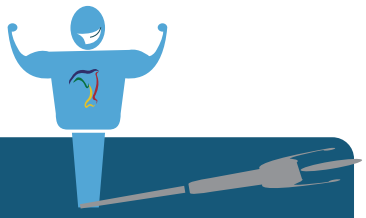
autentia

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...

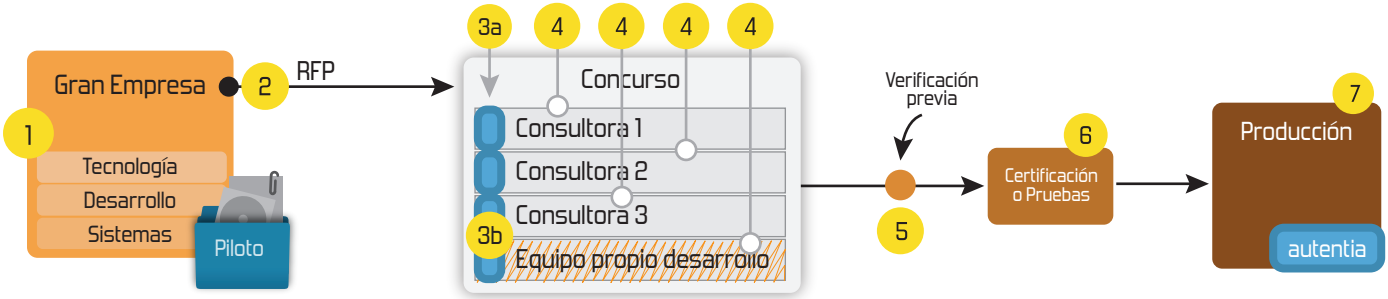
1. Desarrollo de componentes y proyectos a medida



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

DESARROLLADO POR:
Juan Alonso Ramos

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Informática de Gestión e Ingeniero en Informática, especialidad en Ingeniería del Software

Puedes encontrarme en Autentia: Ofrecemos de servicios soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver tutoriales de Juan Alonso Ramos

Fecha de publicación del tutorial: 2011-12-27

Ejecución de tareas asíncronas y planificadas con Spring

Índice de contenidos.

1. Introducción

Una vez más gracias a Spring, de forma muy sencilla podemos ejecutar tareas de forma asíncrona. Tareas que no necesitan de una espera hasta que terminan o tareas que requieren mucho tiempo para ser ejecutadas se deben lanzar en un thread independiente para poder seguir con la ejecución de la aplicación sin esperar a que termine la tarea. Para gestionar la creación y finalización de threads, Spring nos facilita la tarea aportándonos adicionalmente mucha más funcionalidad.

En este tutorial veremos algunos ejemplos de código utilizando varias de las funcionalidades que nos proporciona Spring. Si quieres puedes descargarte el código fuente desde aquí.

2. Entorno

- MacBook Pro 15' (2.4 GHz Intel Core i5, 4GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Snow Leopard 10.6.8

- JDK 1.6.0_29
- Spring 3.0.5

3. Tareas asíncronas

Para empezar vamos a programar un servicio que se encargue de realizar un conjunto de tareas que requieran bastante tiempo en completar y que nos de algo de juego para el ejemplo. Como estas tareas no son el objetivo del tutorial, sino que lo interesante es ver la manera que tiene Spring de ejecutarlas de forma asíncrona, la clase simulará envíos de correo. En realidad no se enviarán correos sino que ejecutará la acción simulando que tardará 1 segundo y continuará con el siguiente correo.

Lo primero será crear un proyecto nuevo, en mi caso lo he creado con Maven, añadirle las dependencias de Spring y crear un applicacionContext-test.xml configurando la propiedad Spring task:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03     xmlns:context="http://www.springframework.org/schema/context"
   xmlns:p="http://www.springframework.org/schema/p"
04     xmlns:task="http://www.springframework.org/schema/task"
05     xsi:schemaLocation="
06         http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd
07         http://www.springframework.org/schema/util
   http://www.springframework.org/schema/util/spring-util.xsd
08         http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context.xsd
09         http://www.springframework.org/schema/task
   http://www.springframework.org/schema/task/spring-task.xsd">
10
11     <context:annotation-config />
12
13     <context:component-scan base-package="com.autentia.tutoriales" />
14
15     <task:annotation-driven />
16
17 </beans>
```

Las líneas 11 y 13 indican a Spring que la configuración se define mediante anotaciones y que se debe escanear en el paquete indicado. La línea 15 es la que nos sirve para configurar Spring para que levante en hilos diferentes los métodos definidos con la anotación @Async. Lo vemos en la clase AsynchronousService donde se define el método sendMails que es asíncrono:

