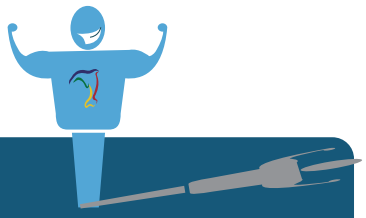
autentia

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...

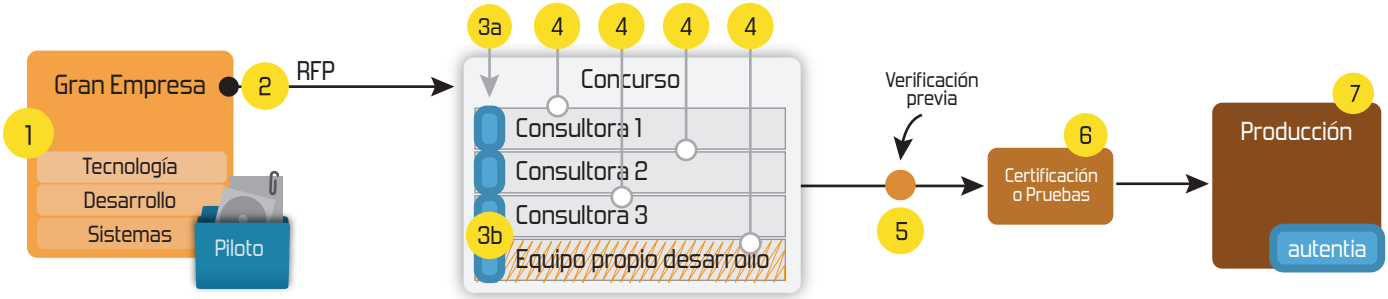
1. Desarrollo de componentes y proyectos a medida



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

Fecha de creación del tutorial: 2009-03-31

Spring WS: Construcción de Clientes de Servicios Web con Spring

En el siguiente tutorial vamos a ver algunas de las aportaciones que nos ofrece Spring en relación a la construcción de clientes de servicios web.

Se presupone que el lector ya posee conocimientos de Servicios Web, Maven y Spring.

Si estás interesado en la construcción de servicios web con Spring, puedes ver el siguiente tutorial

Índice de contenido:

Introducción

Al igual que en otros temas en Spring como JMS o DAO, Spring apuesta por el uso de plantillas que nos ahorren tener que realizar las tareas comunes evitándonos además cometer ciertos errores ligados a estas

partes.
Por ejemplo, en el caso de DAO, las plantillas nos evitan tener que abrir conexiones, liberar recursos, etc.

En el caso de los servicios web, Spring nos proporciona la plantilla `org.springframework.ws.client.core.WebServiceTemplate` a tráves de la cual podremos enviar mensajes (payload) a un servicio Web, y será la plantilla la que se encarge de envolverla en sobres SOAP, de establecer conexiones, liberar recursos, controlar errores de obligada captura, etc.
Además esta plantilla nos proporciona imnumerables métodos para enviar mensajes con cualquier tipo de mensajería y médio transporte.

Cabe destacar que para construir un cliente de un servicio web usando Spring necesitas entender el WSDL que define el servicio web, es decir:

- Protocolo de comunicación.
- Dónde escucha la peticiones.
- Formatos peticiones, respuestas y fallos.

Entender un WSDL no es tan dificil como a veces se puede pensar, en un par de tardes con ganas los lees sin problemas :-)

Nuestro cliente usará por dentro una instancia de `WebServiceTemplate` (normalmente inyectada por Spring) y está plantilla se deberá de configurar dos parámetros (en la configuración de Spring):

1. **La fábrica de mensajes SOAP:** Envuelve el Payload que envias a través de la plantilla en sobres SOAP Envelope.
2. **La clase de envio de menssajes SOAP:** Se encarga del envio en si del mensaje construido anteriormente.

Tendrás información más detallada en los comentarios del archivo de configuración de Spring que se verá más adelante en este mismo tutorial.

Ejemplo de construcción de un cliente de servicio web con Spring:

A continuación vamos a ver un completo ejemplo en donde construiremos un cliente para el Servicio Web de búsqueda de libros que definimos en un tutorial anterior. Ver tutorial.

