 3 5
9 # Description: Java Servlet and JSP Engine
10 ### END INIT INFO
11
12 JAVA_HOME=/usr/lib/jvm/java-7-oracle/
13 JAVA_OPTS="-Xmx800m -Xms800m"
14 CATALINA_HOME=/usr/local/tomcat7
15 CATALINA_BASE=/var/www/instancia-01
16
17 case "$1" in
18 'start')
19 $CATALINA_HOME/bin/catalina.sh start
20 ;;
21 'stop')
22 $CATALINA_HOME/bin/catalina.sh stop
23 ;;
24 *)
25 echo "Usage: $0 { start | stop }"
26 ::
```



Configuración de los workers

```
1 className="org.apache.catalina.ha.tcp.SimpleTcpCluster" channelSendOptions
```

```
1 name="Catalina" defaultHost="localhost" jvmRoute="servidor01-instancia-01"
```



Configuración de los workers

```
1 vi /etc/apache2/mods-enabled/jk.conf
2 # Where to find workers.properties
3 JkWorkersFile /etc/libapache2-mod-jk/workers.properties
4
5 # Where to put jk logs
6 JkLogFile /var/log/apache2/mod_jk.log
7
8 # Set the jk log level [debug/error/info]
9 JkLogLevel info
10
11 JkShmFile /var/log/apache2/jk-runtime-status
12
13 # Select the log format
14 JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
15
16 # JkOptions indicate to send SSL KEY SIZE
17 JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
18
19 JkMount /contexto_de_la_aplicacion_01/ ajp13_worker_01
20 JkMount /contexto_de_la_aplicacion_02/ ajp13_worker_02
21 JkMount /contexto_de_la_aplicacion_03/ ajp13_worker_03
22 JkMount /contexto_de_la_aplicacion_04/ ajp13_worker_04
23
24 JkMount /contexto_de_la_aplicacion-01/*.do ajp13_worker_01
25 JkMount /contexto_de_la_aplicacion-02/*.do ajp13_worker_02
26 JkMount /contexto_de_la_aplicacion-03/*.do ajp13_worker_03
27 JkMount /contexto_de_la_aplicacion-04/*.do ajp13_worker_04
28
29 # vi /etc/libapache2-mod-jk/workers.properties
30
31 worker.list=loadbalancer,stat1,ajp13_worker_01,ajp13_worker_02,ajp13_worker_03,ajp13_worker_04
32
33 worker.ajp13_worker_01.port=8009
34 worker.ajp13_worker_01.host=192.168.0.209
35 worker.ajp13_worker_01.type=ajp13
36 worker.ajp13_worker_01.lbfactor=1
37
38 worker.ajp13_worker_02.port=8109
39 worker.ajp13_worker_02.host=192.168.0.209
40 worker.ajp13_worker_02.type=ajp13
41 worker.ajp13_worker_02.lbfactor=1
42
43 worker.ajp13_worker_03.port=8209
44 worker.ajp13_worker_03.host=192.168.0.209
45 worker.ajp13_worker_03.type=ajp13
46 worker.ajp13_worker_03.lbfactor=1
47
