WebServices en Spring (Spring-WS)

Los **[WebServices](http://www.desarrolloweb.com/articulos/957.php" \o "WebServices" \t "_blank)** nos permiten comunicar diferentes aplicaciones de forma distribuida entre si, esta tecnología empieza a ser muy demandada por las empresas del mundo de la programación, ya que permiten centralizar de forma muy efectiva o distribuir los puntos de acceso de diferentes aplicaciones según convenga de una forma muy dinámica.

A esta tecnología se le puede dar diversas utilidades, pero generalmente pensaremos en un servidor que contiene información, a la que el servidor permite el acceso por parte de otras aplicaciones a través de un[**WebServices**](http://www.desarrolloweb.com/articulos/957.php). Esta es la forma que nos encontraremos mas habitualmente, pero evidentemente la tecnología nos permite mucho mas.

En este ejemplo vamos a desarrollar nuestro **[WebServices](http://www.desarrolloweb.com/articulos/957.php" \o "WebServices" \t "_blank)** con [**Spring-WebServices**](http://static.springsource.org/spring-ws/sites/1.5/), por varias razones, la primera la de seguir mi política personal de desarrollar aplicaciones con este Framework, ya que las prestaciones y opciones que nos brinda van a repercutir directamente en la calidad final de nuestra aplicación.

Para este ejemplo van a ser necesarias las siguientes librerías aparte de las de Spring que venimos usando.

* [**spring-ws-v.v.v-all**](http://static.springsource.org/spring-ws/sites/1.5/)
* [**wsdl4j-v.v.v**](http://sourceforge.net/projects/wsdl4j/)
* [**castor-v.v**](http://www.castor.org/)
* [**castor-v.v-core**](http://www.castor.org/)

Lo primero que tenemos que hacer es declarar en nuestro web.xml nuestro Servlet que se encargara de mapear nuestro **[WebServices](http://www.desarrolloweb.com/articulos/957.php" \o "WebServices" \t "_blank)**, establecemos que nuestro Servlet se hará cargo de las peticiones que contengan la cadena **/services/**, y le activamos la opción **transformWsdlLocations** que nos servirá para que genere automáticamente el [**WSDL**](http://www.w3.org/TR/wsdl) a partir de nuestro [**XML Schema**](http://www.w3.org/XML/Schema).

|  |
| --- |
| <servlet>      <servlet-name>spring-ws</servlet-name>      <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>      <init-param>          <param-name>transformWsdlLocations</param-name>          <param-value>true</param-value>      </init-param>  </servlet>  <servlet-mapping>      <servlet-name>spring-ws</servlet-name>      <url-pattern>/services/\*</url-pattern>  </servlet-mapping> |

El ejemplo que vamos a desarrollar va a ser muy simple, pero nos servirá para entender perfectamente el funcionamiento de [**Spring-WS**](http://static.springsource.org/spring-ws/sites/1.5/), desde nuestro cliente (el cual implementaremos en otra entrada) se solicitara la suma de dos números enteros cualesquiera.

Pensemos en que queremos recibir una petición del siguiente tipo.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <SumaRequest>      <sumando1>primerEntero</sumando1>      <sumando2>segundoEntero</sumando2>  </SumaRequest> |

Y que recibiremos una respuesta del siguiente tipo.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>    <SumaResponse>    <resultado>resultadoEntero</resultado>    </SumaResponse> |

Lo siguiente que tenemos que hacer es crear un [**XML Schema**](http://www.w3.org/XML/Schema) (.xsd) en el que especificaremos como se va a realizar la petición y la respuesta, a nuestra petición le pondremos el sufijo **Request**, y a nuestra respuesta el sufijo **Response** en su element name, ya que estos sufijos serán utilizados por [**Spring-WS**](http://static.springsource.org/spring-ws/sites/1.5/) para generar automáticamente el [**WSDL**](http://www.w3.org/TR/wsdl) de nuestra aplicación.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>    <xs:schema xmlns:xs="<http://www.w3.org/2001/XMLSchema>"      elementFormDefault="qualified"      targetNamespace="<http://programacionj2ee.com/ws/schemas>"      xmlns:ws="<http://programacionj2ee.com/ws/schemas>">        <xs:element name="SumaRequest">          <xs:complexType>              <xs:all>                  <xs:element name="sumando1" type="xs:Integer" />                  <xs:element name="sumando2" type="xs:Integer"/>              </xs:all>          </xs:complexType>      </xs:element>        <xs:element name="SumaResponse">          <xs:complexType>              <xs:all>                  <xs:element name="resultado" type="xs:Integer" />              </xs:all>          </xs:complexType>      </xs:element>    </xs:schema> |

Lo siguiente es crear nuestro **PayLoad**, es la clase que se va a encargar de resolver la petición, al principio comencé utilizando **JDOM** y **Xpath** para serializar “manualmente” el xml de la petición, pero hace poco vi una solución en **[dosideas](http://www.programacionj2ee.com/webservices-en-spring-spring-ws/www.dosideas.com" \o "dosideas" \t "_blank)**, utilizando un **Marshaller** con la librería [**Castor**](http://www.castor.org/) y la clase**AbstractMarshallingPayloadEndpoint** de [**Spring-WS**](http://static.springsource.org/spring-ws/sites/1.5/) que nos permite abstraernos totalmente de la serialización y deserialización, lo que es sin ninguna duda una grandísima ventaja.

