Spring WS: Creación de Servicios Web con Spring

Por **Carlos García Pérez** - 15 marzo, 2009

Spring WS: Creación 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 Servicios Web.

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

Indice de contenido:

Introducción

Todos estaremos de acuerdo en que las aplicaciones actuales no están aisladas, que hay necesidad (y deben) interoperar para resolver los problemas, es más, incluso aunque ahora no sea necesario, se debe diseñar una solución para que en caso de necesidad el impacto sea mínimo.

Un problema debería ser resuelto una vez y ser reutilizado por el resto de sistemas (con independencia de lenguajes de programación y arquitecturas) sin tener que volver que resolver el mismo problema implementando una y otra vez lo mismo en cada nueva aplicación.... (SOA).

En este tutorial vamos a ver un ejemplo de construcción de un servicio web usando el modelo contrato primero (Spring apuesta por este modelo).
Desde mi punto de vista la mejor forma de implementar un servicio Web, cuando de verdad se desea interoperabilidad… no quiero entrar en este tema si os interesa, podéis leer la siguientes webs que explican por que es mejor el modelo contrato primero frente a partir de una interfaz de programación:

- http://static.springframework.org/spring-ws/sites/1.5/reference/html/why-contract-first.html.
- http://tssblog.blogs.techtarget.com/contract-first-or-code-first-design-part-1.

Para terminar la introducción decir que Spring también tiene mucho que aportar en relación a otras técnicas de comunicación remota: RMI, Hessian y Burlap, HTTPInvoker. Pero se salen del ámbito de este tutorial.

Ejemplo de construcción de un Servicio Web con Spring:

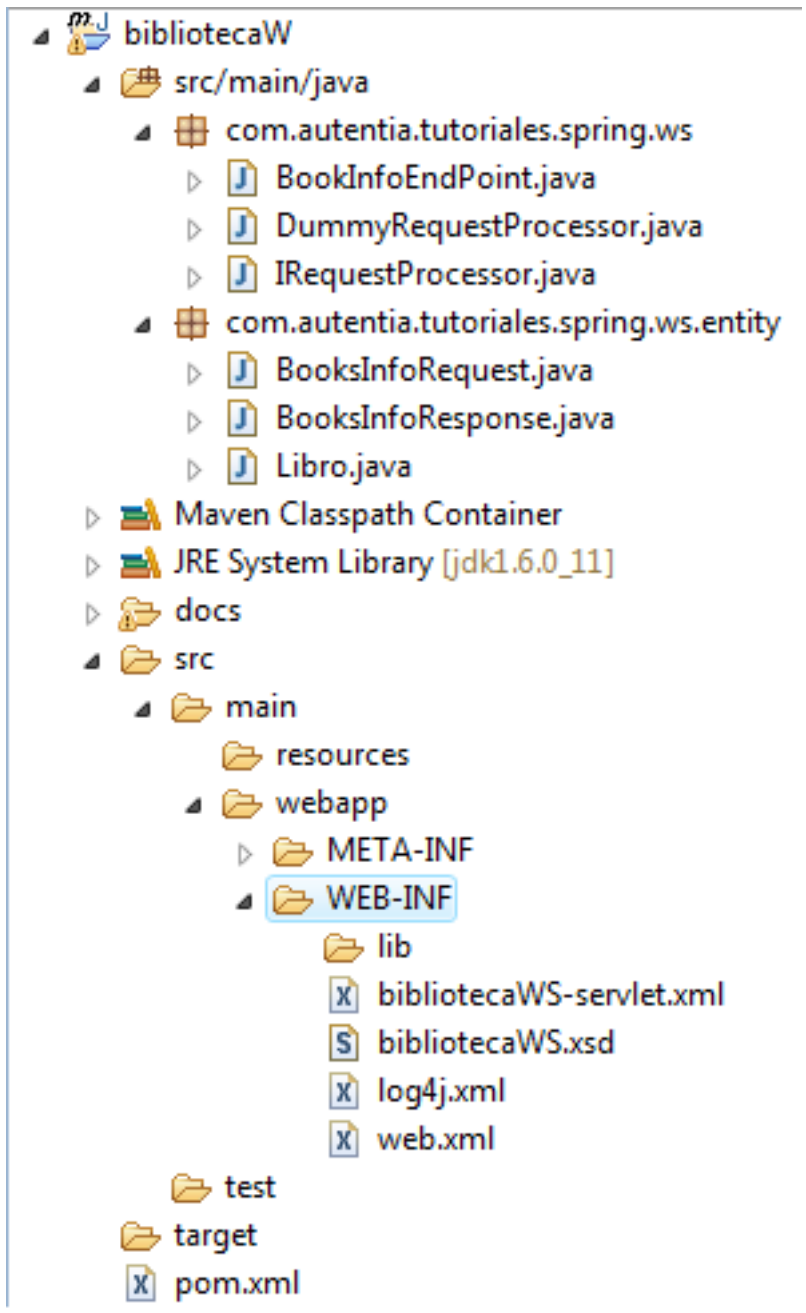
A continuación vamos a ver un completo ejemplo en donde construiremos un Servicio Web que representa al motor de búsqueda de libros de una biblioteca. Las aplicaciones preguntarán sobre libros de una determinada categoría y nivel y el servicio devolverá aquellos libros disponibles que cumplen esos criterios de búsqueda.