48 worker.ajp13_worker_04.port=8309
49 worker.ajp13_worker_04.host=192.168.0.209
50 worker.ajp13_worker_04.type=ajp13
51 worker.ajp13_worker_04.lbfactor=1
```



Configuración de los workers

```
1 vi /etc/apache2/sites-enabled/vhost-balanceo-tomcat.conf
2
3 JkMount /contexto_de_la_aplicacion-01/* ajp13_worker_01
4
5 Options -Indexes FollowSymLinks MultiViews
6 AllowOverride None
7 Order allow,deny
8 allow from all
9
10 JkMount /contexto_de_la_aplicacion-02/* ajp13_worker_02
11
12 Options -Indexes FollowSymLinks MultiViews
13 AllowOverride None
14 Order allow,deny
15 allow from all
16
17 JkMount /contexto_de_la_aplicacion-03/* ajp13_worker_03
18
19 Options -Indexes FollowSymLinks MultiViews
20 AllowOverride None
21 Order allow,deny
22 allow from all
23
24 JkMount /contexto_de_la_aplicacion-04/* ajp13_worker_04
25
26 Options -Indexes FollowSymLinks MultiViews
27 AllowOverride None
28 Order allow,deny
29 allow from all
```



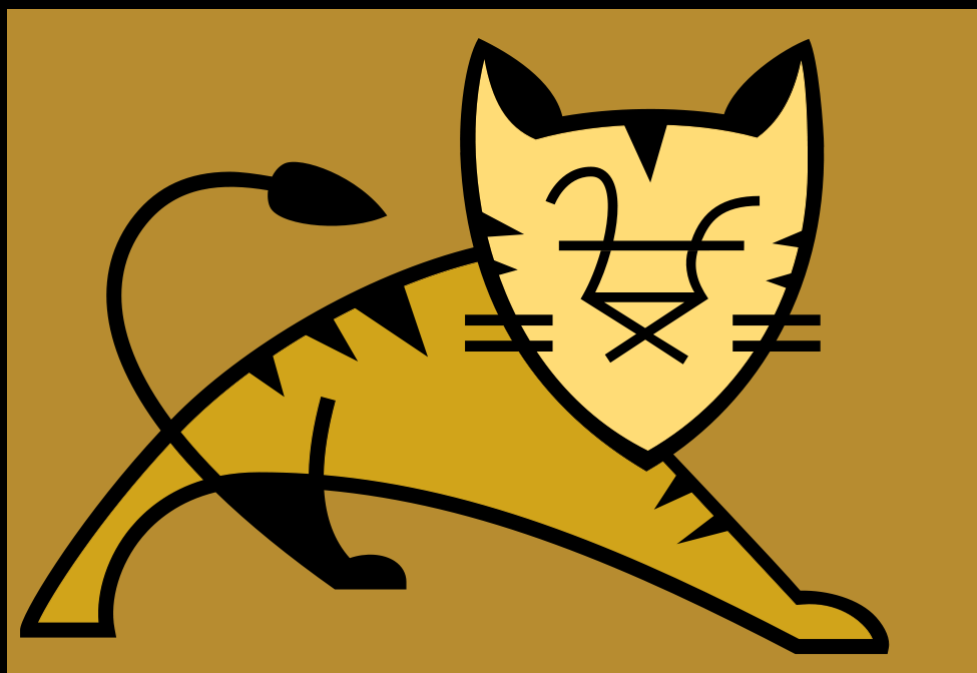
Configurar la aplicación para trabajar en cluster

Y es necesario especificar en el web.xml de la aplicación web a desplegar, la opción `<distributed />`

Usaremos una copia de nuestra aplicación en cada uno de los CATALINA_BASE de nuestros tomcats



HERRAMIENTAS DE PERFILADO



Concepto del profiling de aplicaciones

Los tests de performance son aquellos que sirven para determinar qué tan rápido o qué tan bien se comporta un sistema sometido a una carga en particular. También pueden ser utilizados para validar y verificar otros requerimientos no funcionales del sistema como ser estabilidad, escalabilidad, disponibilidad o consumo de recursos.

Los tests de performance pueden buscar diferentes objetivos. Pueden servir para demostrar que un sistema cumple con determinado criterio de aceptación, para comparar dos sistemas y determinar cuál se comporta mejor o bien para detectar qué sistema externo o qué componente interno es el cuello de botella.

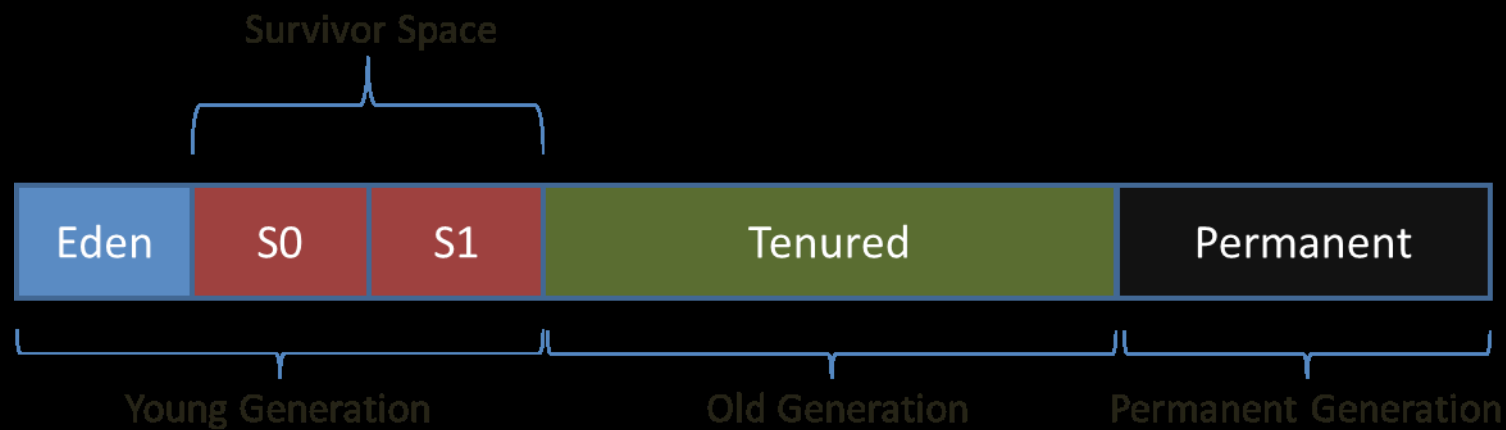


Necesidades de profiling en Java

Dado que las aplicaciones Java no se ejecutan directamente por el sistema operativo, debe iniciar Java Virtual Machine y pasarle la aplicación. Por lo tanto, para perfilar un archivo Java, debe especificar el lanzador de aplicaciones Java como la aplicación de host.



