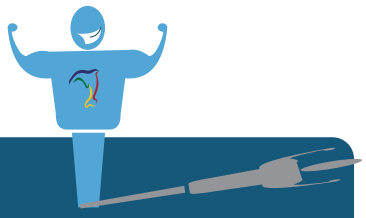
autentia

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...

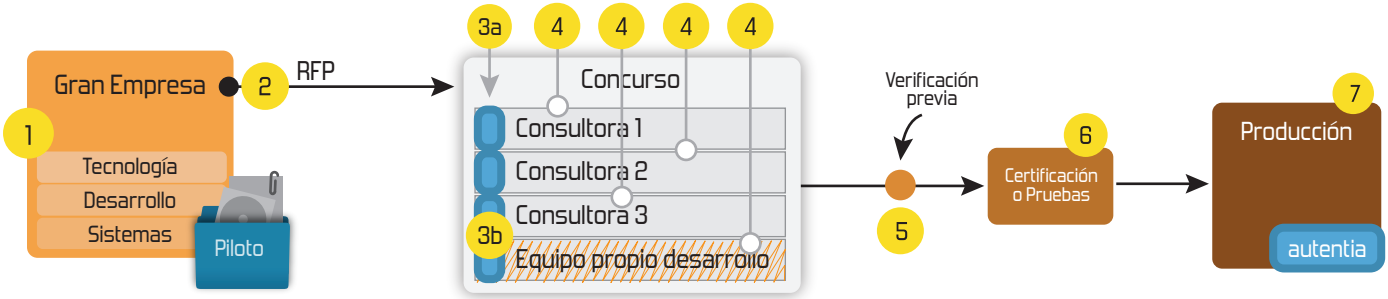
1. Desarrollo de componentes y proyectos a medida



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

Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

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

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

También puedes seguirme a través de: Twitter

Fecha de publicación del tutorial: 2012-02-20 Tutorial visitado 3 veces Descargar en PDF

Introducción a Spring Integration.

0. Índice de contenidos.

1. Introducción

Cada vez los sistemas empresariales son más potentes y complejos. Muchas compañías manejan gran cantidad de aplicaciones que se adaptan a sus modelos empresariales. Con el paso de los años, cada vez que una compañía necesitaba una solución informática para la gestión de alguno de sus procesos, se le hacía una aplicación y punto. Otra necesidad, otra aplicación. Y otra y otra...

Sin embargo, esto a veces conlleva un problema. Puede surgir la necesidad de que diferentes aplicaciones necesiten interactuar entre sí. Nace la necesidad de una plataforma empresarial donde los procesos se gestionen de una forma centralizada y no a través de 200 aplicaciones diferentes.

Spring Integration (módulo de Spring Framework) es una plataforma de integración que ayuda a crear sistemas desacoplados con componentes reutilizables. Nos ayuda a crear sistemas modulares con baja cohesión acercándonos a una arquitectura orientada a servicios.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

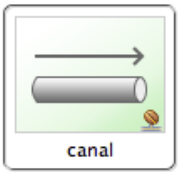
- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Snow Leopard 10.6.7
- Spring Integration 2.0.5.

- Spring 3.0.5.

3. Canales.

Spring Integration está basado en mensajes y canales. Esta es una definición tan mala como: que el fútbol son 11 tíos contra 11 corriendo detrás de un balón. Sin embargo es así. Spring Integration usa mensajes, canales y un amplio set de componentes (representan distintos patrones de integración) que interactuan con ellos. Los tipos de canales son:

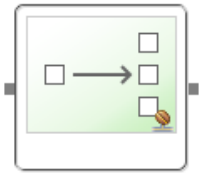
- Point-to-Point channel: canal por el que se garantiza que el emisor del mensaje lo envía a un único receptor.
- Publish-Subscribe channel: canal por el que un mensaje enviado por el emisor puede llegar a varios receptores.
- Data-Typed channel: canal por el que solo se pueden enviar mensajes de un tipo concreto.
- Messaging Bridge: conecta dos canales o adaptadores.
- Channel Adapter: canal usado para integrarse con diferentes sistemas de mensajería.
- Invalid Message channel: canal por el que se envían los mensajes que no pasan las validaciones.
- Dead Letter channel: cuando un mensaje no ha podido ser enviado por el canal correspondiente dicho mensaje es enviado por el Dead Letter channel.
- Guaranteed Delivery: evita que los mensajes, normalmente almacenados en memoria, se pierdan si el sistema se viene abajo. Son almacenados en una cola de mensajes JMS.