En nuestra clase **PayLoad** tenemos un constructor el cual mediante inyección de dependencias recibe la configuración del **Marshaller** de la clase.

|  |
| --- |
| public class SumaEndPoint extends AbstractMarshallingPayloadEndpoint {        public SumaEndPoint(Marshaller marshaller){          super(marshaller);      }        protected Object invokeInternal(Object sm) throws Exception {          Sumandos sumandos = new Sumandos();          Resultado resultado = new Resultado();            sumandos = (Sumandos) sm;            resultado.setResultado(sumandos.getSumando1() + sumandos.getSumando2());            return resultado;      }  } |

La clase **Sumandos**.

|  |
| --- |
| public class Sumandos {        private int sumando1;      private int sumando2;        public int getSumando1() {          return sumando1;      }        public void setSumando1(int sumando1) {          this.sumando1 = sumando1;      }        public int getSumando2() {          return sumando2;      }        public void setSumando2(int sumando2) {          this.sumando2 = sumando2;      }  } |

La clase **Resultado**.

|  |
| --- |
| public class Resultado {        private int resultado;        public int getResultado() {          return resultado;      }        public void setResultado(int resultado) {          this.resultado = resultado;      }    } |

El objeto **Marshaller** que recibe nuestro **PayLoad** esta configurado como un bean en el **spring-ws-servlet.xml**que veremos mas adelante, este hace uso del siguiente archivo de mapeo de clases para conocer la estructura de la petición y poder mapearla a nuestro objeto **Sumandos**. En el parametro xml debemos poner el nombre especificado para nuestra petición en el [**XML Schema**](http://www.w3.org/XML/Schema).

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE mapping PUBLIC "-//EXOLAB/Castor Mapping DTD Version 1.0//EN" "<http://castor.org/mapping.dtd>">    <mapping>      <description>Serializador de objetos suma.</description>        <class name="com.programacionj2ee.springws.beans.Sumandos">          <map-to xml="SumaRequest" ns-uri="<http://programacionj2ee.com/ws/schemas>" ns-prefix="xs" />            <field name="sumando1" type="integer">              <bind-xml name="sumando1" />          </field>            <field name="sumando2" type="integer">              <bind-xml name="sumando2" />          </field>      </class>        <class name="com.programacionj2ee.springwsclient.beans.Resultado">          <map-to xml="SumaResponse" ns-uri="<http://programacionj2ee.com/ws/schemas>" ns-prefix="xs" />            <field name="resultado" type="integer">             <bind-xml name="resultado" />          </field>      </class>    </mapping> |

Con esto ya tenemos todos los archivos necesarios para que nuestro **[WebServices](http://www.desarrolloweb.com/articulos/957.php" \o "WebServices" \t "_blank)** funcione, ahora todo lo que nos queda es crear los beans necesarios en nuestro **spring-ws-servlet.xml**.

El bean **payloadMapping** es el encargado de recoger la peticion y enviarla al **PayLoad** correspondiente.

|  |
| --- |
| <bean id="payloadMapping" class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">      <property name="mappings">          <props>              <prop key="{<http://programacionj2ee.com/ws/schemas>}SumaRequest">sumaEndpoint</prop>          </props>      </property>      <property name="interceptors">          <list>              <ref bean="validadorInterceptor"/>          </list>      </property>  </bean> |

El bean **validadorInterceptor** configura nuestro interceptor.

|  |
| --- |
| <bean id="validadorInterceptor"          class="org.springframework.ws.soap.server.endpoint.interceptor.PayloadValidatingInterceptor">      <property name="schema" value="/WEB-INF/suma.xsd"/>      <property name="validateRequest"  value="true"/>      <property name="validateResponse" value="true"/>  </bean> |

El bean **messageFactory** especifica la version [**SOAP**](http://www.w3.org/TR/soap/) que se va a utilizar.

|  |
| --- |
| <bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">      <property name="soapVersion">          <util:constant static-field="org.springframework.ws.soap.SoapVersion.SOAP\_11"/>      </property>  </bean> |

Los bean **suma** y **sumaSchema** los utiliza [**Spring-WS**](http://static.springsource.org/spring-ws/sites/1.5/) para generar el [**WSDL**](http://www.w3.org/TR/wsdl) automáticamente a partir de nuestro [**XML Schema**](http://www.w3.org/XML/Schema).

|  |
| --- |
| <bean id="suma" class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition">      <property name="schema" ref="sumaSchema"/>      <property name="portTypeName" value="SumaRequest"/>      <property name="locationUri" value="/services/sumaService"/>      <property name="targetNamespace" value="<http://programacionj2ee.com/ws/schemas>"/>  </bean>    <bean id="sumaSchema" class="org.springframework.xml.xsd.SimpleXsdSchema">      <property name="xsd" value="/WEB-INF/suma.xsd"/>  </bean> |

El bean **sumaEndpoint** configura nuestra clase **PayLoad**, a la que se le pasa mediante inyeccion de dependencias un **Marshaller** configurado.

|  |
| --- |
| <bean id="sumaEndpoint" class="com.programacionj2ee.springws.webservices.SumaEndPoint">      <constructor-arg ref="castorMarshaller" />  </bean> |

El bean **castorMarshaller** configura nuestro **Marshaller** haciendo uso del castor.xml configurado anteriormente.

|  |
| --- |
| <bean id="castorMarshaller" class="org.springframework.oxm.castor.CastorMarshaller" >      <property name="mappingLocation" value="classpath:castor.xml" />  </bean> |

Con esta configuración, Spring-WS creara y publicara el WSDL de nuestra aplicación en la siguiente dirección.

|  |
| --- |
| <http://host:port/pjcomspringws/services/suma.wsdl> |

**Código fuente:**  
[**WebServices en Spring (Spring-WS)**](https://docs.google.com/open?id=0B_QXf3LoKfiIc0ZSZ2JxTzRJdjQ) | Netbeans 6.7  
**Contraseña:** www.programacionj2ee.com\_sws\_93/strong