Conociendo el heap y cómo actúa



Hotspot VM Structure



Necesidades de profiling en Java

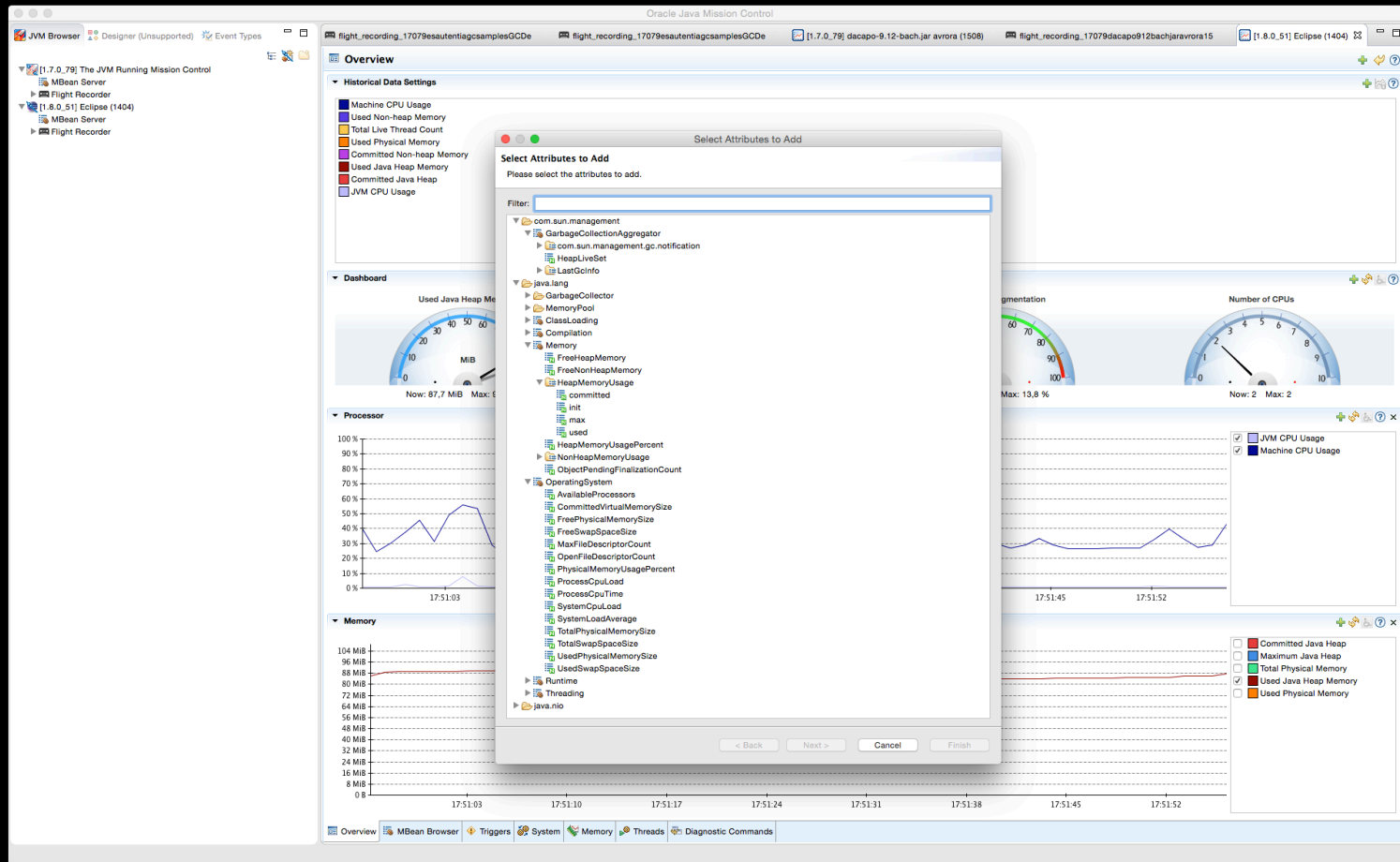
- Cualquier objeto nuevo es colocado en Eden.