4. Flujo de mensajes.

Normalmente los mensajes no siguen un flujo lineal sino que será variable en función de la propia composición del mensaje. Spring Integration se basa en diferenes patrones de flujo de mensajes para decidir el camino que "dibujará" el mensaje a través de la plataforma.

- Agregador (aggregator): componente que, con diferentes mensajes, construye un único mensaje.
- Enrutador (router): determina si un mensaje va a un canal u otro (otros).
- Filtro (filter): decide si un mensaje puede llegar o no a su canal destino.
- Resequencer: decide el orden en el que un grupo de mensajes serán consumidos o procesados.
- Splitter: descompone un mensaje en múltiples mensajes. Igual que el agregador, pero al revés.



Splitter y Router

5. Transformadores.

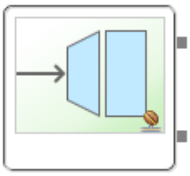
Los transformadores son parte importante en cualquier sistema de integración. Permiten adaptar un mensaje enviado por un emisor a la entrada esperada por un receptor. Aunque podemos definir nuestros propios transformadores concretos, Spring Integration nos proporciona los siguientes tipos:

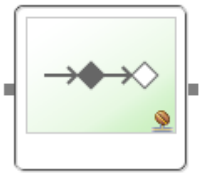
- object-to-string transformer: El mensaje saliente de un canal es transformado en un string (toString del mensaje) que será recibido por el siguiente canal.
- payload-serializing y payload-deserializing transformer: Estos dos transformadores son simétricos, el primero serializa un objeto (el mensaje) en un array de bytes y el segundo lo deserializa (array de bytes a objeto).
- object-to-map y map-to-object transformer: El primero transforma el mensaje (objeto) en un mapa de pares clave valor y el segundo realiza la operación inversa.
- json-to-object y object-to-json transformer: Pues un poco más de lo mismo, el primero transforma un objeto en una representación JSON y el segundo realiza la operación inversa.

6. Endpoints: adaptadores, gateways y service adapter.

Podríamos decir, sin miedo a equivocarnos, que el endpoint es la pieza clave en cualquier plataforma de integración. Los endpoints permiten conectar el sistema de integración con diferentes servicios o aplicaciones. Los endpoints soportan diferentes tipos de protocolos como pueden ser: servicios web, ficheros, mail, JMS o JDBC. En este apartado debemos destacar los siguientes componentes:

- Gateways: Exponen, en forma de fachada (interface), cómo la plataforma se comunicará con una aplicación o servicio. Abstrae al usuario de la lógica de negocio que lleva debajo.
- Service Activator: es un tipo de enpoint que maneja unos datos entrantes y devuelve una respuesta en caso de que sea necesaria.





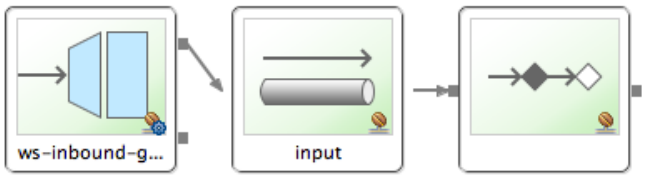
Gateway y Service Activator

7. ¿Vemos un ejemplo?

Pues bien, llegados a este punto vamos a ver un sencillo ejemplo de funcionamiento. Para ello los chicos de Spring nos han dejado en un repositorio un proyecto con múltiples ejemplos: https://github.com/SpringSource/spring-integration-samples

Lo descargamos, lo montamos en nuestro entorno y vamos a poner el foco de atención en un ejemplo que se llama "ws-inbound-gateway".

El ejemplo es sencillo: enviamos peticiones SOAP (XML) y la plataforma de integración nos responde lo que le hemos enviado envuelto en otro XML de respuesta. Gráficamente sería esto:



Y ahora alguien puede decir: "Pues vaya tontería, yo eso lo hago con un Web-Service...". Efectivamente, pero ese WS además de devolver la respuesta, ¿es capaz de interactuar con otros sistemas, monitorizar el tráfico u orquestar el flujo que llevará el mensaje de entrada en función de su contenido? Difícilmente...