El código fuente de este tutorial puede ser descargado desde aquí (proyecto Eclipse con Maven 2).

Entorno

El siguiente ejemplo está construido en el siguiente entorno:

- HP Pavilion.
- Windows Vista Home Premium.
- Eclipse Ganymede.
- Java 6.
- Maven 2.
- Plugin Maven 4QE para Eclipse.

Describiendo la petición y la respuesta de comunicación con el servicio web:

Spring **SÓLO APUESTA por el modelo contrato-primero (Contract-First)**, por lo que debemos de enviar y recibir la información con el formato que se especifique en el contrato del servicio Web `(WSDL)`.

A continuación vemos el formato de datos de la comunicación con el servicio web y al cual debemos ceñirnos estrictamente.

- Formato de la petición a enviar al servicio web.
- Formato de la respuesta que nos genera el servicio web.

Código fuente del cliente:

`com.autentia.tutoriales.spring.ws.biblioteca.cliente.BibliotecaCliente`:

view plain print ?

```
01. package com.autentia.tutoriales.spring.ws.biblioteca.cliente;
02.
03. import java.util.ArrayList;
04. import java.util.List;
05. import java.io.StringReader;
06. import javax.xml.transform.dom.DOMResult;
07. import javax.xml.transform.stream.StreamSource;
08. import org.springframework.ws.client.core.WebServiceTemplate;
09. import org.w3c.dom.*;
10.
11. /**
12.  * Spring cliente WS del servicio de consulta de libros
13.  * @author Carlos García. Autentia.
14.  * @see http://www.mobiletest.es
15.  */
16. public class BibliotecaCliente  {
17.     private final String       URI = "http://localhost:8080/bibliotecaWS
    /services";
18.     private WebServiceTemplate webServiceTemplate;
19.
20.     /**
21.      * Será inyectada por Spring
22.      */
23.     public void setWebServiceTemplate(WebServiceTemplate webServiceTemplate) {
24.         this.webServiceTemplate = webServiceTemplate;
25.     }
26.
27.     /**
28.      * @param categoria Categoría del libro a consultar
29.      * @param nivel Nivel (avanzado, medio o básico)
30.      * @return Devuelve la lista de libros que cumplen los criterios especifi
31.      */
32.     public java.util.List<Libro> getLibros(String categoria, String nivel) {
33.         String       xmlRequest = this.getPeticion(categoria, nivel);
34.         StreamSource peticion   = new StreamSource(new StringReader(xmlRequest));
35.         DOMResult    respuesta  = new DOMResult();
36.         List<Libro>  libros = null;
37.
38.         boolean hayRespuesta = webServiceTemplate.sendSourceAndReceiveToResult(URI, p
39.         if (hayRespuesta){
40.             libros = this.resultToBooks((Document) respuesta.getNode());
41.         }
42.         return libros;
43.     }
44.
45.     /**
46.      * @return Genera la petición (payload) que se enviará al servicio Web
47.      */
48.     private String getPeticion(String categoria, String nivel){
49.         StringBuffer buffer = new StringBuffer(1024);
50.         buffer.append("<BooksInfoRequest xmlns='http://www.adictosaltrabajo.com
    /spring/ws/schemas'>");
51.         buffer.append("<categoria>");
52.         buffer.append(categoria);
53.         buffer.append("</categoria>");
54.         buffer.append("<nivel>");
55.         buffer.append(nivel);
56.         buffer.append("</nivel>");
57.         buffer.append("</BooksInfoRequest>");
58.         return buffer.toString();
59.     }
60.
61.     /**
62.      * @return Devuelve una lista de Libro a partir del DOM
63.      */
64.     private List<Libro> resultToBooks(Document doc){
65.         NodeList nodos   = doc.getFirstChild().getChildNodes();
66.         Node     current = null;
67.         Libro    libro   = null;
68.
69.         ArrayList<Libro> libros = new ArrayList<Libro>();
70.         for (int i = 0, num = nodos.getLength(); i < num; i++){
71.             current = nodos.item(i);
72.
73.             libro   = new Libro();
74.             libro.setEditorial(this.getProperty(current, "editorial"));
75.             libro.setTitulo(this.getProperty(current, "titulo"));
76.             libro.setPrecio(Integer.parseInt(this.getProperty(current, "precio")));
```