- Cuando el Eden se llena, se realiza un reciclaje menor (minor garbage collection).
- Los objetos con referencias se mueven al primer espacio de supervivientes (S0). El resto serán eliminados en la siguiente limpieza de Eden.
- En el siguiente reciclaje menor, Eden sufre el mismo proceso antes descrito. Los objetos no referenciados son eliminados de Eden; sin embargo, los objetos viejos tanto de Eden como de S0 son movidos al segundo espacio de supervivientes (S1). Una vez terminada la copia, tanto Eden como S0 son limpiados.



Necesidades de profiling en Java

- En el siguiente reciclaje menor, los espacios de supervivientes (S0 y S1) se intercambian los roles y vuelven a repetir el proceso.
- Cuando algún objeto de cualquier de los espacios de supervivientes alcanza un determinado tiempo de vida, es promocionado a la Old Generation.
- Eventualmente, se producirá un reciclaje amplio (major garbage collection) que limpiará y compactará este espacio de la Old Generation.

Monitorización del heap, uso de herramientas



Monitorización del heap, uso de herramientas

Oracle Java Mission Control

JVM Browser [1.7.0_79] The JVM Running Mission Control

MBean Browser

MBean Tree

Filter:

- JVMImplementation
- MBeanServerDelegate
- com.sun.management
 - GarbageCollectionAggregator
 - HotSpotDiagnostic
- java.lang
 - GarbageCollector
 - PS MarkSweep
 - PS Scavenge
 - MemoryManager
 - CodeCacheManager
 - MemoryPool
 - Code Cache
 - PS Eden Space
 - PS Old Gen**
 - PS Perm Gen
 - PS Survivor Space
 - ClassLoading
 - Compilation
 - Memory
 - OperatingSystem
 - Runtime
 - Threading
- java.nio
 - BufferPool
 - direct
 - mapped
- java.util.logging
 - Logging

MBean Features

Name	Value	Update Interval
CollectionUsage	CompositeData, size 4	Default
CollectionUsageThreshold	0 8	Default
CollectionUsageThresholdCount	0	Default
CollectionUsageThresholdExceeded	false	Default
CollectionUsageThresholdSupported	true	Default
MemoryManagerNames	String[]	Default
Name	PS Old Gen	Once
ObjectName	java.lang:type=MemoryPool,name=PS Old Gen	Once
PeakUsage	CompositeData, size 4	5 s
Type	HEAP	Once
Usage	CompositeData, size 4	5 s
UsageThreshold	0 8	Default
UsageThresholdCount	0	Default
UsageThresholdExceeded	false	Default
UsageThresholdSupported	true	Default
Valid	true	Default

Overview MBean Browser Triggers System Memory Threads Diagnostic Commands



Monitorización del heap, uso de herramientas

The screenshot displays the Oracle Java Mission Control (JVM Browser) interface. The main window is titled "Oracle Java Mission Control" and shows a list of monitored JVMs at the top. The left pane, labeled "JVM Browser", lists several JVMs, including "The JVM Running Mission Control" and "dacapo-9.12-bach.jar lusearch (1008)". The right pane is divided into two sections: "Triggers" and "Rule Details".