```
01 package com.autentia.tutoriales.spring.asynchronous;
02
03 import org.springframework.scheduling.annotation.*;
04 import org.springframework.stereotype.*;
05
06 @Service
07 public class AsynchronousService {;
08
09     private AsyncResult<String> result = new AsyncResult<String>("0 correos
   enviados");
10
11     @Async
12     public void sendMails(int totalMails) {
13         for (int i = 1; i <= totalMails; i++) {
14             try {
15                 sendMail(i);
16                 result = new AsyncResult<String>("Enviados " + i + " de " +
   totalMails);
17             } catch (InterruptedException e) {
18                 e.printStackTrace();
19             }
20         }
21     }
22
23     private void sendMail(int num) throws InterruptedException {
24         Thread.sleep(1000);
25         System.out.println("Mail " + num + " enviado.");
26     }
27
28     public AsyncResult<String> getResult() {
29         return result;
30     }
31
32     public String getMailsSender() {
33         return result.get();
34     }
35 }
```

En la línea 9 definimos un atributo result de tipo AsyncResult. Mediante la clase AsyncResult que trabaja con genéricos por lo que podemos añadirle cualquier objeto. Se utiliza para guardar el resultado del método asíncrono de manera que podemos consultarlo a medida que el método sigue en ejecución. En el ejemplo se simula el envío de correos cada segundo y se va almacenando un contador de los que ya han sido enviados. Imagina esta funcionalidad en una aplicación donde se desea mandar un correo de felicitación navideña a los usuarios y de vez en cuando queremos ver el número de correos enviados para ver los que quedan.

En la línea 11 podemos ver la anotación **@Async** que indica que este método será asíncrono es decir

web para dispositivos móviles

Simulación de navegación manteniendo la sesión con wget

Varias cuentas de gmail en Chrome

Apache Camel, primeros pasos

Últimos Tutoriales del Autor

Migración a ICEfaces 2.0

Cómo saber si tu navegador soporta HTML5 con Modernizr

Introducción a Selenium Grid y Test Paralelos con JUnit

Lanzar test de Selenium 2 en un navegador remoto

Introducción a Selenium 2 y WebDriver

Síguenos a través de:

Últimas ofertas de empleo

2011-09-08 Comercial - Ventas - MADRID.

2011-09-03 Comercial - Ventas - VALENCIA.

2011-08-19 Comercial - Compras - ALICANTE.

2011-07-12 Otras Sin catalogar - MADRID.

2011-07-06 Otras Sin catalogar - LUGO.

que se lanzará en un hilo independiente.

Ojo, en el ejemplo se utiliza un servicio que es un Singleton por lo que únicamente habrá una instancia de la clase, compartiendo todo el mundo el atributo result. A mí me vale para el ejemplo pero en un entorno concurrente no estaría bien diseñado.

Para probar el servicio creamos un test para que se encargue de llamar al método sendMails:

```
01 package com.autentia.tutoriales.spring.asynchronous;
02
03 import org.junit.*;
04 import org.junit.runner.*;
05 import org.springframework.beans.factory.annotation.*;
06 import org.springframework.test.context.*;
07 import org.springframework.test.context.junit4.*;
08
09 @RunWith(SpringJUnit4ClassRunner.class)
10 @ContextConfiguration({ "classpath:applicationContext-test.xml" })
11 public class AsynchronousServiceTest {
12
13     @Autowired
14     private AsynchronousService testAsynch;
15
16     @Test
17     public void sendAsynchronousMails() throws Exception {
18         testAsynch.sendMails(100);
19         System.out.println(testAsynch.getMailsSender());
20
21         sleepALittle(10000);
22
23         System.out.println(testAsynch.getMailsSender());
24
25         stopSendMails();
26     }
27
28     private void stopSendMails() {
29         testAsynch.getResult().cancel(true);
30     }
31
32     private void sleepALittle(int time) {
33         try {
34             Thread.sleep(time);
35         } catch (InterruptedException e) {
36             e.printStackTrace();
37         }
38     }
39 }
```

La clase de test no comprueba nada especialmente, sirve para lanzar la ejecución del servicio. Una vez que se llama al método de envío de 100 correos se duerme 10 segundos para dejar tiempo a que el servicio haga algo, podéis ver la traza de ejecución más abajo. Cuando se despierta el thread se llama al método stopSendMails que se encargará de parar el método asíncrono, de esta manera se comprueba que este tipo de métodos pueden ser detenidos antes de su finalización aunque hay que tener en cuenta que su parada no es inmediata, siempre tarda un poco hasta que se finaliza por completo.

La traza que se va dejando de la ejecución es la siguiente:

```
01 0 correos enviados
02 Mail 1 enviado.
03 Mail 2 enviado.
04 Mail 3 enviado.
05 Mail 4 enviado.
06 Mail 5 enviado.
07 Mail 6 enviado.
08 Mail 7 enviado.
09 Mail 8 enviado.
10 Mail 9 enviado.
11 Enviados 9 de 100
12 Mail 10 enviado.
```

En primera línea, como aún no le hemos dado tiempo a que el thread de envío de correos empiece, indica que no se ha enviado ningún correo. Seguidamente va saliendo una traza por cada uno de los correos enviados. En la línea 11 sale la traza de la consulta que hacemos antes de parar el servicio de envío de correo. En la última línea nos indica que se ha mandado un último correo ya que la parada del thread no es inmediata.

4. Tareas planificadas

Otra funcionalidad muy común en el desarrollo de aplicaciones es la ejecución de tareas planificadas o repetitivas. En nuestro ejemplo vamos a consultar cada 3 segundos el número de correos que ha enviado y así no lo tendremos que consultar desde la clase de test. Para ello añadimos el método logSendMailsState y lo anotamos @Scheduled configurando el tiempo entre llamadas al método:

```
1 @Scheduled(fixedDelay = 3000)
2 public void logSendMailsState() {
3     System.out.println(result.get());
4 }
```

Una vez hecho esto, podemos borrar parte del código del cliente que llama al servicio ya que obtendremos cada 3 segundos una traza del servicio de correo:

```
@Test
public void sendAsynchronousMails() throws Exception {
    testAsynch.sendMails(100);

    sleepALittle(10000);

    stopSendMails();
}
```

```
0 correos enviados
Mail 1 enviado.
Mail 2 enviado.
Mail 3 enviado.
Enviados 3 de 100
Mail 4 enviado.
Mail 5 enviado.
Mail 6 enviado.
Enviados 6 de 100
Mail 7 enviado.
Mail 8 enviado.
Enviados 8 de 100
Mail 9 enviado.
Mail 10 enviado.
```

Esta opción también podremos configurarla desde el fichero applicationContext-test.xml

```
<task:scheduled-tasks scheduler="myScheduler">
    <task:scheduled ref="asynchronousService" method="logSendMailsState"
fixed-rate="3000" />
</task:scheduled-tasks>

<task:scheduler id="myScheduler" />
```

También podremos añadirle un cron para que sean lanzadas de forma periódica al estilo: cron="*/5 * * * * MON-FRI".

5.Conclusiones

En este tutorial hemos visto la forma de lanzar tareas en segundo plano o asíncronas de forma sencilla con Spring. Tareas muy comunes como el envío masivo de correos bajo demanda, indexación de contenidos, generación de documentos pesados, consultas muy largas, etc. Si tienes alguna de estas tareas en tu aplicación y el usuario de la misma no debe recibir una respuesta inmediata, es preferible hacerlas de forma asíncrona para no perjudicar el rendimiento de la aplicación y que el usuario no esté un tiempo indefinido esperando la respuesta.

Espero que te haya servido de ayuda.

Un saludo.

Juan.

COMENTARIOS