com.autentia.tutoriales.spring.ws.biblioteca.cliente.Libro:

view plain print ?

```
01. package com.autentia.tutoriales.spring.ws.biblioteca.cliente;
02.
03. /**
04.  * Representación de un libro
05.  * @author Carlos García. Autentia
06.  * @see http://www.mobiletest.es
07.  */
08. public class Libro {
09.     private String editorial;
10.     private String titulo;
11.     private int    paginas;
12.     private int    precio;
13.
14.     public String getEditorial() {
15.         return editorial;
16.     }
17.     public void setEditorial(String editorial) {
18.         this.editorial = editorial;
19.     }
20.     public String getTitulo() {
21.         return titulo;
22.     }
23.     public void setTitulo(String titulo) {
24.         this.titulo = titulo;
25.     }
26.     public int getPaginas() {
27.         return paginas;
28.     }
29.     public void setPaginas(int paginas) {
30.         this.paginas = paginas;
31.     }
32.     public int getPrecio() {
33.         return precio;
34.     }
35.     public void setPrecio(int precio) {
36.         this.precio = precio;
37.     }
38. }
```

Archivo de configuración de Spring 2 (/main/resources /applicationContext.xml):

El archivo está autocomentado.

```
view plain  print  ?
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema
/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                           http://www.springframework.org
/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd">

    <!-- Wrapper del cliente  -->
    <bean id="biblioStub" class="com.autentia.tutoriales.spring.ws.biblioteca.cliente
        <property name="webServiceTemplate" ref="wsTemplate"/>
    </bean>

    <!-- Plantilla para comunicarnos con el WS -->
    <bean id="wsTemplate" class="org.springframework.ws.client.core.WebServiceTemplat
        <property name="defaultUri" value="http://localhost:8080/bibliotecaWS
/services"/>

        <!-- Creación de mensajes SOAP -->
        <property name="messageFactory">
              <!-- Saaj usa DOM, si se quiere más rendimiento y consumir menos recurs
                   puede usar: org.springframework.ws.soap.axiom.AxiomSoapMessageFact
                   que usa AXIOM. -->

            <bean name="innermf" class="org.springframework.ws.soap.saaj.SaajSoapMess
                <property name="soapVersion">
                    <util:constant static-
field="org.springframework.ws.soap.SoapVersion.SOAP_12"/>
                </property>
            </bean>
        </property>

        <!-- Envio de mensajes, Spring proporciona dos clases el envio de mensajes s
            a) org.springframework.ws.transport.http.HttpUrlConnectionMessageSender:
               Usa HTTPConnection (funcionalidad limitada)
            b) org.springframework.ws.transport.http.CommonsHttpMessageSender:
               Usa HTTPClient (proporciona más funcionalidad)
               http://www.adictosaltrabajo.com/tutoriales
/tutoriales.php?pagina=HTTPClient
         -->
        <property name="messageSender">
              <bean class="org.springframework.ws.transport.http.HttpUrlConnectionMes
        </property>
    </bean>
</beans>
```

Archivo de configuración de Maven 2: `pom.xml`:

A continuación exponemos el archivo de configuración de Maven, se presupone que el lector ya tiene nociones de Maven.

```
view plain  print  ?
01. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance"
02.     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org
    /maven-v4_0_0.xsd">
03.     <modelVersion>4.0.0</modelVersion>
04.     <groupId>com.autentia.tutoriales</groupId>
05.     <artifactId>bibliotecaWSClienteSpring</artifactId>
06.     <packaging>jar</packaging>
07.     <version>1.0-SNAPSHOT</version>
08.     <name>bibliotecaWSClienteSpring</name>
09.     <url>http://www.adictosaltrabajo.com</url>
10.     <build>
11.         <plugins>
12.             <!-- Sintáxis Java 5 -->
13.             <plugin>
14.                 <artifactId>maven-compiler-plugin</artifactId>
15.                 <configuration>
16.                     <source>1.5</source>
17.                     <target>1.5</target>
18.                     <encoding>UTF-8</encoding>
19.                 </configuration>
20.             </plugin>
21.         </plugins>
22.     </build>
23.     <dependencies>
24.         <!-- Clases de Spring WS -->
25.         <dependency>
26.             <groupId>org.springframework.ws</groupId>
27.             <artifactId>spring-ws-core</artifactId>
28.             <version>1.5.6</version>
29.         </dependency>
30.
31.         <!-- Para volcar las trazas. (sólamente dejamos trazas SOAP) -->
32.         <dependency>
33.             <groupId>log4j</groupId>
34.             <artifactId>log4j</artifactId>
35.             <version>1.2.14</version>
36.         </dependency>
37.
38.         <!-- Para nuestros tests -->
39.         <dependency>
40.             <groupId>junit</groupId>
41.             <artifactId>junit</artifactId>
42.             <version>4.3.1</version>
43.             <scope>test</scope>
44.         </dependency>
45.     </dependencies>
46. </project>
```

Configuración de log para monitorizar el tráfico SOAP: `/main/resources/log4j.xml`:

view plain print ?

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "dtds/log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org
/log4j/" debug="false">

    <!-- Las trazas SOAP irán a parar al siguiente archivo:  -->
    <appender name="soapFile" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="bibliotecaWSClienteSOAP.log" />
        <param name="MaxFileSize" value="2000000" />
        <param name="MaxBackupIndex" value="5" />

        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                value="%n%d{yyyy-MM-dd HH:mm:ss} [%-5p] [%l] %n%m%n" />
        </layout>
    </appender>


    <!-- Monitorizamos el tráfico SOAP -->
    <category name="org.springframework.ws.client.MessageTracing">
        <priority value="debug" />
        <appender-ref ref="soapFile" />
    </category>
</log4j:configuration>
```

Invocación del servicio Web:

A continuación nos creamos un test funcional con JUnit para probar la invocación y respuesta del servicio Web.

`com.autentia.tutoriales.spring.biblioteca.cliente.test.BibliotecaTest:`

```
view plain  print  ?
01. package com.autentia.tutoriales.spring.biblioteca.cliente.test;
02.
03. import org.junit.Assert;
04. import org.springframework.context.ApplicationContext;
05. import org.springframework.context.support.FileSystemXmlApplicationContext;
06.
07. import com.autentia.tutoriales.spring.ws.biblioteca.cliente.BibliotecaCliente;
08. import com.autentia.tutoriales.spring.ws.biblioteca.cliente.Libro;
09.
10. /**
11.  * Tests de verificación del servicio web de consulta de libros
12.  * @author Carlos García. Autentia
13.  * @see http://www.mobiletest.es
14.  */
15. public class BibliotecaTest  {
16.     private ApplicationContext  factory;
17.
18.     /**
19.       * Inicializamos el contexto de Spring
20.       */
21.     @org.junit.Before
22.     public void initTests(){
23.         this.factory  = new FileSystemXmlApplicationContext("classpath:applicationCo
24.     }
25.
26.     /**
27.       * En verdad esto no es un test, simplemente lo pongo aqui por comodidad
28.       * tener que crearme un proyecto independenciente con una aplicación que
29.       * que puedo hacer con este "test".
30.       */
31.     @org.junit.Test
32.     public void test1(){
33.         BibliotecaCliente       stub    = (BibliotecaCliente) factory.getBean("biblio
34.         java.util.List<Libro> libros  =  stub.getLibros(".net", "avanzado");
35.         if (libros != null){
36.              for (int i = 0, lcount = libros.size(); i < lcount; i++){
37.                  Libro libro = libros.get(i);
38.                  System.out.println(libro.getEditorial() + " " + libro.getTitulo() +
39.              }
40.         } else {
41.             System.out.println("No hay libros");
42.         }
43.
44.         Assert.assertTrue(true);
45.     }
46. }
```

Y para terminar, ejecutamos los tests y vemos la salida generada `mvn test`:

```
........
........