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.

Estructura del proyecto:



Una de las muchas ventajas de Maven es que estandariza la estructura de los proyectos, es decir, cualquier persona con conocimientos de Maven tendría facilidad de comprender como se estructura y dónde está cada cosa dentro del proyecto.

Si nos fijamos hay un arquetipo Maven 2 para proyectos Spring WS: http://www.mavenrepository.com/artifact/org.springframework.ws/spring-ws-archetype

Yo cree el proyecto con el comando:

```
mvn archetype:create -
DarchetypeGroupId=org.springframework.ws -
DarchetypeArtifactId=spring-ws-archetype -
DarchetypeVersion=1.5.6 -
DgroupId=com.autentia.tutoriales -
DartifactId=bibliotecaWS
```

Vosotros como ya lo tenéis hecho, simplemente tendréis que importarlo desde vuestro IDE favorito. Yo uso Eclipse Ganymede con el plugin Q4E para gestión de proyectos Maven.

Describiendo la petición al servicio web: `booksInfoRequest.xml`

¿Cómo nos gustaría que fueran los mensajes de solicitud o consulta de libros?... pues, por ejemplo así:

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<BooksInfoRequest  xmlns="http://www.adictosaltrabajo.com/spring/ws/sch
    <categoria>Servicios Web</categoria>
    <nivel>avanzado</nivel>
</BooksInfoRequest>
```

Una categoría y un nivel, en donde el nivel está restringido a ser:

básico o medio o avanzado

Ahora lo definimos con un XML Schema: `booksInfoRequest.xsd`

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.adictosaltrabajo.com/spring/ws/schemas
    xmlns:schemas="http://www.adictosaltrabajo.com/spring/ws/schemas">

    <xsd:element name="BooksInfoRequest">
        <xsd:complexType>
            <xsd:sequence>
```

```
                <xsd:element name="categoria" type="xsd:string" />
                <xsd:element name="nivel"     type="schemas:nivelType"
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:simpleType name="nivelType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="basico" />
            <xsd:enumeration value="medio" />
            <xsd:enumeration value="avanzado" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

¿Qué nó sabes hacerlo o no te apetece?, pues mira esté tutorial: Ver tutorial.

Describiendo la respuesta del servicio web: `booksInfoResponse.xml`

¿Cómo nos gustaría que fueran los mensajes de respuesta?... pues, por ejemplo así (con N libros):

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<BooksInfoResponse xmlns="http://www.adictosaltrabajo.com/spring/ws/s
    <libro>
        <editorial>Editorial Guay</editorial>
        <titulo>Java en ejemplos</titulo>
        <paginas>520</paginas>
        <precio>40</precio>
    </libro>
    <libro>
        <editorial>Editorial XYZ</editorial>
        <titulo>Aprenda Java en 50 dias</titulo>
        <paginas>700</paginas>
        <precio>80</precio>
    </libro>
</BooksInfoResponse>
```

N libros en donde cada uno está definido por una editorial, un título (ambos texto libre), un número de páginas y un precio (ambos números positivos).

Ahora lo definimos con un XML Schema: `booksInfoResponse.xsd`

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.adictosaltrabajo.com/spring/ws/schemas
    xmlns:schemas="http://www.adictosaltrabajo.com/spring/ws/schemas">

  <xsd:element name="BooksInfoResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="schemas
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="libro">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="editorial" type="xsd:string"/>
        <xsd:element name="titulo"    type="xsd:string"/>
        <xsd:element name="paginas"   type="xsd:positiveInteger"/>
        <xsd:element name="precio"    type="xsd:positiveInteger"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Definir el Web Service: `/WEB-INF/bibliotecaWS.xsd`

Sencillo, unimos los XML Schema anteriores y el resto lo configuramos en Spring. Es decir, se podría decir que en conjunto es como el WSDL del servicio Web, es más, de hecho se generará automáticamente.

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.adictosaltrabajo.com/spring/ws/schemas
    xmlns:schemas="http://www.adictosaltrabajo.com/spring/ws/schemas">

    <xsd:element name="BooksInfoRequest">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="categoria" type="xsd:string" />
                <xsd:element name="nivel"     type="schemas:nivelType"
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:simpleType name="nivelType">
```

```
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="basico" />
            <xsd:enumeration value="medio" />
            <xsd:enumeration value="avanzado" />
        </xsd:restriction>
    </xsd:simpleType>

  <xsd:element name="BooksInfoResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="schemas
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="libro">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="editorial" type="xsd:string"/>
        <xsd:element name="titulo"   type="xsd:string"/>
        <xsd:element name="paginas"  type="xsd:positiveInteger"/>
        <xsd:element name="precio"   type="xsd:positiveInteger"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Código fuente de las clases que componen el servicio Web.

A continuación iremos viendo las clases que componen el proyecto.

NOTA: Lo voy a hacer a mano, pero las clases Libro, BookInfoRequest y BookInfoResponse podrían de muchas formas ser generadas automáticamente desde los XSD.
Por ejemplo, en el siguiente enlace https://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=xmlBeans mi compañero Alejandro García os explica una de ellas.

`com.autentia.tutoriales.spring.ws.enti`

Shell

```
package com.autentia.tutoriales.spring.ws.entity;

/**
 * Representación de un libro
 * @author Carlos García. Autentia
```

```
 * @see http://www.mobiletest.es
 */
public class Libro {
    private String editorial;
    private String titulo;
    private int    paginas;
    private int    precio;

    public String getEditorial() {
        return editorial;
    }
    public void setEditorial(String editorial) {
        this.editorial = editorial;
    }
    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    public int getPaginas() {
        return paginas;
    }
    public void setPaginas(int paginas) {
        this.paginas = paginas;
    }
    public int getPrecio() {
        return precio;
    }
    public void setPrecio(int precio) {
        this.precio = precio;
    }
}
```

com.autentia.tutoriales.spring.ws.enti

Shell

```
package com.autentia.tutoriales.spring.ws.entity;

/**
 * Representación de una petición.
 * @author Carlos García. Autentia
 * @see http://www.mobiletest.es
 */
public class BooksInfoRequest {
    private String categoria;
    private String nivel;

    public String getCategoria() {
        return categoria;
    }
    public void setCategoria(String categoria) {
        this.categoria = categoria;
    }
    public String getNivel() {
```

```
        return nivel;
    }
    public void setNivel(String nivel) {
        this.nivel = nivel;
    }
}
```

com.autentia.tutoriales.spring.ws.enti

Shell

```
package com.autentia.tutoriales.spring.ws.entity;

/**
 * Representación la respuesta a una petición BooksInforRequest.
 * @author Carlos García. Autentia
 * @see http://www.mobiletest.es
 */
public class BooksInfoResponse {
    private java.util.ArrayList<Libro> libros;

    public BooksInfoResponse(){
        this.libros = new java.util.ArrayList<Libro>();
    }

    public void addLibro(Libro libro){
        this.libros.add(libro);
    }

    public java.util.List<Libro> getLibros() {
        return this.libros;
    }
}
```

com.autentia.tutoriales.spring.ws.IReq

Cuando el servicio web reciba una petición, la lógica de negorio real de tratamiento
de la misma será realizada por alguna clase que implemente esta interfaz.

Shell

```
package com.autentia.tutoriales.spring.ws;

import com.autentia.tutoriales.spring.ws.entity.BooksInfoRequest;
import com.autentia.tutoriales.spring.ws.entity.BooksInfoResponse;

/**
 * Tratamiento de una petición de búsqueda de libros
 * @author Carlos García. Autentia.
 * @see http://www.mobiletest.es
```

```
 */
public interface IRequestProcessor {
    /**
     * Procesa la petición de consulta
     * @param request Datos de la consulta
     * @return Devuelve la respuesta
     */
    public BooksInfoResponse process(BooksInfoRequest request);
}
```

com.autentia.tutoriales.spring.ws.Dumm

Implementación sencilla de la interfaz IRequestProcessor para este este ejemplo.

Shell

```
package com.autentia.tutoriales.spring.ws;

import com.autentia.tutoriales.spring.ws.entity.BooksInfoRequest;
import com.autentia.tutoriales.spring.ws.entity.BooksInfoResponse;
import com.autentia.tutoriales.spring.ws.entity.Libro;

/**
 * Implementación dummy de IRequestProcesor
 * @author Carlos García. Autentia.
 * @see http://www.mobiletest.es
 */
public class DummyRequestProcessor implements IRequestProcessor {

    /*
     * @see com.autentia.tutoriales.spring.ws.IRequestProcessor#proce
     */
    public BooksInfoResponse process(BooksInfoRequest request) {
        BooksInfoResponse response = new BooksInfoResponse();
        Libro             libro    = null;

        for (int i = 0; i < 5; i++){
            libro = new Libro();

            libro.setTitulo("Titulo libro " + i);
            libro.setEditorial("Editorial libro " + i);
            libro.setPaginas(100 + i);
            libro.setPrecio(50 + i);

            response.addLibro(libro);
        }

        return response;
    }
}
```

com.autentia.tutoriales.spring.ws.Book

EndPoint del WS, recibe las peticiones de consulta de libros (peticiones XML), las convierte en objetos
y delega su procesamiento a un IRequestProcessor.

Yo he elegido hacerlo con DOM (mensajes completos en memoria...), pero hay otras muchas implementaciones que nos permiten implementarlo con Stax, SAX, JDOM...
Simplemente deberemos leer, procesar y constuir en base a DOM los XML.

Shell

```
package com.autentia.tutoriales.spring.ws;

import java.util.Iterator;
import java.util.List;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.ws.server.endpoint.AbstractDomPayloadEndp
import com.autentia.tutoriales.spring.ws.entity.*;

/**
 * EndPoint del WS, recibe las peticiones de consulta de libros (pet
 * las convierte en objetos y delega su procesamiento a un IRequestPr
 * @author Carlos García. Autentia
 * @see http://www.mobiletest.es
 */
public class BookInfoEndPoint extends AbstractDomPayloadEndpoint {
    private Log       logger = LogFactory.getLog(BookInfoEndPoint.cl

    private IRequestProcessor procesor;

    /**
     * Será inyectado por Spring
     */
    public void setProcesor(IRequestProcessor procesor) {
        this.procesor = procesor;

    }

    /*
     * @see org.springframework.ws.server.endpoint.AbstractDomPayloa
     */
    protected Element invokeInternal(Element domRequest, Document do
        if (logger.isDebugEnabled()){
            logger.debug("Petición de consulta de libros");
        }

