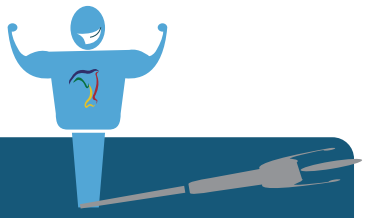
autentia

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...

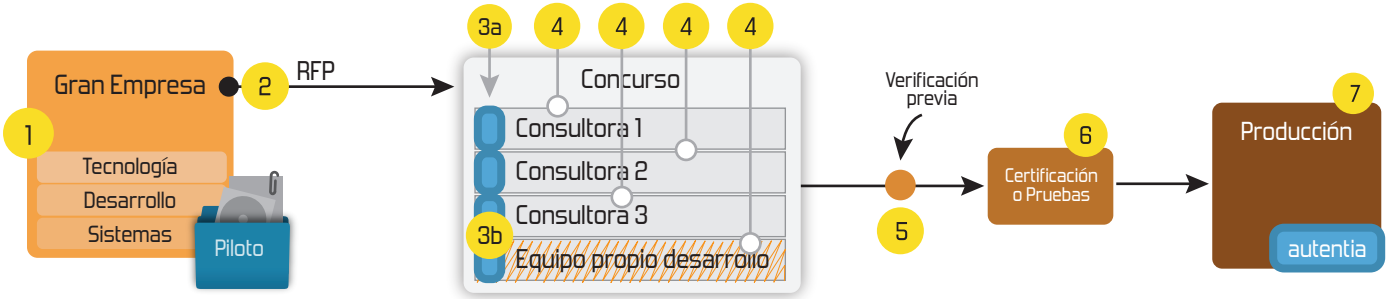
1. Desarrollo de componentes y proyectos a medida



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 conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Jose Manuel Sánchez Suárez

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en:
jmsanchez@autentia.com

Fecha de creación del tutorial: 2007-12-12

Ejb3 Timer Service: scheduling.

0. Índice de contenidos.

1. Introducción

Las tareas programadas son una realidad en las aplicaciones JEE: tablas que deben historificar, alarmas que se deben generar,. normal puede ser la definición de un proceso batch que se invoca periódicamente mediante el planificador de tareas de Windows en UNIX, a través del cron, o de alguna solución comercial tipo control-M.

Ya hemos visto, en adictos, cómo planificar tareas en Jboss de mano de Francisco Javier Martínez Páez, quién me dio, a su vez, idea sobre éste tutorial.

Ahora vamos a ver cómo planificar una tarea usando la solución estándar de JEE: EJB Timer Services.

El principal beneficio de su uso es que, al formar parte de estandar, no nos ligamos a un servidor de aplicaciones concreto.

EJB3 Time Service permite especificar un método que es invocado automáticamente después de un determinado intervalo de tiempo.

Vamos a ver un ejemplo de definición de un EJB3 Timer Service, que invoca periódicamente, una tarea ficticia, la generación alg alarma de sistema.

Se da por hecho que el lector conoce lo básico sobre ejb3 y sus anotaciones.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Sobremesa Dell Dimension 6400, 2.13 Ghz, 2 Gb RAM
- Sistema operativo: Windows XP Media center Edition
- JDK 1.6.0_2
- Eclipse 3.3.
- Jboss 4.2.1.GA

3. Creación de un TimerService.

La implementación de un EJB Timer Service es realmente sencilla. Hablamos de EJB3, lo que implica hablar de anotaciones e injección de servicios:

