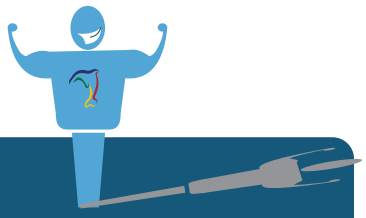
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

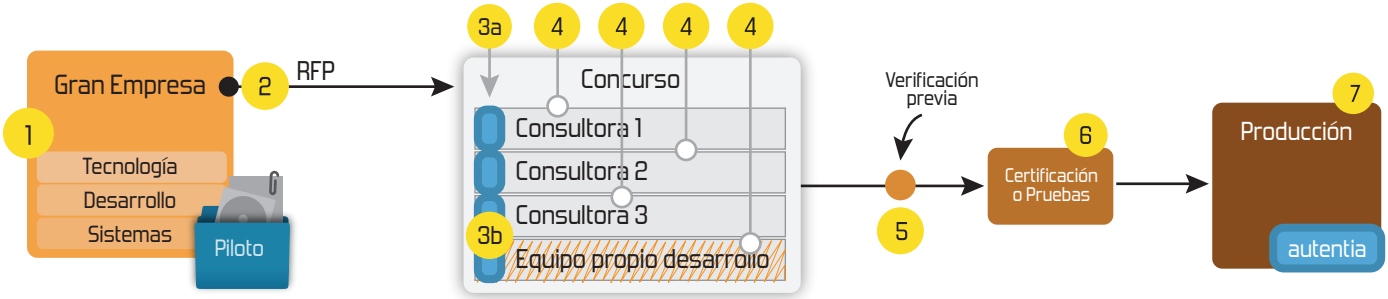
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Isaac Gutierre Gómez.

Isaac está actualmente trabajando como responsable de arquitectura de una importante empresa perteneciente a una gran entidad financiera. Ha liderado varios proyectos de aplicaciones para Internet, así como proyectos de telecomunicaciones y de IT para distintas industrias.

Contacta en: isaac@adictosaltrabajo.com

J2EE PERFORMANCE TESTING

En este artículo os vamos a mostrar como medir el rendimiento en aplicaciones J2EE. Cuando se está desarrollando una aplicació veces, no se tienen en cuenta aspectos como el rendimiento, sino que la aplicación funcione. Otras veces, la aplicación es tan compleja y tienen tantos elementos relacionados que no es fácil ver como se comporta hasta que no está en producción.

Y otras veces, cuando la aplicación ya están en producción queremos saber como se comporta realmente, qué pasa cuando existe un usuario conectado, y varios? ¿Cuántos usuarios aguanta? ¿Qué tiempos de respuesta tenemos entre las distintas capas? ¿La BBDD da buenos tiempos de respesta? Todas estas preguntas podemos resolverlas de distinta manera.

Si la aplicación está en Internet, existen multitud de webs y herramientas donde le damos las distintas urls de las páginas que queremos estudiar y el número de usuarios que queremos simular conectados y nos dan tiempos de respuesta.

Algunas herramientas de "stresss" pueden ser:

- Bean-Test de Empirix (http://www.empirix.com) en esta dirección también podemos encontrar otras herramientas de testing.
- JProbe de Quest Software (http://www.quest.com)
- Opensta *(Open System Testing Architecture)* con licencia GNU General Public License y lo podemos encontrar en (http://www.opensta.org).

Pero nosotros nos vamos a centrar en una API (JAMon Java Applicaton Monitor http://jamonapi.sourceforge.net) a parte de que e gratis, permite ver claramente los tiempos entre capas, cuando tiempo tarde en ejecutarse un determinado EJB o un JSP o un servlet. Cuanto tarda la BBDD en ejecutar cierta consulta y pasarla a la clase que le ha pedido, etc. La verdad que este API me ha agradado bastante y es muy sencilla de utilizar, y se puede utilizar con distintos servidores de aplicaciones (Sybase EAServer, BE Weblogic, Tomcat, JBoss, ...). Vamos a ello:

JAMon consta de un fichero .jar (90k) que se tiene que poner en el CLASSPATH de vuestra aplicación. Para probar la performance de JAMon podéis ejecutar el siguiente comando:

java -cp JAMon.jar com.jamonapi.TestClassPerformance 500000

Y nos tiene que aparecer algo como lo siguiente:

```
***** Performance Tests:
All performance code loops 500000 times

timingNoMonitor() - timing the old fashioned way with System.currentTimeMillis() (i.e.no monitors)
System.currentTimeMillis() - startTime
It took 281 ms.

basicTimingMonitor() - this is the most lightweight of the Monitors
BasicTimingMonitor mon=new BasicTimingMonitor();
mon.start()
mon.stop()
It took 290 ms.

NullMonitor() - Factory disabled so a NullMonitor is returned
MonitorFactory.setEnabled(false);
Monitor mon=MonitorFactory.start();
mon.stop()
It took 50 ms.

NullMonitor2() - Factory disabled so a NullMonitor is returned
MonitorFactory.setEnabled(false);
Monitor mon=MonitorFactory.start('pages.admin');
mon.stop()
It took 140 ms.
```

```
basic Factory TimingMonitor()
Monitor mon=MonitorFactory.start();
mon.stop();
It took 641 ms.

Full Factory TimingMonitor() using debug factory - uses cached version so doesn't create child monitors
Monitor mon=MonitorFactory.getDebugFactory().start('pages.admin');
mon.stop();
0 ms. (Hits=500.000 Avg=0 ms. Total=460 ms. Min=0 ms. Max=10 ms. Active=0 Avg Active=1 Max Active=1 First
access=24/12/03 21:22:58 Last access=24/12/03 21:23:00 )
It took 2.394 ms.

Full Factory TimingMonitor()- uses cached version so doesn't create child monitors
Monitor mon=MonitorFactory.start('pages.admin');
mon.stop();
0 ms. (Hits=1.000.000 Avg=0 ms. Total=930 ms. Min=0 ms. Max=10 ms. Active=0 Avg Active=1 Max Active=1 First
access=24/12/03 21:22:58 Last access=24/12/03 21:23:03 )
It took 2.293 ms.

Executing full factory monitors a second time. The second time reflects performance characteristics more accurately

Full Factory TimingMonitor() using debug factory - uses cached version so doesn't create child monitors
Monitor mon=MonitorFactory.getDebugFactory().start('pages.admin');
mon.stop();
0 ms. (Hits=1.500.000 Avg=0 ms. Total=1.490 ms. Min=0 ms. Max=10 ms. Active=0 Avg Active=1 Max Active=1 First
access=24/12/03 21:22:58 Last access=24/12/03 21:23:05 )
It took 2.263 ms.

Full Factory TimingMonitor()- uses cached version so doesn't create child monitors
Monitor mon=MonitorFactory.start('pages.admin');
mon.stop();
0 ms. (Hits=2.000.000 Avg=0 ms. Total=1.970 ms. Min=0 ms. Max=10 ms. Active=0 Avg Active=1 Max Active=1 First
access=24/12/03 21:22:58 Last access=24/12/03 21:23:07 )
It took 2.314 ms.

***** Total time for performance tests were: 10.686 ms.
```

La última etiqueta "Full Factory TimingMonitor()" nos indica cuanto tiempo tarda en ejecutarse el test cuando el "monitoring" está activado, y la etiqueta NullMonitor2() nos indica cuando tiempo tarda en ejecutarse este test cuando el monitoring está desactivad

Un ejemplo de uso es el siguiente:

```
import com.jamonapi.*;

.......

Monitor mimonitor = MonitorFactory.start("MiMonitor");

.....Código que queremos mirar el rendimiento .....

mimonitor.stop();

........

System.out.println(mimonitor);

.......
```

MonitorFactory.start("MiMonitor") crea un monitor de nombre MiMonitor, al método start() se le puede llamar con un String como parámetro, este String pude ser el nombre de un JSP, de un Servlet, de un EJB e indica el nombre del monitor del que luego se podrán consultar las estadísticas.

Cogiendo la salida de la prueba de performance del JAMon vamos a explicar como mirar las estadísticas resultantes.

0 ms. (Hits=2.000.000 Avg=0 ms. Total=1.970 ms. Min=0 ms. Max=10 ms. Active=0 Avg Active=1 Max Active=1 First access=24/12/03 21:22:58 Last access=24/12/03 21:23:07)
It took 2.314 ms.

O ms: Es el tiempo de ejecución en milisegundos.

Hits=2.000.000. Un hit ocurre cuando el método start() es llamado con el nombre del monitor idéntico. En el ejemplo anterior "MiMonitor" es el nombre del monitor.

Avg=0 ms. Es la media del tiempo de ejecución total divido por los hits.

Total=1.970ms. Es el tiempo total acumulado para todos los 10 monitores con el mismo nombre.

Min=0 ms. Es el mínimo tiempo de ejecución para 10 hits.

Max=10 ms. Es el máximo tiempo de ejecución para 10 hits.

Active=0. Esta etiqueta tiene sentido en entornos multithread. Indica el número de monitores en ejecución simultánea con el mismo nombre. Es muy útil para medir el tiempo de respuesta de JSPs, Servlets cuando muchos usuarios están accediendo a la misma página.

Avg Active=1. Indica la media del número de monitores en ejecución simultánea.

Max Active=1. Indica el máximo número de ejecuciones simultáneas del mismo monitor.

First Access=24/12/03 21:22:58. Indica cuando un monitor empieza su ejecución.

Last access=24/12/03 21:23:07. Indica cuando un monitor ha tarminado su ejecución.

Cómo podemos ver con JAMon no sólo vemos podemos sacar estadísticas de tiempo en la ejecución, sino que con las etiquetas "Active" podemos ver cuantos usuarios simultáneos hay en la aplicación y también podemos detectar errores de programación.

Espero que os haya ayudado a ver si vuestras aplicaciones son óptimas. En un siguiente artículo mostraremos como podemos corregir cuellos de botella y qué acciones tomar cuando se trabaja con J2EE para hacer aplicaciones con un rendimiento más que aceptable.

Respecto al autor
Isaac Gutiérrez está actualmente trabajando como responsable de arquitectura de una importante empresa perteneciente a una gran entidad financiera. Ha liderado varios proyectos de aplicaciones para Internet, así como proyectos de telecomunicaciones y IT para distintas industrias.

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Optimización de Serialización Java	Os mostramos una sencilla técnica para mejorar el rendimiento de la serialización de objetos en Java, a través de Streams asociados a buffers en memoria.
Temperatura del PC y Acoplamiento Electromagnético	Os mostramos algunos programas para ver en pantalla la temperatura de vuestra CPU y velocidad de ventiladores así como consejos de como proteger a vuestro PC de campos electromagnéticos.

Test con JUnit	Cuando se hacen desarrollo profesionales, no basta con hacer los programas, hay que asegurarse de que van a funcionar. Una de las técnicas más seguras es crear aplicaciones que incluyan el código para autoprobarse. Os mostramos como usar JUnit
Rendimiento de aplicaciones Web	En este tutorial veremos una introducción al funcionamiento de la Suite e-Test de Empirix.
Administrar UNIX mediante Webmin	En este artículo os mostramos como administrar una máquina UNIX, mediante un interfaz web, con Webmin
Gestión errores en JSPs	Os mostramos como realizar ciertas labores intermedias en JSPs: Comentarios, gestión de errores, formateo de fechas y precompilación de ficheros
Despliegue gráfico de EJBs	Os mostramos como crear y desplegar de un modo gráfico un EJB de sesión el el servidor de aplicaciones de referencia de Sun
Compartir impresoras y ficheros con Linux	Cesar Crespo Martín y Alejandro Perez García nos enseñan como realizar la compartición de impresoras y ficheros con Linux, CUPS y SAMBA con clientes Windows.

www.AdictosAlTrabajo.com Opimizado 800X600