```

```
        BooksInfoRequest  request     = this.xmlToInfoRequest(domRequ

        if (logger.isDebugEnabled()){
            logger.debug("PETICION: categoria=" + request.getCategor
        }

        BooksInfoResponse response    = procesor.process(request);

        if (logger.isDebugEnabled()){
            int numLibros = 0;
            if (response.getLibros() != null){
                numLibros = response.getLibros().size();
            }
            logger.debug("RESPUESTA: Número de libros: " + numLibros
        }

        Element           domResponse = this.responseToXml(response,

        return domResponse;
    }

    /**
     * @param request Elemento XML <BooksInfoRequest ...>
     * @return Genera una instancia de BooksInfoRequest a partir de u
     */
    private BooksInfoRequest xmlToInfoRequest(Element request){
        String categoria = request.getElementsByTagName("categoria")
        String nivel     = request.getElementsByTagName("nivel").iter

        BooksInfoRequest bookInfoRequest = new BooksInfoRequest();

        bookInfoRequest.setCategoria(categoria);
        bookInfoRequest.setNivel(nivel);

        return bookInfoRequest;
    }

    /**
     * @param request Elemento XML <BooksInfoRequest ...>
     * @return Genera una instancia de BooksInfoRequest a partir de u
     */
    private Element responseToXml(BooksInfoResponse response, Documen
        Element     root    = document.createElementNS("http://www.ac
        List<Libro> libros  = response.getLibros();

        if (libros != null){
            Iterator<Libro> iteLibros = null;
            Libro           libro     = null;
            Element         domLibro  = null;

            iteLibros = libros.iterator();
            while (iteLibros.hasNext()){
                libro    = iteLibros.next();
                domLibro = this.bookToXml(document, libro);
                root.appendChild(domLibro);
            }
```

```
        }

        return root;
    }

    /**
     *
     * @param document Document para construir el DOM
     * @param libro    Objeto a convertir a XML
     * @return Devuelve el objeto libro en XML.
     */
    private Element bookToXml(Document document, Libro libro){
        Element domLibro = document.createElement("libro");

        this.addHijo(document, domLibro, "editorial", libro.getEditor
        this.addHijo(document, domLibro, "titulo",  libro.getTitulo()
        this.addHijo(document, domLibro, "paginas", String.valueOf(l
        this.addHijo(document, domLibro, "precio",  String.valueOf(l

        return domLibro;
    }

    /**
     * Añade una nueva propiedad <tag>valor</tag> al elemento <padre>
     * @param document Para crear elementos
     * @param padre    La nueva propiedad será agregada a este elemen
     * @param tag      Nombre de la propiedad
     * @param valor    Valor de la propiedad
     */
    private void addHijo(Document document, Element padre, String tag
        Element domNombre = document.createElement(tag);
        Text    domValor  = document.createTextNode(valor);

        domNombre.appendChild(domValor);
        padre.appendChild(domNombre);
    }
}
```

Archivo de configuración de Log4J: `/WEB-ING/log4j.xml`:

A continuación exponemos el archivo de configuración de Log4J. Los mensajes con nivel WARN o superior irán a parar a un archivo y el resto (de cualquier nivel DEBUG, INFO, etc.) a otro.

Shell

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

    <!-- Los errores irán a parar al siguiente archivo:  -->
    <appender name="error_file" class="org.apache.log4j.RollingFileApp
```

```
        <param name="File" value="bibliotecaWS_error.log" />
        <param name="MaxFileSize" value="2000000" />
        <param name="MaxBackupIndex" value="5" />
        <param name="Threshold" value="WARN" />

        <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>

    <!-- Todas las trazas (debug, warn, error, etc.) irán a parar a es
    <appender name="debug_file" class="org.apache.log4j.RollingFileApp
        <param name="File" value="bibliotecaWS_debug.log" />
        <param name="MaxFileSize" value="5000000" />
        <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>

    <!-- Habilitamos sólo los logs de nivel warning o superior para to
    <root>
        <level value="warn" />
        <appender-ref ref="error_file" />
    </root>

    <!-- Habilitamos todos los LOGS de todas las clases del paquete co
    <category name="com.autentia.tutoriales.spring.ws">
        <priority value="debug" />
        <appender-ref ref="debug_file" />
    </category>
</log4j:configuration>
```

Archivo de configuración de Spring 2 `(\WEB-INF\bibliotecaWS-servlet.xml)`:

Atención, el **nombre del archivo es importante**, debe ser igual que el nombre del servlet que verémos más adelante y luego “-servlet.xml”.

El archivo está autocomentado.

Shell

```
<?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/util
```

```
    <!-- Realiza la lógica de negocio de consulta de libros en base a
    <bean name="requestProcesor" class="com.autentia.tutoriales.spring

    <!-- EndPoint del WS: Recibirá la petición del WS (WSDL operation)
    <bean id="bibliotecaWSEndpoint" class="com.autentia.tutoriales.spr
        <property name="procesor" ref="requestProcesor" />
    </bean>