Editorial libro 0 Titulo libro 0 100 50
Editorial libro 1 Titulo libro 1 101 51
Editorial libro 2 Titulo libro 2 102 52
Editorial libro 3 Titulo libro 3 103 53
Editorial libro 4 Titulo libro 4 104 54
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.681 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

........
........
```

Vemos los logs que nos ha dejado la aplicación `(bibliotecaWSClienteSOAP.log)`:

```
2009-03-29 23:05:39 [TRACE] [org.springframework.ws.client.core.WebServiceTemplate.sendRequest(W
Sent request [<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"><env:Header/><en

2009-03-29 23:05:39 [TRACE] [org.springframework.ws.client.core.WebServiceTemplate.logResponse(W
Received response [<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"><env:Header
```

Referencias

- http://static.springframework.org/spring-ws/sites/1.5/reference/html/index.html
- http://static.springsource.org/spring-ws/sites/1.5/reference/html/client.html

Conclusiones

Si lo piensas bien, este forma de comunicación con servicios Web puede parecer que requiere un gran esfuerzo en comparación con otras técnicas, pero no es para tanto, usando técnicas OXM (Object XML Mapping) y el uso de plantillas (`WebServiceTemplate`) el esfuerzo se reduce enormemente y lo que es mejor, para las personas que nos gusta tener el control sobre lo que está pasando, esta solución es ideal, pues todo son mensajes en XML fácilmente depurables.

Otra cosa, recuerda que esto no es más que un tutorial, si quieres profundizar más, lee un buen libro, mira otras webs o solicita a tu empresa formación al respecto y en Autentia estaremos encantados de ofrecerte un curso a la medida de vuestras necesidades.

Un saludo, espero que os haya parecido útil este tutorial.
Carlos García. Creador de MobileTest, un complemento educativo para los profesores y sus alumnos.

Formulario de subcripción a novedades:

E-mail Aceptar

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	Votos	Pdf
Web Services con Estado	Este tutorial muestra un ejemplo de un web service con estado (stateful webservice)	2008-10-27	2224	Muy bueno	2	
Web Service Security	En este tutorial vamos a ver como crear un servicio web seguro con autenticación mediante usuario y contraseña. WS-Security define cómo utilizar los tokens de seguridad, XML Signature y Xml Encryption en los mensajes SOAP para proporcionar autenticación,	2009-02-09	955	Muy bueno	3	
Spring WS: Creación de Servicios Web con Spring	En este tutorial veremos un completo ejemplo de creación de un servicio web Contract-First con Spring y Maven	2009-03-15	566	Muy bueno	1	
Web Services en tu IPAQ	Cesar Crespo nos enseña como programar accesos Web Services desde tu IPAQ en Visual C++ con PocketSOAP, Apache SOAP y Axis	2004-08-02	37070	Muy bueno	1	
Servicios Web RESTful en Axis 2	En este tutorial vamos realizar una descripción de REST y vamos a ver un ejemplo práctico de un Servicio Web RESTful en AXIS 2	2008-04-03	4911	Muy bueno	8	
Web Services con Axis2. Configuración y ejemplo	Este es un tutorial básico que introduce a los servicios web en Java, y muestra cómo configurar el equipo con Eclipse, Apache Tomcat y Axis2 para poder crear luego un web service de ejemplo	2008-10-24	4607	Muy bueno	17	
Monitorización de Web Services con Glassfish Wsmonitor	En este tutorial vamos a realizar una introducción a una herramienta de monitorización de mensajes SOAP o Servicios Web en general.	2008-04-04	1523	Bueno	1	
Metro: pila de webservices de Sun. Integración con Maven 2	En este tutorial Germán nos enseñara a integrar la generación de webservices con Metro y Maven2.	2008-04-05	2027	Bueno	1	
Metro: pila de webservices de Sun.	NE este tutorial Germán nos enseñara qué es y cómo usar Metro: pila de webservices de Sun en nuestras aplicaciones	2008-04-05	3348	Bueno	5	
Axis2. Ejemplo de creación de un servicio Web	En este tutorial veremos como crear un servicio web a partir de un interface Java así como otros aspectos de está tecnología.	2008-04-04	4424	Bueno	5	

Nota: