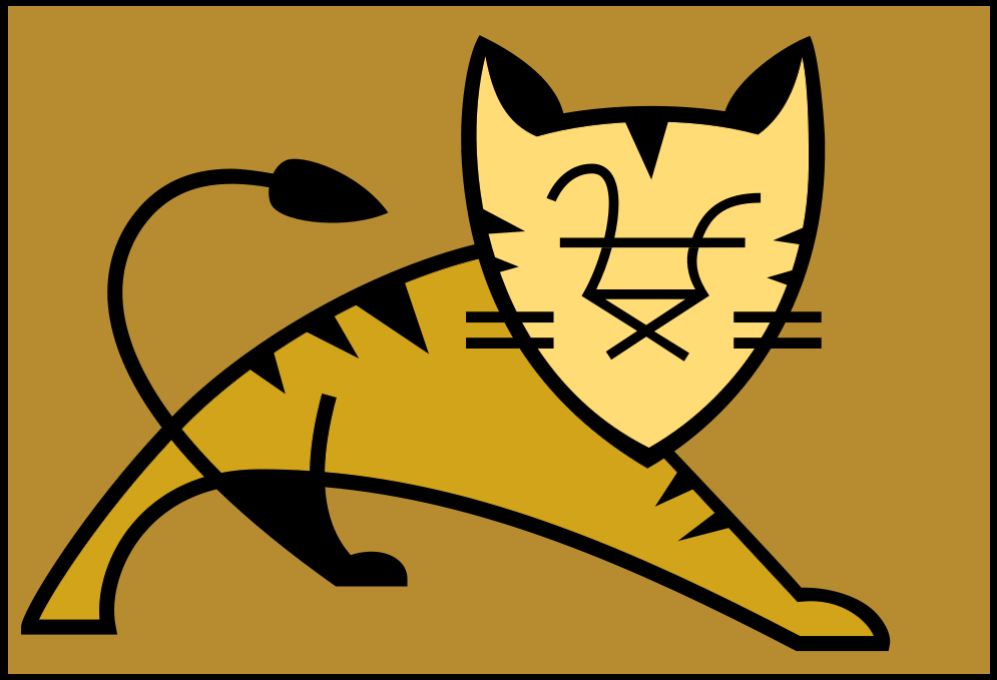
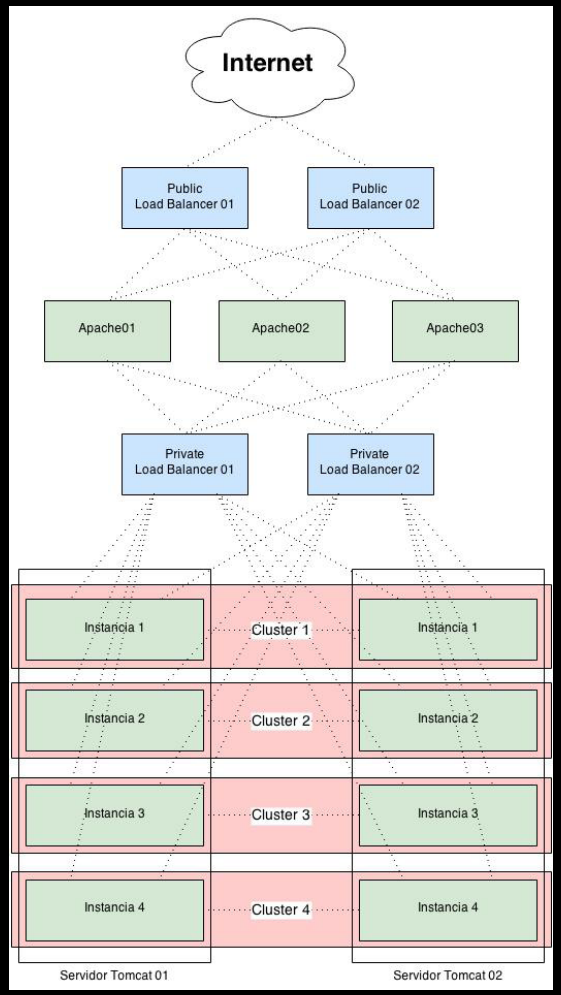
I

CONFIGURACIÓN AVANZADA DE TOMCAT



Iñaki Respaldiza

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 ademas 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 poprocionarnos 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 caracteristicas y funcionamiento varian 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 demas servidores

Configuración de los workers

instancia-01

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

instancia-02

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

instancia-03

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

instancia-04

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

Configuración de los workers

```
vi /etc/init.d/tomcat7-01

#!/bin/sh
### BEGIN INIT INFO
# Provides: Tomcat
# Required-Start: $network
# Required-Stop: $network
# Default-Start: 2 3 5
# Description: Java Servlet and JSP Engine
### END INIT INFO

JAVA_HOME=/usr/lib/jvm/java-7-oracle/
JAVA_OPTS="-Xmx800m -Xms800m"
CATALINA_HOME=/usr/local/tomcat7
CATALINA_BASE=/var/www/instancia-01

case "$1" in
'start')
$CATALINA_HOME/bin/catalina.sh start
;;
'stop')
$CATALINA_HOME/bin/catalina.sh stop
;;
*)
echo "Usage: $0 { start | stop }"
;;
```

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

```
vi /etc/apache2/mods-enabled/jk.conf
# Where to find workers.properties
JkWorkersFile /etc/libapache2-mod-jk/workers.properties

# Where to put jk logs
JkLogFile /var/log/apache2/mod_jk.log

# Set the jk log level [debug/error/info]
JkLogLevel info

JkShmFile /var/log/apache2/jk-runtime-status

# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

# JkOptions indicate to send SSL KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

JkMount /contexto_de_la_aplicacion_01/ ajp13_worker_01
JkMount /contexto_de_la_aplicacion_02/ ajp13_worker_02
JkMount /contexto_de_la_aplicacion_03/ ajp13_worker_03
JkMount /contexto_de_la_aplicacion_04/ ajp13_worker_04

JkMount /contexto_de_la_aplicacion-01/*.do ajp13_worker_01
JkMount /contexto_de_la_aplicacion-02/*.do ajp13_worker_02
JkMount /contexto_de_la_aplicacion-03/*.do ajp13_worker_03
JkMount /contexto_de_la_aplicacion-04/*.do ajp13_worker_04

# vi /etc/libapache2-mod-jk/workers.properties

worker.list=loadbalancer,stat1,ajp13_worker_01,ajp13_worker_02,ajp13_work

worker.ajp13_worker_01.port=8009
worker.ajp13_worker_01.host=192.168.0.209
worker.ajp13_worker_01.type=ajp13
worker.ajp13_worker_01.lbfactor=1

worker.ajp13_worker_02.port=8109
worker.ajp13_worker_02.host=192.168.0.209
worker.ajp13_worker_02.type=ajp13
worker.ajp13_worker_02.lbfactor=1

worker.ajp13_worker_03.port=8209
worker.ajp13_worker_03.host=192.168.0.209
worker.ajp13_worker_03.type=ajp13
worker.ajp13_worker_03.lbfactor=1

worker.ajp13_worker_04.port=8309
worker.ajp13_worker_04.host=192.168.0.209
worker.ajp13_worker_04.type=ajp13
worker.ajp13_worker_04.lbfactor=1
```

XIV

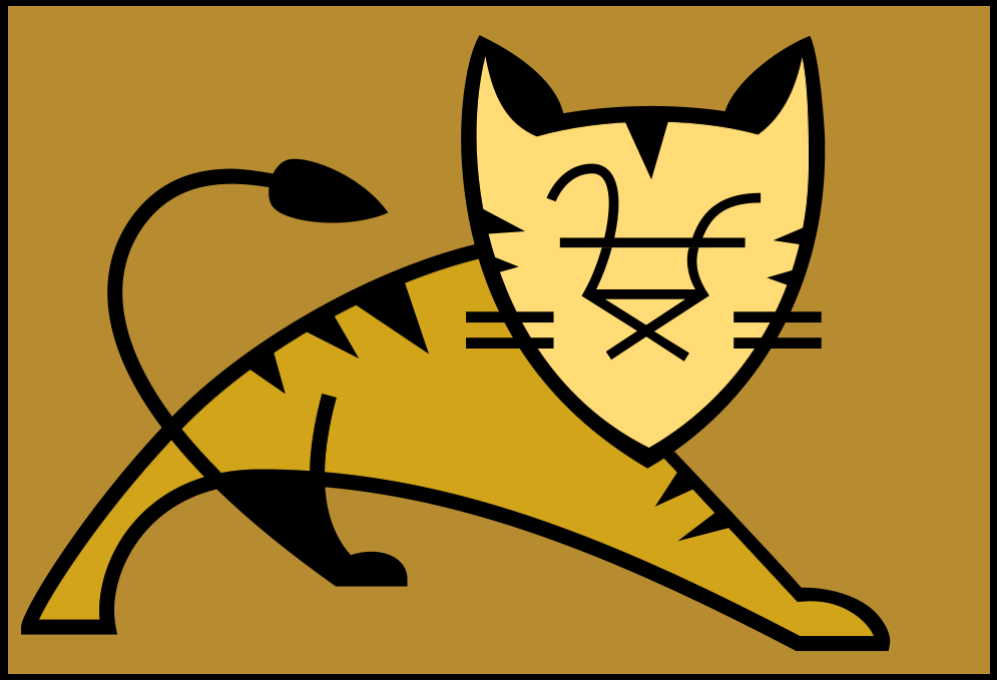
Configuración de los workers

```
vi /etc/apache2/sites-enabled/vhost-balanceo-tomcat.conf

JkMount /contexto_de_la_aplicacion-01/* ajp13_worker_01

Options -Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all

JkMount /contexto_de_la_aplicacion-02/* ajp13_worker_02

Options -Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all

JkMount /contexto_de_la_aplicacion-03/* ajp13_worker_03

Options -Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all

JkMount /contexto_de_la_aplicacion-04/* ajp13_worker_04

Options -Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
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 <*distributable* />
Usaremos una copia de nuestra aplicación en cada uno de los CATALINA_BASE de nuestros tomcats

HERRAMIENTAS DE PERFILADO



Iñaki Respaldiza

Concepto del profiling de aplicaciones

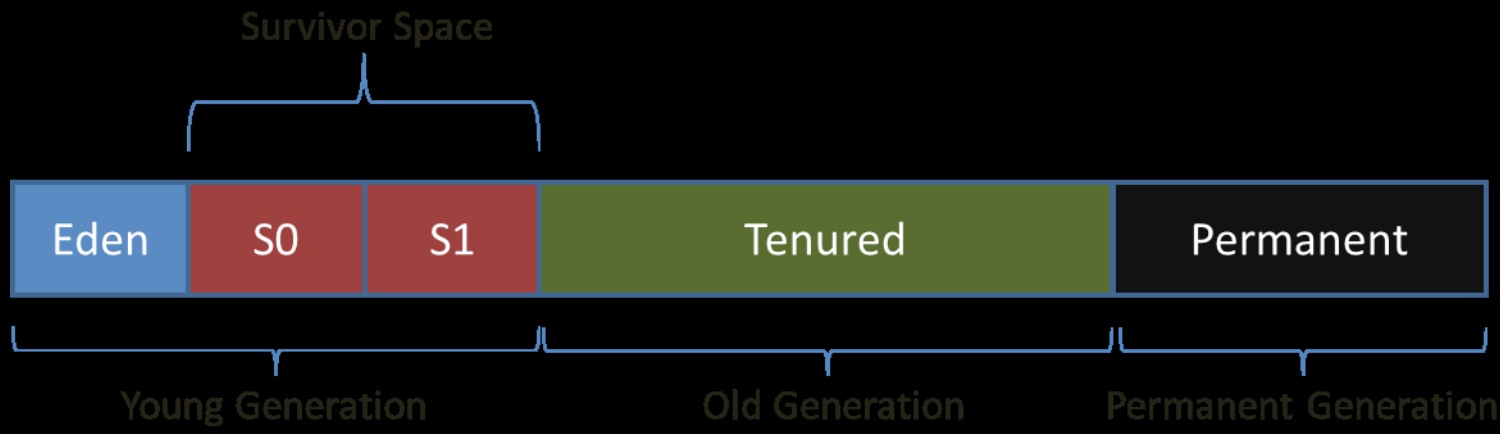
Los tests de performance son aquellos que sirven para determinar qué tan rápido o qué tan bien se comporta un sistema sometido a una carga en particular. También pueden ser utilizados para validar y verificar otros requerimientos no funcionales del sistema como ser estabilidad, escalabilidad, disponibilidad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

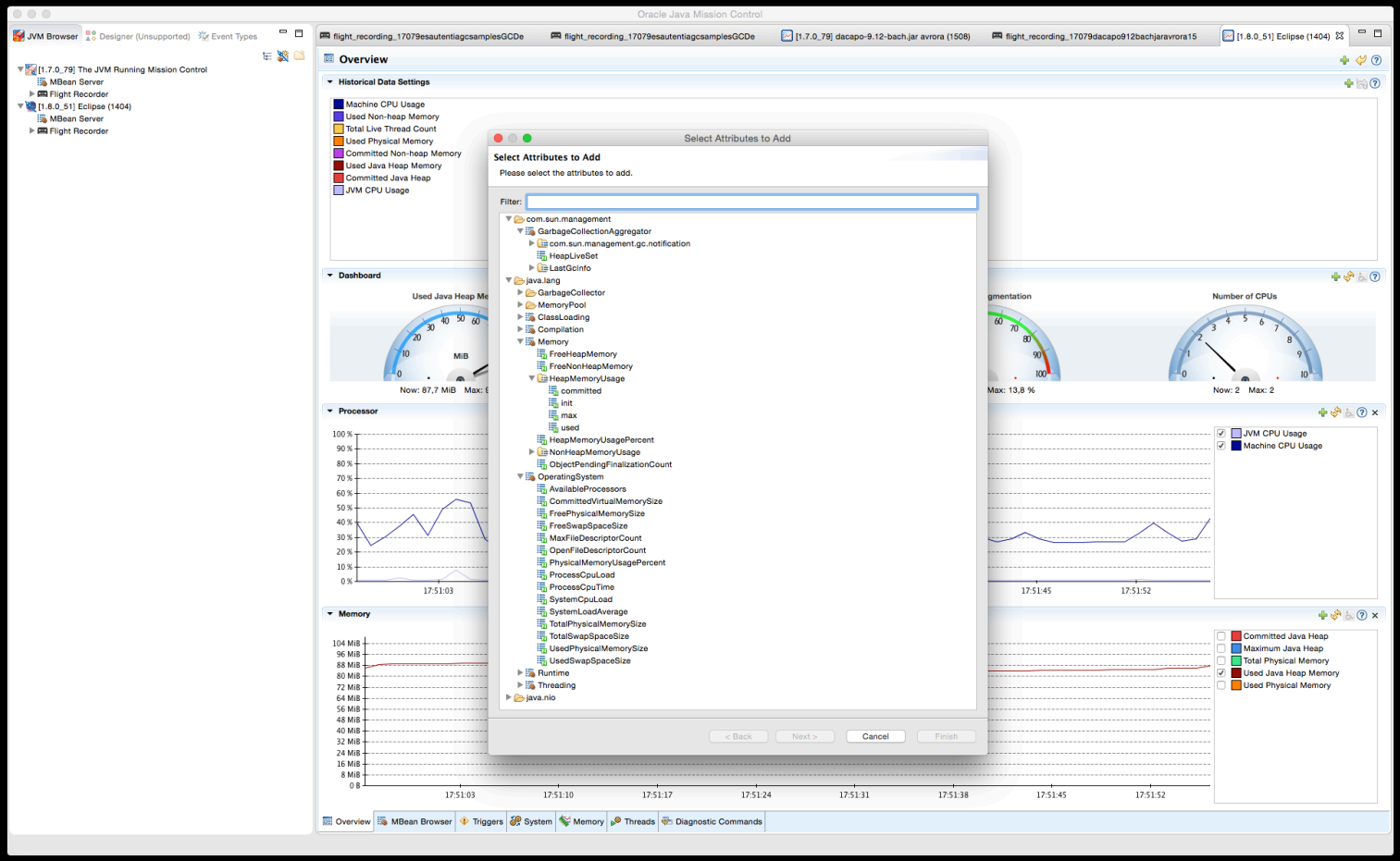
Necesidades de profiling en Java

- 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.

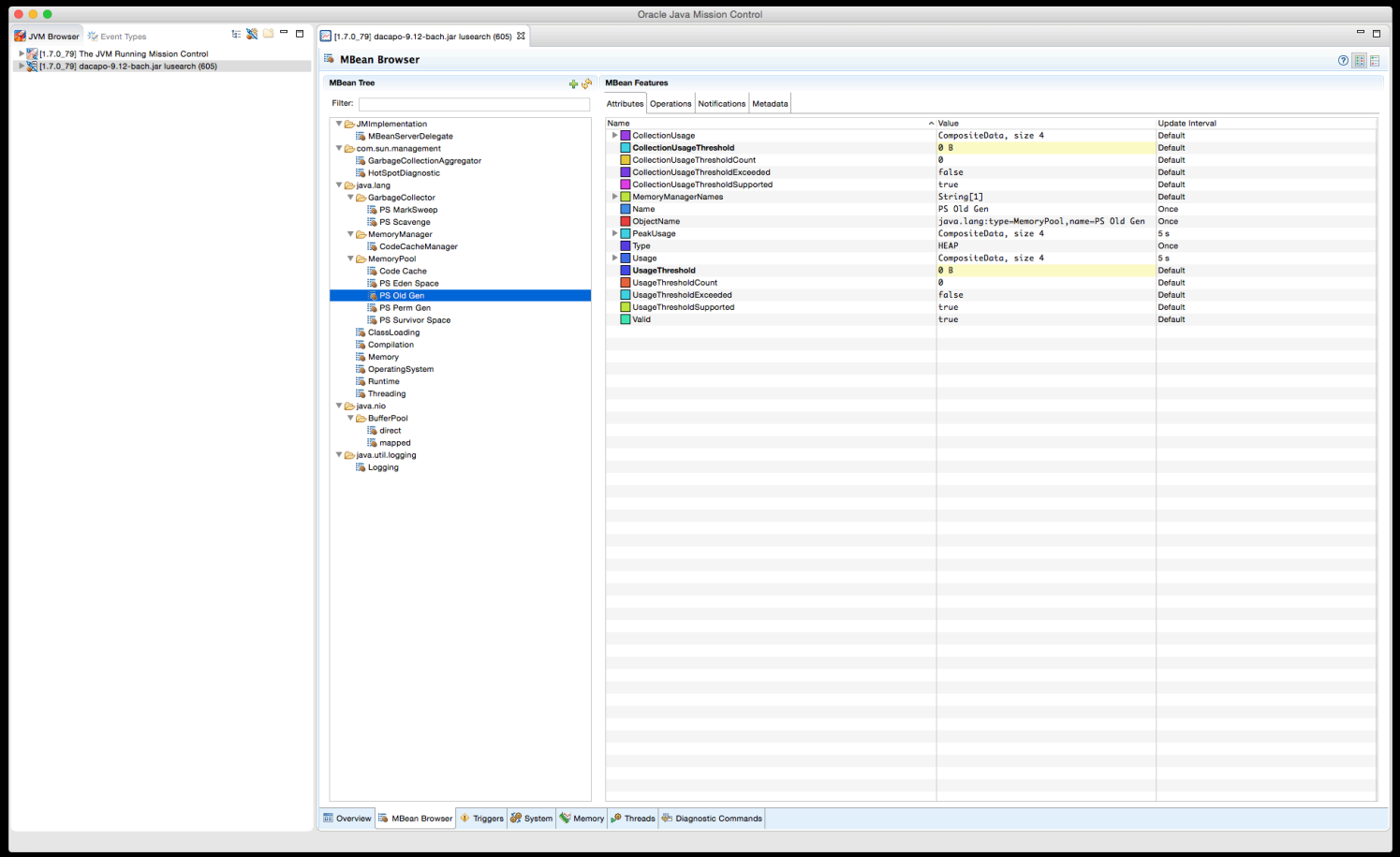
Necesidades de profiling en Java

- 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.

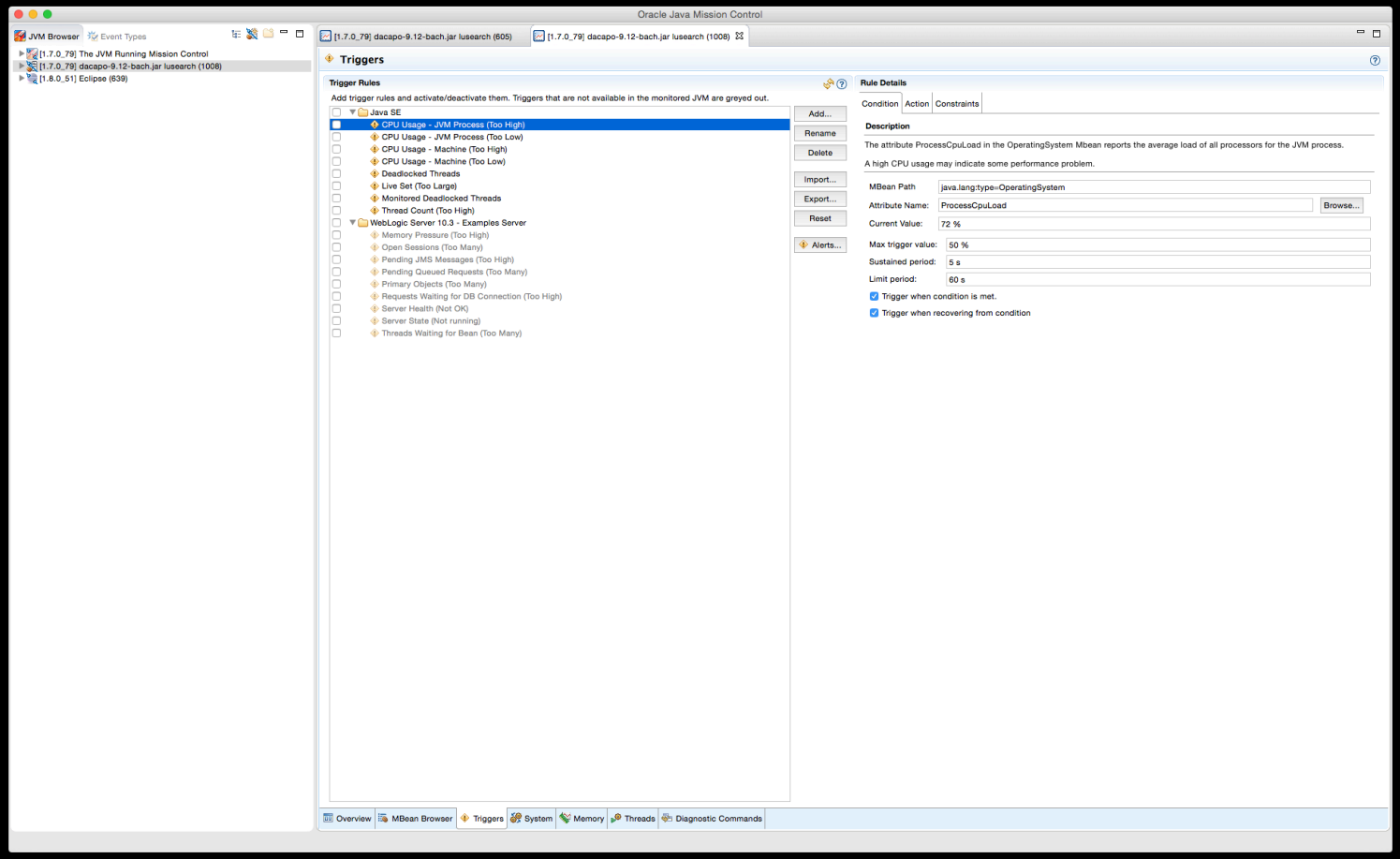
Monitorización del heap, uso de herramientas



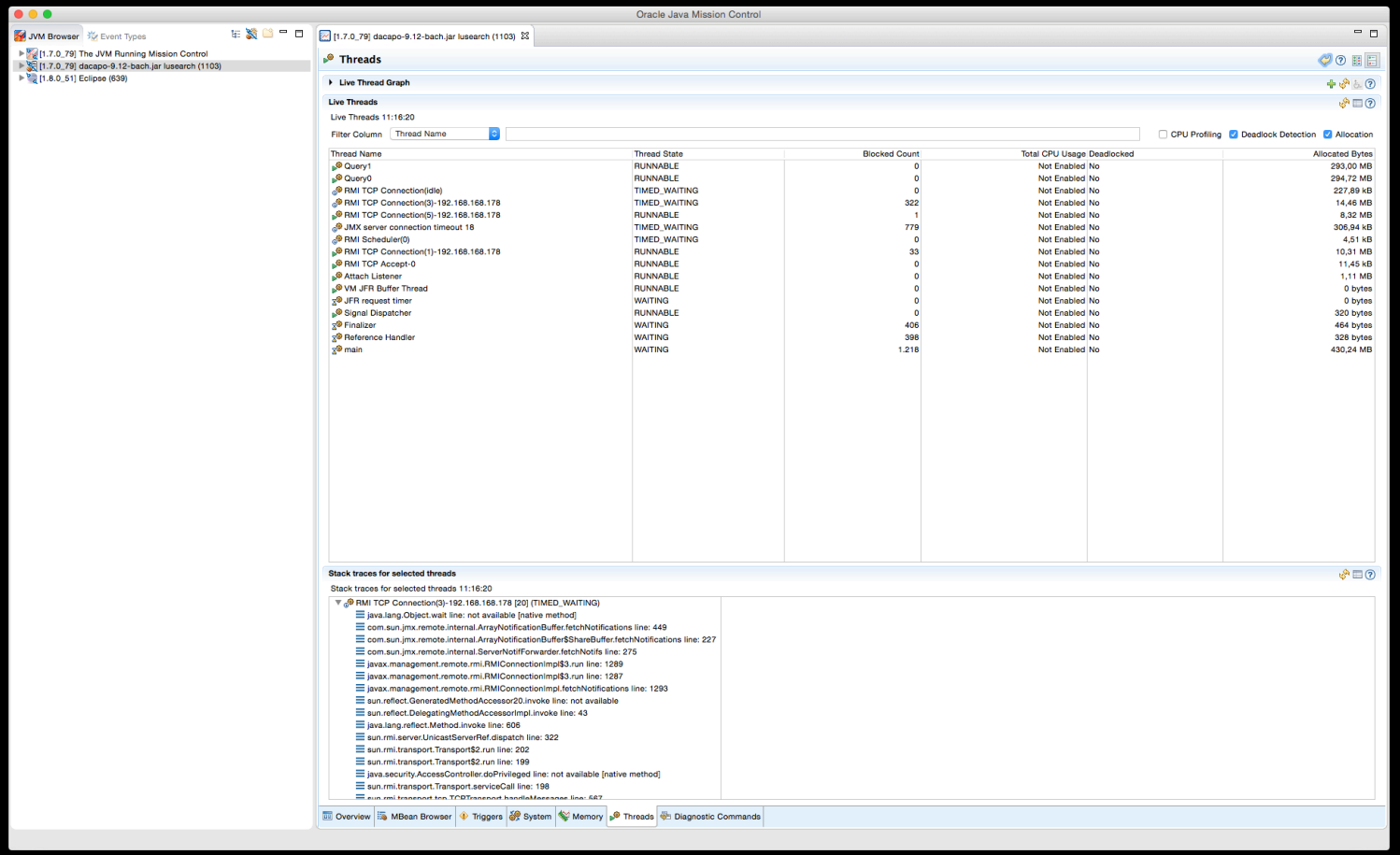
Monitorización del heap, uso de herramientas



Monitorización del heap, uso de herramientas



Monitorización del heap, uso de herramientas



Monitorización del heap, uso de herramientas

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

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

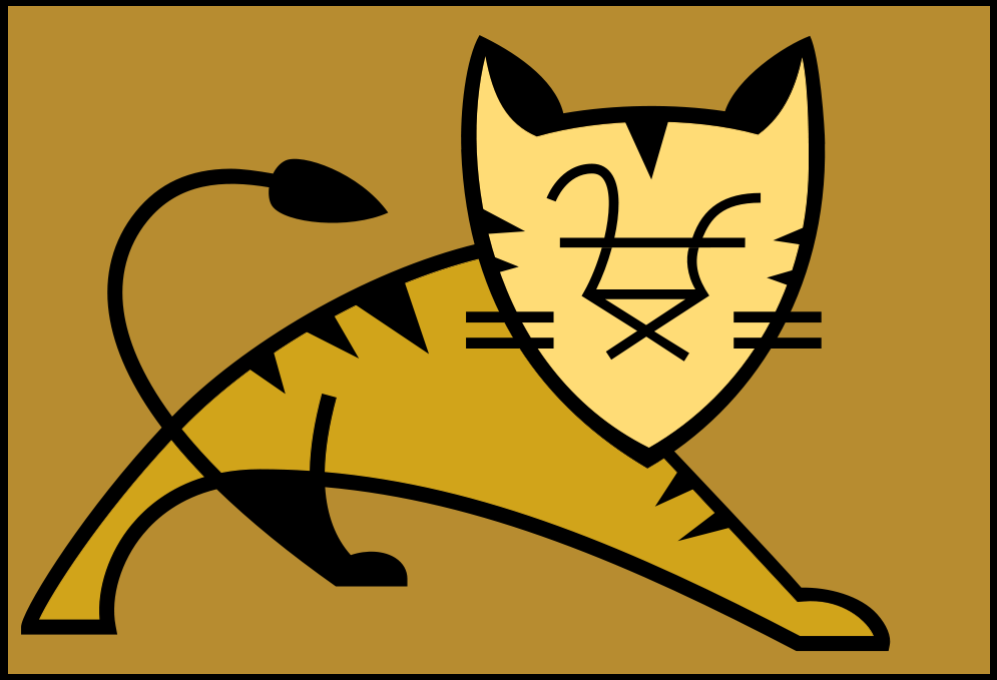
Localizando cuellos de botella en llamadas y cpu

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



Iñaki
Respaldiza

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 jóven 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 jóven 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 marcandolos 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 ergonomico 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 incomvenientes:

- 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

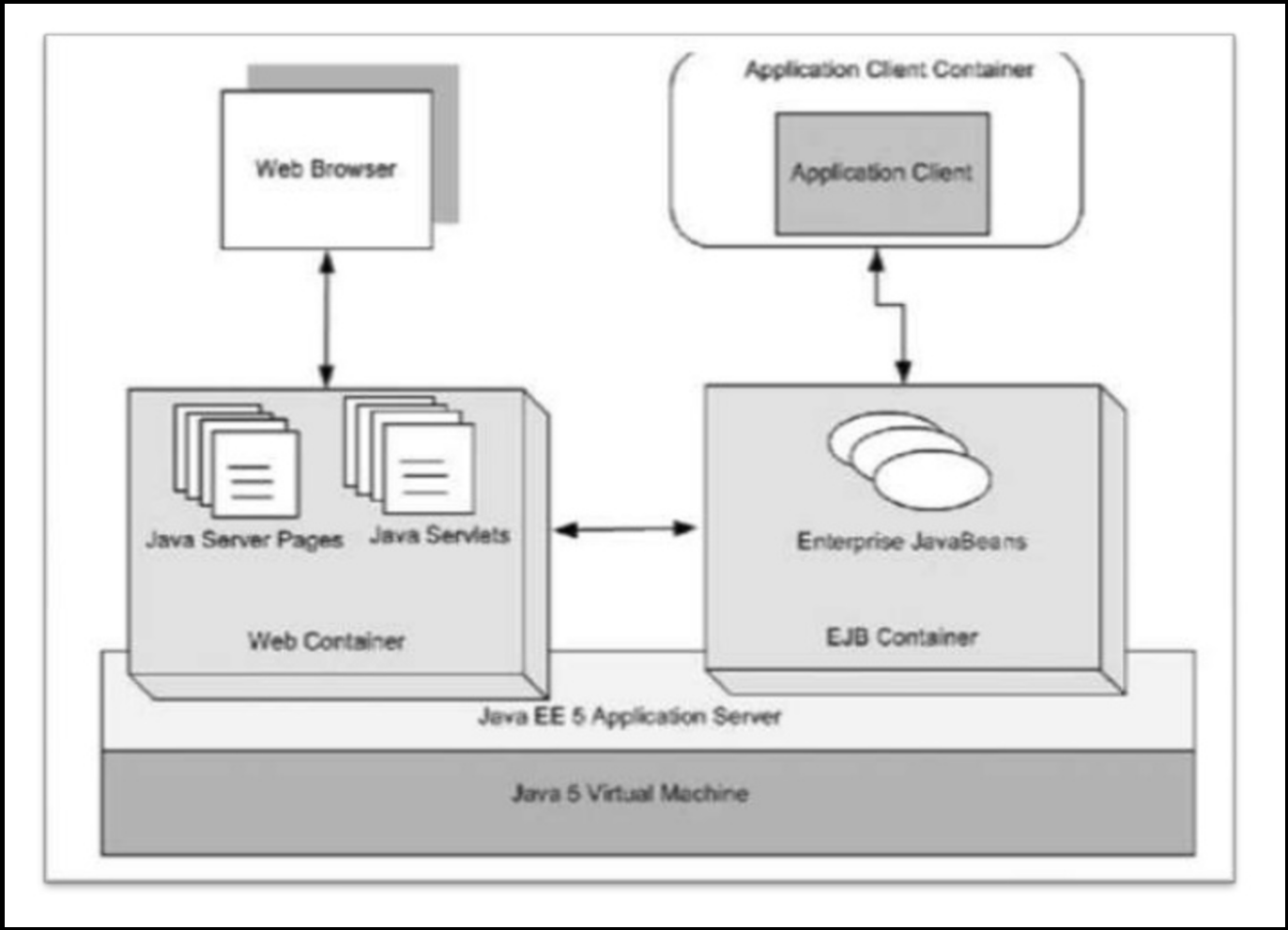
Conociendo la arquitectura de un servidor JEE

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



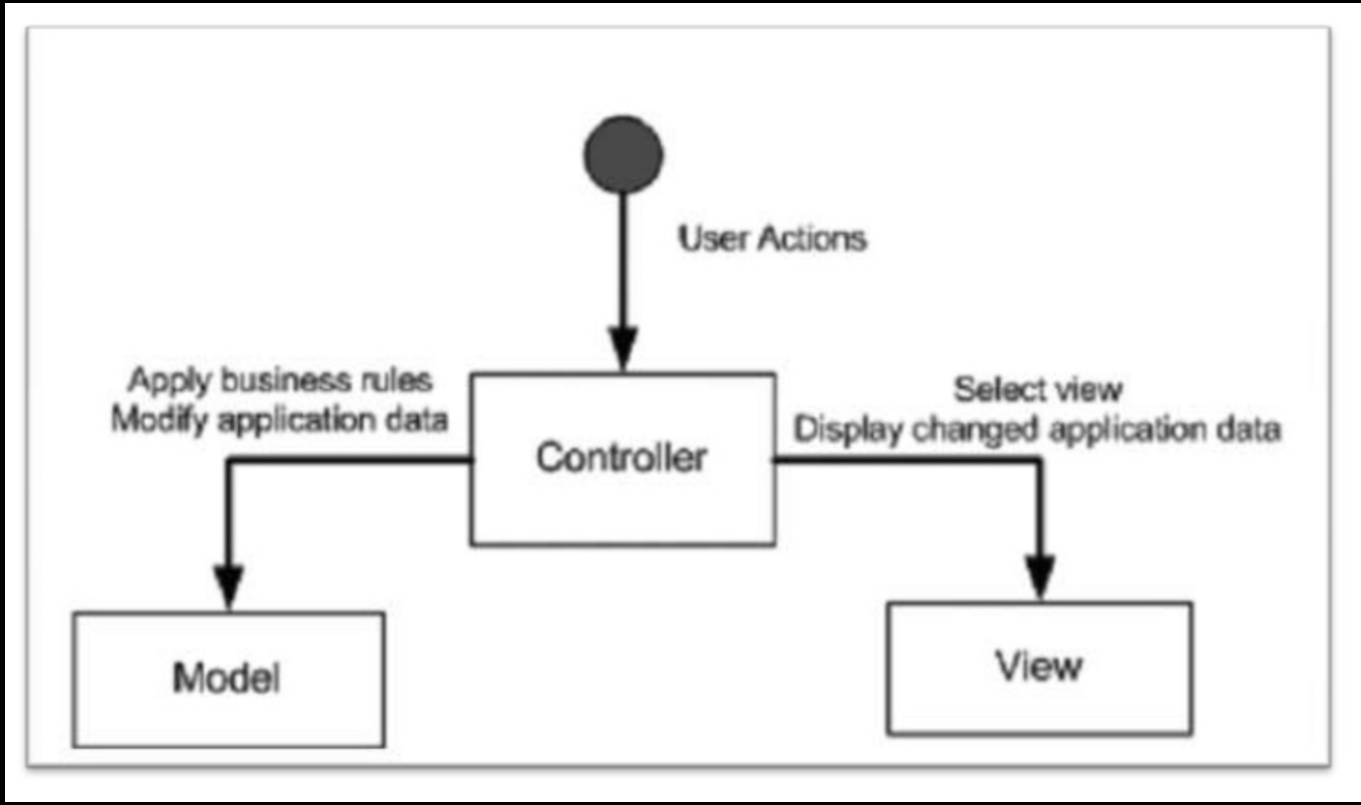
Conociendo la arquitectura de un servidor JEE

- 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 tecnologia 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