    <!-- Indicamos que hable SOAP 1.2 -->
    <bean id="messageFactory" class="org.springframework.ws.soap.saaj
        <property name="soapVersion">
            <util:constant static-field="org.springframework.ws.soap.S
        </property>
    </bean>

    <!--
        payloadMapping: Redirige mensajes XML entrantes hacia el EndPo
        en función del PayLoad del SOAP:Body del mensaje.
        (Otra opción podría ser a través de la cabecera SOAPAction: or
     -->
     <bean id="payloadMapping" class="org.springframework.ws.server.en
        <property name="endpointMap">
            <map>
                <!-- ¡¡ Ojo !! No dejar espacios entre el namespace y
                <entry key="{http://www.adictosaltrabajo.com/spring/ws
                value-ref="bibliotecaWSEndpoint"/>
            </map>
        </property>

        <!-- Validación de peticiones y/o respuestas -->
        <property name="interceptors">
           <list>
               <ref bean="validatingInterceptor"/>
           </list>
        </property>
     </bean>

     <!-- Para validar las peticiones y/o respuestas -->
     <bean id="validatingInterceptor"
        class="org.springframework.ws.soap.server.endpoint.interceptor
         <property name="schema" value="/WEB-INF/bibliotecaWS.xsd"/>
         <property name="validateRequest"  value="true"/>
         <property name="validateResponse" value="false"/>
     </bean>


    <!-- Este bean convertirá cualquier excepción Java en un fallos SO
    <bean id="endpointExceptionResolver"
        class="org.springframework.ws.soap.server.endpoint.SoapFaultMa
      <property name="defaultFault"  value="RECEIVER,Server error" />
      <property name="exceptionMappings">
        <props>
          <prop key="org.springframework.oxm.UnmarshallingException">S
          <prop key="org.springframework.oxm.ValidationFailureExceptio
        </props>
```

```
      </property>
    </bean>


    <!--  DynamicWsdl11Definition genera automáticamente el WSDL del s
    <bean id="bibliotecaWS" class="org.springframework.ws.wsdl.wsdl11
      <property name="schema">
        <bean class="org.springframework.xml.xsd.SimpleXsdSchema">
          <property name="xsd" value="/WEB-INF/bibliotecaWS.xsd"/>
        </bean>
      </property>

      <property name="createSoap12Binding" value="true"/>
      <property name="portTypeName" value="bibliotecaWS"/>
      <property name="locationUri"  value="http://localhost:8080/bibl
    </bean>
</beans>
```

Archivo de configuración y despliegue de la aplicación `(/WEB-INF/web.xml)`:

Las peticiones SOAP serán atendidas por el servlet `org.springframework.ws.transport.http.MessageDispatcherServlet`

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.su
    version="2.4">
    <display-name>Autentia. Ejemplo de Spring-WS</display-name>

    <!-- Configuración de Log4J -->
    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>/WEB-INF/log4j.xml</param-value>
    </context-param>

    <!-- Configuración de Log4J -->
    <listener>
        <listener-class>org.springframework.web.util.Log4jConfigListe
    </listener>

    <!-- Servlet que atenderá y redifirá penticiones SOAP -->
    <servlet>
        <servlet-name>bibliotecaWS</servlet-name>
        <servlet-class>org.springframework.ws.transport.http.MessageD

        <!-- Para que genere el WSDL desde el XSD -->
        <init-param>
            <param-name>transformWsdlLocations</param-name>
            <param-value>true</param-value>
```

```
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>bibliotecaWS</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>

    <!-- Para que devuelva el WSDL  -->
    <servlet-mapping>
        <servlet-name>bibliotecaWS</servlet-name>
        <url-pattern>*.wsdl</url-pattern>
    </servlet-mapping>
</web-app>
```

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.

Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http:/
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.autentia.tutoriales</groupId>
    <artifactId>bibliotecaWS</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>BibliotecaWS con Spring-WS</name>
    <url>http://www.adictosaltrabajo.com</url>

    <!-- Sintáxis Java 5 -->
     <build>
         <plugins>
             <plugin>
                 <artifactId>maven-compiler-plugin</artifactId>
                 <configuration>
                     <source>1.5</source>
                     <target>1.5</target>
                     <encoding>UTF-8</encoding>
                 </configuration>
             </plugin>

         </plugins>
     </build>

    <dependencies>
        <!-- Spring WS: Maven gestionará automáticamente todas sus dep
        <dependency>
```

```
            <groupId>org.springframework.ws</groupId>
            <artifactId>spring-ws-core</artifactId>
            <version>1.5.6</version>
        </dependency>
    </dependencies>
</project>
```

Construcción y despliegue la aplicación:

A continuación ejecutamos el siguiente comando Maven: `mvn package` y desplegamos el archivo generado `target\bibliotecaWS-1.0-SNAPSHOT.war` en nuestro servidor preferido (JBoss, Tomcat, WebLogic, WebSphere, Jetty, etc..).

Ejemplo de construcción de un cliente (Axis2) para probar el servicio web.

Este apartado da por sentado que el usuario ya sabe algo de Axis2, así que no voy a explicar con sumo detalle que es cada cosa (puedes consultar otros tutoriales en https://www.adictosaltrabajo.com de como se instala, configura, etc).

Como dije anteriormente, el servicio Web está desplegado y es capaz de autodescribirse a generando su propio WSDL, pues bien vamos a crear un cliente automáticamente desde su WDSL.

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

```
%AXIS2_HOME%/bin/wsdl2java -sp -s -p
com.autentia.tutoriales.spring.ws.cliente -uri
http://localhost:8080/bibliotecaWS/bibliotecaWS.wsdl
```

En donde:

- -sp: Por el namespace axis2 por defecto pone un namespace ns1 pero nuestro en el WS el Payload no está qualificado (no tiene namespace). Puedes verlo tu mismo copiando y pegando la URL en tu navegador...
- -s: No queremos ni necesitamos que nos genere funcionalidad de invocación asíncrona.
- -p: En que paquete deseamos que nos genere las clases.

Invocando el servicio Web.

`com.autentia.tutoriales.spring.ws.clie`

Shell

```
package com.autentia.tutoriales.spring.ws.cliente;

import com.autentia.tutoriales.spring.ws.cliente.BibliotecaWSServiceS

/**
 * Ejemplo de invocación del WS de consulta de libros
 * @author Carlos García. Autentia
 * @see http://www.mobiletest.es
 */
public class BibliotecaWSApp {
    public static void main(String[] args) throws Exception {
        BibliotecaWSServiceStub stub = new BibliotecaWSServiceStub();
        BibliotecaWSServiceStub.BooksInfoRequest  peticion  = new Bib

        peticion.setCategoria("java");
        peticion.setNivel(BibliotecaWSServiceStub.NivelType.avanzado)

        BibliotecaWSServiceStub.BooksInfoResponse respuesta = stub.Boo

        Libro_type0[] libros = respuesta.getLibro();
        if (libros != null){
             for (int i = 0, lcount = libros.length; i < lcount; i++)
                 System.out.println(libros[i].getEditorial() + " " + 
             }
        } else {
            System.out.println("No hay libros");
        }
    }
}
```

Y para terminar, al ejecutar la aplicación anterior, nos produce la siguiente salida:

Shell

```
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
```

Ejemplo de construcción de un cliente (con Spring) para probar el servicio web.

Este apartado está descrito en el siguiente tutorial https://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=spring_ws_client_example1.

Referencias

- http://static.springframework.org/spring-ws/docs/0.9.1/spring-ws-reference.pdf
- http://static.springsource.org/spring-ws/sites/1.5/reference/html/tutorial.html

Conclusiones

Bueno, como veis Spring no deja de sorprendernos en cuanto a su potencia y ventajas en el desarrollo de software de calidad (bajo acomplamiento, alta cohesión, etc.)

Resaltar que Spring también proporciona mucha funcionalidad para consumir servicios Web (hacer clientes), pero prefiero dejar esto para otro tutorial o para que el lector investige al respecto, haga un tutorial y nos lo mande para que lo publiquemos :-).

Carlos García Pérez. Creador de MobileTest, un complemento educativo para los profesores y sus alumnos.

cgpcosmad@gmail.com

Carlos García Pérez

Técnico especialista en informática de empresa (CEU). Ingeniero Técnico en Informática de Sistemas (UPM) Creador de MobileTest, Haaala!, Girillo, toi18n. Charla sobre desarrollo de aplicaciones en Android. @cgpcosmad