view plain print ?

```
01. package com.autentia.tutorial.ejb.scheduler;
02.
03. import java.util.Calendar;
04. import java.util.Collection;
05. import java.util.Iterator;
06.
07. import javax.annotation.Resource;
08. import javax.ejb.Stateless;
09. import javax.ejb.Timeout;
10. import javax.ejb.Timer;
11. import javax.ejb.TimerService;
12.
13. import org.apache.commons.logging.Log;
14. import org.apache.commons.logging.LogFactory;
15.
16. /**
17.  * Implementación del servicio @local IAlarmScheduler.
18.  */
19. @Stateless
20. public class AlarmScheduler implements IAlarmScheduler {
21.
22.     private static final Log log = LogFactory.getLog(AlarmScheduler.class);
23.
24.     /** Injección del TimerService */
25.     @Resource TimerService timerService;
26.
27.     /** Hora de ejecución: 23 horas */
28.     private static final int START_HOUR = 23;
29.
30.     /** Minutos de ejecución: 0 minutos */
31.     private static final int START_MINUTES = 0;
32.
33.     /** Segundos de ejecución: 00  */
34.     private static final int START_SECONDS = 0;
35.
36.     /** Intervalo de la ejecución: 1440 = 24 horas */
37.     private static final int INTERVAL_IN_MINUTES = 1440;
38.
39.     /**
40.      * Levanta el servicio
41.      */
42.     public void startUpTimer() {
43.
44.         log.info("startUpTimer - alarm scheduler service is active.");
45.
46.         shutDownTimer();
47.
48.         Calendar initialExpiration = Calendar.getInstance();
49.         initialExpiration.set(Calendar.HOUR_OF_DAY, START_HOUR );
50.         initialExpiration.set(Calendar.MINUTE, START_MINUTES);
51.         initialExpiration.set(Calendar.SECOND, START_SECONDS);
52.
53.         long intervalDuration = new Integer(INTERVAL_IN_MINUTES).longValue()*60*1000;
54.
55.         log.info("startUpTimer - create new timer service at \""+initialExpiration.getTime()+"\", with \""+intervalDuration-
56.         timerService.createTimer(initialExpiration.getTime(),intervalDuration,null);
57.
58.     }
59.
60.     /**
61.      * Para el servicio
62.      */
63.     public void shutDownTimer() {
64.         Collection<Timer> timers = timerService.getTimers();
65.         log.info("shutDownTimer - existing timers? " + timers);
66.         if (timers != null)
67.         {
68.             for (Iterator iterator = timers.iterator(); iterator.hasNext();) {
69.                 Timer t = (Timer) iterator.next();
70.                 t.cancel();
71.                 log.info("shutDownTimer - timer \""+t+"\" canceled.");
72.             }
73.         }
74.     }
75.
76.     /**
77.      *  método callback que se invocará al terminar el intervalo definido
78.      */
79.     @Timeout
80.     public void execute(Timer timer)
81.     {
```

Explicamos el código:

- Injectamos el TimerService en la línea 23: **@Resource TimerService timerService;**
- En la línea 46, creamos el temporizador (**timerService.createTimer(...);**) con dos parámetros: ejecución inicial (las 23:00:00 horas) e intervalo (cada 24 horas).
- El método **shutDownTimer()** lista todos los temporizadores y acaba con ellos uno a uno. El hecho de parar el servidor de aplicaciones no implica el borrado de los temporizadores, con lo que si el método startUpTimer no invocase al método shutDownTimer antes de crear un nuevo temporizador, se irían acumulando.
- Por último, el método execute está marcado con la anotación **@Timeout**. Será el método que se invoque a la hora señalac después de finalizar cada intervalo.

Para poder acceder al servicio de temporizador debemos crear una interfaz, local o remota, en función de nuestras necesidades, publique al menos el método que crea el temporizador:

view plain print ?

```
01. package com.autentia.tutorial.ejb.scheduler;
02.
03. import javax.ejb.Local;
04.
05. @Local
06. public interface IAlarmScheduler{
07.
08.     /** starts the timer */
09.     public void startUpTimer();
10.
11.     /** stops all the timers */
12.     public void shutDownTimer();
13.
14. }
```

4. Levantando el servicio.

Para levantar el servicio debemos invocar al método startUpTimer de la interfaz local. Pueden existir varias formas de hacerlo, nosotros vamos a invocarlo a través de un servlet de inicialización.

Definimos un servlet en nuestro web.xml, que se cargue en el arranque del servidor:

view plain print ?

```
01. <servlet>
02.     <servlet-name>Initialize</servlet-name>
03.     <servlet-class>com.autentia.tutorial.web.servlets.InitAppServlet</servlet-class>
04.     <load-on-startup>1</load-on-startup>
05. </servlet>
```

El código del servlet de inicialización que implementa la llamada al servicio local vendría a ser el siguiente:

view plain print ?

```
01. package com.autentia.tutorial.web.servlets;
02.
03. import javax.naming.InitialContext;
04. import javax.naming.NamingException;
05. import javax.servlet.ServletException;
06. import javax.servlet.http.HttpServlet;
07. import javax.servlet.http.HttpServletRequest;
08. import javax.servlet.http.HttpServletResponse;
09.
10. import org.apache.commons.logging.Log;
11. import org.apache.commons.logging.LogFactory;
12.
13. import com.autentia.tutorial.ejb.scheduler.AlarmScheduler;
14. import com.autentia.tutorial.ejb.scheduler.IAlarmScheduler;
15.
16.
17. /**
18.  * A servlet that will be called the very first when Application Server is started.
19.  */
20. public class InitAppServlet extends HttpServlet
21. {
22.     private static final long serialVersionUID = 7040095709523857004L;
23.
24.     private static final Log log = LogFactory.getLog(InitAppServlet.class);
25.
26.     private static final String jndiPrefix = "";
27.
28.     IAlarmScheduler alarmScheduler;
29.
30.     private static final InitialContext ctx;
31.     static {
32.         try {
33.             ctx = new InitialContext();
34.         } catch (NamingException e) {
35.             log.fatal("It is not possible to create a new InitialContext.", e);
36.             throw new RuntimeException(e);
37.         }
38.     }
39.
40.     public void init() throws ServletException
41.     {
42.         log.info("InitAppServlet - init");
43.
44.         try {
45.             alarmScheduler =  (IAlarmScheduler) ctx.lookup(jndiPrefix + AlarmScheduler.class.getSimpleName() + "/local
46.         } catch (NamingException e) {
47.             log.error("InitAppServlet - NamingException",e);
48.         }
49.
50.         log.info("InitAppServlet - starting alarm scheduling notification.");
51.
52.         // start up alarm scheduler
53.         alarmScheduler.startUpTimer();
54.     }
55.
56.     protected void doGet( HttpServletRequest request, HttpServletResponse response ) throws ServletException
57.     {
58.         // do nothing, only for initialization purposes
59.     }
60.
61. }
```

Solo voy a hacer hincapié en la línea 53, la invocación al método de inicialización del Timer Service.

Doy por hecho que el resto del código se entiende (la variable jndiPrefix contendría el "nombre_de_ear.ear/" en el caso de desplegar la aplicación bajo un ear), y que tenemos definidas las propiedades de acceso al servicio vía jndi en un fichero de recursos (jndi.properties), visible desde el classLoader de la aplicación web:

view plain print ?

```
01. java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
02. java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
03. java.naming.provider.url=localhost:1099
```

Arrancando la aplicación en nuestro jboss:

- el EJB Timer Service: debe estar correctamente registrado:

Global JNDI Namespace

```
+- TopicConnectionFactory (class: org.jboss.naming.LinkRefPair)
+- jmx (class: org.jnp.interfaces.NamingContext)
|   +- invoker (class: org.jnp.interfaces.NamingContext)
|   |   +- RMIAdaptor (proxy: $Proxy48 implements interface org.jboss.jmx.adaptor.rmi.RMIAdaptor,interface o
|   +- rmi (class: org.jnp.interfaces.NamingContext)
|   |   +- RMIAdaptor[link -> jmx/invoker/RMIAdaptor] (class: javax.naming.LinkRef)
+- AlarmScheduler (class: org.jnp.interfaces.NamingContext)
|   +- local (proxy: $Proxy67 implements interface com.autentia.tutorial.ejb.scheduler.IAlarmScheduler,inter
```

- y debemos tenerlo activo... "esperando que den las 23:00:00 horas".

JMX MBean Operation Result listTimerHandles()

Bean View Reinvoke MBean Operation

```
2ee:service=EJB3,jar=SimpleEJB.jar,name=AlarmScheduler],first=02-dic-2007 23:00:00.265,periode=86400000]]
```

5. Conclusiones.

Perfecto para planificar tareas programadas con un coste mínimo, puesto que delegamos su ejecución al contenedor de ejb's de servidor de aplicaciones.

Por ahora no tenemos una herramienta con una interfaz gráfica de administración de dichas tareas, con lo que si el volumen de tareas es elevado quizás sea un poco tedioso administrarlas.

En función de nuestras necesidades, debemos ser nosotros los que decidamos si utilizamos el estándar o acudimos a aplicacione terceros para su gestión.

Un saludo.

Jose Manuel

Autentia

Puedes opinar sobre este tutorial aquí

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**
EJB 3.0: Resurrection	Este turorial nos va a presentar las nuevas funcionalidades que nos aportan los EJB 3.0.
Interceptando un EJB en JBoss	En este tutorial os vamos a enseñar la arquitectura de EJBs en JBoss y a como modificarla, insertando un interceptor propio dentro de la cadena de interceptores del Proxy Cliente.
EJB´s y Orion	Recreación de la guía paso a paso de como crear una aplicación Web con EJB´s y Servlets y su despliegue con ANT sobre Orion
EJB 3.0, un ejemplo práctico con Maven y JBoss	Este tutorial presenta un ejemplo sencillo donde se verá como desarrollar EJBs de sesión y de entidad, inyección de dependencias, llamar a los EJBs desde una aplicación Web, definición de un DataSource, y como configurarlo y hacerlo funcionar en JBoss, y
Hibernate y las anotaciones de EJB 3.0	En este tutorial Alejandro Pérez nos muestra las ventajas que nos aporta Hibernate y las anotaciones de EJB 3.0
EJB 3.0 y pruebas unitarias con Maven, JUnit y Embedded JBoss	En este tutorial Alejandro Pérez nos enseña como realizar test unitarios sobre EJB 3.0. Para ello se usará Maven, JUnit y Embedded JBoss
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
Comparativa entre EJB3 y Spring	En este tutorial os mostramos una comparativa entre EJB3 y Spring esperando que os ayude a decidir qué tecnología utilizar.
Comparativa entre Hibernate y EJB3 en la Capa de Persistencia	El presente documento pretende dar algunas luces a la comparativa entre la opción de usar Hibernate y/ó EJB3 para la capa de persistencia
Anotaciones en EJB 3.0	Este tuotiral nos va a enseñar algunas características del API de EJB 3.0 y las mejoras introducidas en la nueva version 3.0