Triggers Section:

- Java SE:**
 - ☒ CPU Usage - JVM Process (Too High)
 - ☐ CPU Usage - JVM Process (Too Low)
 - ☐ CPU Usage - Machine (Too High)
 - ☐ CPU Usage - Machine (Too Low)
 - ☐ Deadlocked Threads
 - ☐ Live Set (Too Large)
 - ☐ Monitored Deadlocked Threads
 - ☐ Thread Count (Too High)
- WebLogic Server 10.3 - Examples Server:**
 - ☐ Memory Pressure (Too High)
 - ☐ Open Sessions (Too Many)
 - ☐ Pending JMS Messages (Too High)
 - ☐ Pending Queued Requests (Too Many)
 - ☐ Primary Objects (Too Many)
 - ☐ Requests Waiting for DB Connection (Too High)
 - ☐ Server Health (Not OK)
 - ☐ Server State (Not running)
 - ☐ Threads Waiting for Bean (Too Many)

Rule Details Section:

Condition: CPU Usage - JVM Process (Too High)

Action:

Constraints:

Description:
The attribute ProcessCpuLoad in the OperatingSystem Mbean reports the average load of all processors for the JVM process.
A high CPU usage may indicate some performance problem.

MBean Path: java.lang:type=OperatingSystem

Attribute Name: ProcessCpuLoad

Current Value: 72 %

Max trigger value: 50 %

Sustained period: 5 s

Limit period: 60 s

☒ Trigger when condition is met.

☒ Trigger when recovering from condition



Monitorización del heap, uso de herramientas

Oracle Java Mission Control

JVM Browser Event Types

[1.7.0.79] The JVM Running Mission Control

[1.7.0.78] dacapo-9.12-bach.jar lusearch (1103)

[1.8.0.51] Eclipse (839)

Threads

Live Thread Graph

Live Threads 11:16:20

Filter Column Thread Name

☐ CPU Profiling ☒ Deadlock Detection ☒ Allocation

Thread Name	Thread State	Blocked Count	Total CPU Usage	Deadlocked	Allocated Bytes
Query1	RUNNABLE	0	Not Enabled	No	293,00 MB
Query0	RUNNABLE	0	Not Enabled	No	294,72 MB
RMI TCP Connection(idle)	TIMED_WAITING	0	Not Enabled	No	227,89 kB
RMI TCP Connection(3)-192.168.168.178	TIMED_WAITING	322	Not Enabled	No	14,46 MB
RMI TCP Connection(5)-192.168.168.178	RUNNABLE	1	Not Enabled	No	8,32 MB
JMX server connection timeout 18	TIMED_WAITING	779	Not Enabled	No	306,94 kB
RMI Scheduler(0)	TIMED_WAITING	0	Not Enabled	No	4,51 kB
RMI TCP Connection(1)-192.168.168.178	RUNNABLE	33	Not Enabled	No	10,31 MB
RMI TCP Accept-0	RUNNABLE	0	Not Enabled	No	11,45 kB
Attach Listener	RUNNABLE	0	Not Enabled	No	1,11 MB
VM JFR Buffer Thread	RUNNABLE	0	Not Enabled	No	0 bytes
JFR request timer	WAITING	0	Not Enabled	No	0 bytes
Signal Dispatcher	RUNNABLE	0	Not Enabled	No	320 bytes
Finalizer	WAITING	406	Not Enabled	No	464 bytes
Reference Handler	WAITING	398	Not Enabled	No	328 bytes
main	WAITING	1.218	Not Enabled	No	430,24 MB

Stack traces for selected threads

Stack traces for selected threads 11:16:20

▼ RMI TCP Connection(3)-192.168.168.178 [20] (TIMED_WAITING)

- java.lang.Object.wait line: not available [native method]
- com.sun.jmx.remote.internal.ArrayNotificationBuffer.fetchNotifications line: 449
- com.sun.jmx.remote.internal.ArrayNotificationBuffer\$ShareBuffer.fetchNotifications line: 227
- com.sun.jmx.remote.internal.ServerNotificationForwarder.fetchNotifs line: 275
- javax.management.remote.rmi.RMIConnectionImpl\$3.run line: 1289
- javax.management.remote.rmi.RMIConnectionImpl\$3.run line: 1287
- javax.management.remote.rmi.RMIConnectionImpl.fetchNotifications line: 1293
- sun.reflect.GeneratedMethodAccessor20.invoke line: not available
- sun.reflect.DelegatingMethodAccessorImpl.invoke line: 43
- java.lang.reflect.Method.invoke line: 606
- sun.rmi.server.UnicastServerRef.dispatch line: 322
- sun.rmi.transport.Transport\$2.run line: 202
- sun.rmi.transport.Transport\$2.run line: 199
- java.security.AccessController.doPrivileged line: not available [native method]
- sun.rmi.transport.service.Call line: 198
- sun.rmi.transport.tcp.TCPEndpoint.handleMessage line: 687

Overview MBean Browser Triggers System Memory Threads Diagnostic Commands



Monitorización del heap, uso de herramientas

Existen muchísimas aplicaciones para el perfilado. Las más conocidas son:

- Java Flight Recorder
- JProfiler
- GC Viewer
- VisualVM
- Eclipse Memory Analyzer



Localizando cuellos de botella en llamadas y cpu

Decimos que hay cuello de botella y que el mismo lo produce la CPU de nuestro equipo cuando la misma es incapaz de ofrecer un rendimiento a la altura del resto del sistema y por tanto frena a otros componentes, impidiendo que éstos puedan desarrollar todo su potencial.



Localizando cuellos de botella en llamadas y cpu

Decimos que hay cuello de botella y que el mismo lo produce la CPU de nuestro equipo cuando la misma es incapaz de ofrecer un rendimiento a la altura del resto del sistema y por tanto frena a otros componentes, impidiendo que éstos puedan desarrollar todo su potencial.



OPTIMIZANDO EL JDK



El garbage collector, configuración y tipos

Los procesos de GC permiten a una aplicación Java usar de manera eficiente la memoria que le es asignada, pero ya que estos procesos requieren en algunos casos detener las transacciones de la JVM, tienen la capacidad de afectar el desempeño de una aplicación si se parametrizan de manera inadecuada.

Pueden ser clasificados de manera global en:

- Algoritmos de GC Seriales.
- Algoritmos de GC Paralelos.
- Algoritmos de GC Concurrentes.
- Algoritmos de GC Mixtos.



Recolector Serie.

Recolector que trabaja sobre una única CPU tanto para los objetos de generación joven como de generación vieja. Este recolector es el recomendado para sistemas embebidos que no tienen mucha capacidad de procesamiento.



El recolector Parallel

Recolecta los objetos de generación joven en paralelo, mientras que los de generación vieja los recolecta en serie. Este recolector es útil cuando se tienen multiples CPUs.



El recolector Parallel compacting

Tanto los objetos de generación joven como de generación vieja se recolectan en paralelo y además se compactan.



El recolector Parallel compacting

Tanto los objetos de generación joven como de generación vieja se recolectan en paralelo.



El recolector Parallel compacting

Utiliza un algoritmo Mark / Sweep para recolectar los objetos que consiste en:

- Revisión del heap identificando los objetos referenciados y marcándolos como vivos y los no referenciados y marcándolos como muertos.
- De nuevo se vuelve a revisar el heap buscando los objetos marcados como muertos y liberando los espacios de memoria ocupados por estos.

Este recolector no mantiene la compactación activa por defecto y utiliza más memoria y CPU que los anteriores recolectores en favor de mantener unos períodos STW más cortos.



El HotSpot según ergonómico

Hablamos de un proceso ergonómico cuando sincronizamos la JVM y el GC para mejorar el rendimiento de las aplicaciones.

La JVM ofrece selecciones predeterminadas para el recolector de basura, tamaño del heap, y el compilador en tiempo de ejecución. Estas selecciones se ajustan a las necesidades de los diferentes tipos de aplicaciones, mientras que requiere menos de sintonización de línea de comandos. Además, el ajuste basado en el comportamiento optimiza dinámicamente el tamaño de la pila para satisfacer un comportamiento específico de la aplicación.



El HotSpot según ergonómico

Los recolectores de basura de Java HotSpot VM se pueden configurar para satisfacer preferentemente uno de dos objetivos: el máximo tiempo de pausa y el rendimiento de las aplicaciones. Si se cumple el objetivo preferido, los colectores tratarán de maximizar la otra:

- Menor tiempo de pausa
- Aumentar el rendimiento
- Reducir el tamaños del Head



Conociendo la arquitectura de un servidor JEE

Los servidores Java EE permiten hacer aplicaciones de n-nivles, lo cual tiene las siguientes ventajas:

- Permite una mejor utilización de los recursos
- Permite una mayor especialización de cada nivel

Y sus inconvenientes:

- Los niveles se han de integrar de forma eficiente
- Se requieren servicios
- Hay que mantener conexiones entre los niveles
- Dificulta la portabilidad y el mantenimiento



Conociendo la arquitectura de un servidor JEE

Java EE facilitan hacer aplicaciones en n-niveles.