Bueno, al lío. Vamos a echarle un ojo al fichero de configuración de Spring:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:int="http://www.springframework.org/schema/integration"
    xmlns:int-ws="http://www.springframework.org/schema/integration/ws"
    xsi:schemaLocation="http://www.springframework.org/schema/integration/ws http://www.
        http://www.springframework.org/schema/integration http://www.springframework.org
        http://www.springframework.org/schema/beans http://www.springframework.org/schem

    <int:channel id="input"/>

    <int-ws:inbound-gateway id="ws-inbound-gateway" request-channel="input"/>

    <int:service-activator input-channel="input">
        <bean class="org.springframework.integration.samples.ws.SimpleEchoResponder"/>
    </int:service-activator>

</beans>
```

Lo que se está haciendo es definir un canal "input" por el que pasarán los mensajes que hayan entrado por nuestro gateway de entrada "ws-inbound-gateway" y cuya salida irá a un Service Activator representado por la clase SimpleEchoResponser.

```
public class SimpleEchoResponder {

    public Source issueResponseFor(DOMSource request) {
        return new DomSourceFactory().createSource(
                "<echoresponse xmlns="\"http://www.springframework.org/spring-ws/samples/
                request.getNode().getTextContent() + "</echoresponse>");
    }
}
```

Esta clase recibe el mensaje que ha pasado por el canal en forma de DOMSource (representación del XML que nos ha enviado el cliente en un objeto), mete su contenido en un tag "echoResponse" y lo devuelve.

Para probar el ejemplo, podemos hacerlo con herramientas como JMeter o SOAP UI o mediante la clase de test que nos viene con el ejemplo.

```
@ContextConfiguration("/META-INF/spring/integration/inbound-gateway-config.xml")
@RunWith(SpringJUnit4ClassRunner.class)
```

```
3  public class InboundGatewayTests {
4
5      @Autowired
6      private SimpleWebServiceInboundGateway gateway;
7
8      /**
9       * Emulate the Spring WS MessageDispatcherServlet by calling the gateway
10      * with a DOMSource object representing the payload of the original SOAP
11      * 'echoRequest' message.  Expect an 'echoResponse' DOMSource object
12      * to be returned in synchronous fashion, which the MessageDispatcherServlet
13      * would in turn wrap in a SOAP envelope and return to the client.
14      */
15     @Test
16     public void testSendAndReceive() throws Exception {
17         String xml = "<echorequest xmlns="\"http://www.springframework.org/spring-ws/sampl
18         DomPoxMessageFactory messageFactory = new DomPoxMessageFactory();
19         DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance
20         documentBuilderFactory.setNamespaceAware(true);
21         DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
22         Document document = documentBuilder.parse(new InputSource(new StringReader(xml)));
23         Transformer transformer = TransformerFactory.newInstance().newTransformer();
24         DomPoxMessage request = new DomPoxMessage(document, transformer, "text/xml");
25         MessageContext messageContext = new DefaultMessageContext(request, messageFactory)
26         gateway.invoke(messageContext);
27         Object reply = messageContext.getResponse().getPayloadSource();
28         assertThat(reply, is(DOMSource.class));
29         DOMSource replySource = (DOMSource) reply;
30         Element element = (Element) replySource.getNode().getFirstChild();
31         assertThat(element.getTagName(), equalTo("echoResponse"));
32     }
33 }
```

8. Referencias.

- Spring Integration: Reference Manual.

9. Conclusiones.

En este tutorial hemos pretendido presentar la implementación de plataforma de integración que nos propone Spring. Además hemos visto cómo hace uso de los principales patrones de integración.

Recordad que esta no es la única solución para integrar diferentes servicios o sistemas, los potentes ESB´s (ServiceMix, MuleESB, Synapse, etc...) suelen ser una solución muy recurrida en cuanto a diseño de arquitecturas horizontales nos referimos. ¿La ventaja de Spring Integration? Pues probablemente que a la gente que está acostumbrada a usar Spring le costará menos hacerse con las riendas de este producto antes que con un ESB.

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Registrate** y accede a esta y otras ventajas «