- Los contenedores proveen un entorno de ejecución
- Tienen servicios de conectividad, seguridad, administración, etc

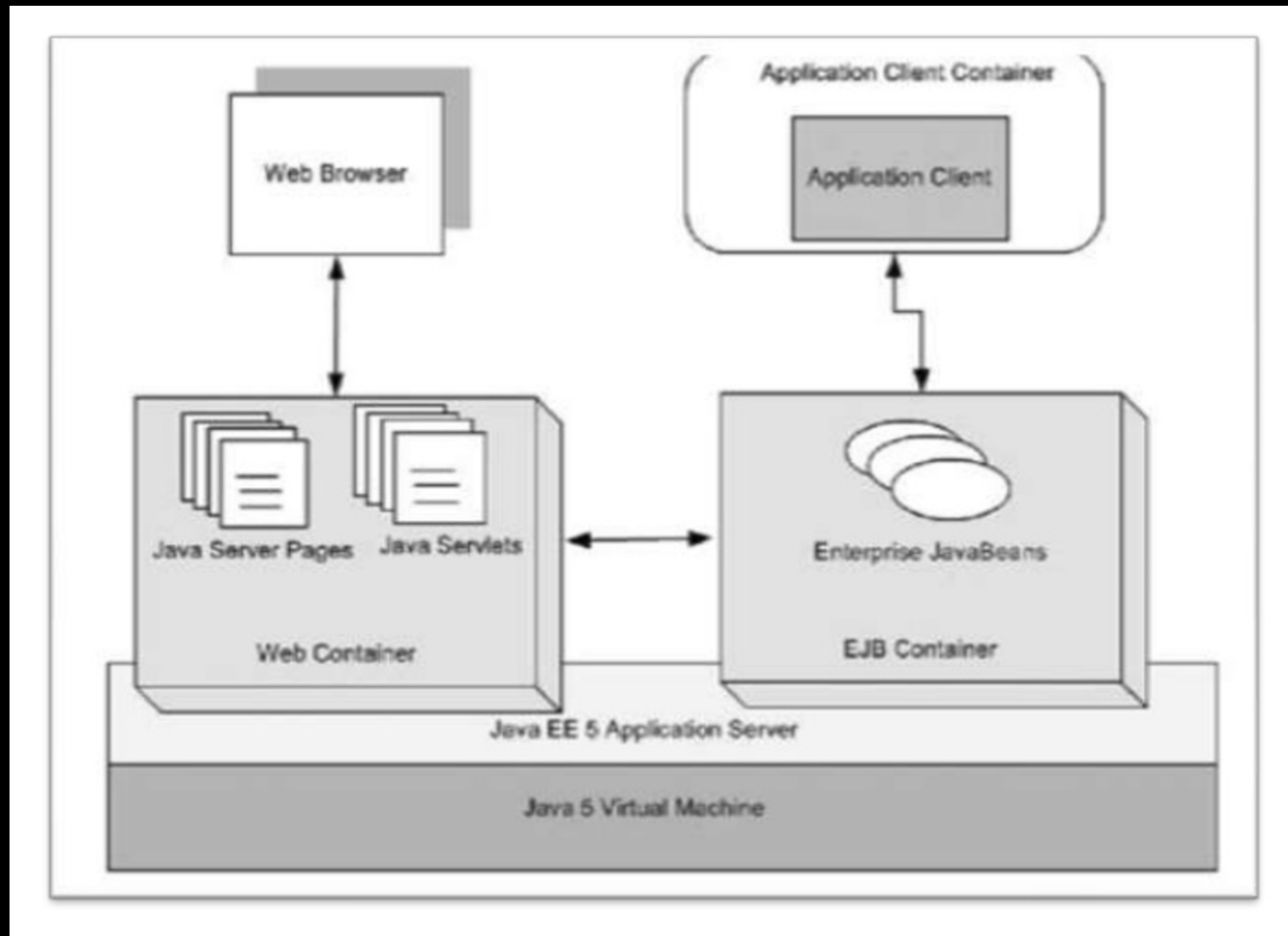
Existen dos tipos de contenedores:

Conedores Web (Servlet, JSP)

Contenedores EJB (Que administran la ejecución de componentes)



Conociendo la arquitectura de un servidor JEE



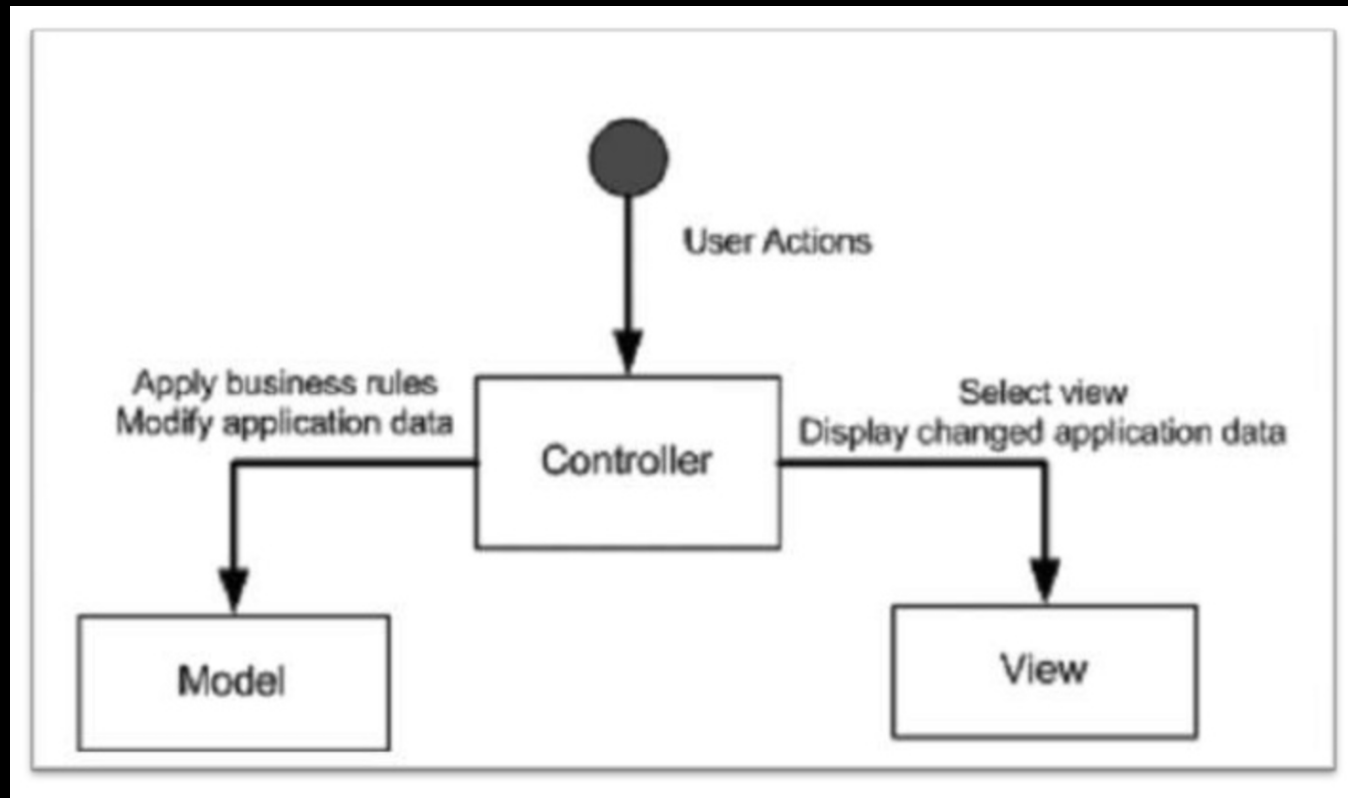
Conociendo la arquitectura de un servidor JEE

- Los contenedores brindan servicios a las aplicaciones.
- Se accede a los servicios a traves de APIs JavaEE

Gracias a esto se acelera el desarrollo y se simplifica el mantenimiento

Conociendo la arquitectura de un servidor JEE

Modelo MCV



Conociendo la arquitectura de un servidor JEE

- La tecnología Servlet sirve para construir el controlador
- Los Servlet pueden invocar los EJB
- Los datos obtenidos se pueden mostrar como JSP
- Los componentes de cada capa son facilmente manipulables

