



Desarrollo Avanzado Web

Un portal dedicado a brindar orientacion de tecnologias utilizadas en la actualidad para el desarrollo de aplicaciones web.

jueves, 24 de octubre de 2013

Spring: Spring-WS con JAXB + Hibernate + DAO

Recorriendo Spring

- [Spring: Introducción + Inyección de dependencias](#)
- [Spring: AOP Programación Orientada a Aspectos](#)
- [Spring: AOP con Aspect mediante archivos xml y Anotation.](#)
- [Spring: Mejorando JDBC](#)
- [Spring Transaction](#)
- [Spring: Mejorando el Trabajo con Hibernate ORM](#)
- [Spring MVC una recorrida contando lo bueno y lo malo \(En contruccion\)](#)
- [Spring Security, pensando en seguridad \(En contruccion\)](#)
- [Spring: Remoting gestionando conexiones](#)
- [Spring: Spring-WS con JAXB + Hibernate + DAO](#)
- [Spring: JMS](#)
- [Spring: Automatizando pruebas con JUNIT STUBS Mock \(En contruccion\)](#)

Las Aplicaciones empresariales modernos rara vez están solas, a menudo se basan en datos, proporcionados por servicios externos.

Para favorecer la comunicación debe haber un protocolo de comunicación algún tipo, una forma estándar de enviar y recibir mensajes en un formato que sea reconocido y apoyado por todas las principales plataformas.

SOAP (Simple Object Application Protocol) es un protocolo de este tipo que permite a las aplicaciones comunicarse mediante el intercambio de mensajes en un formato XML estándar.

Los Servicios web SOAP proporciona un mecanismo de integración entre plataformas, por ejemplo, los servicios web SOAP se usan comúnmente para integrar .NET y Java.

Casi todas las plataformas y marcos modernos (Java, .NET, Ruby, PHP, etc) ofrecen amplias bibliotecas y herramientas que permiten a los desarrolladores exponer y consumir servicios SOAP rápida y fácilmente.

Para este ejemplo recorreremos las virtudes de Spring para construir, desplegar y probar un servicio SOAP para la recuperación sencilla de una cuenta bancaria.

Estos son algunos de los hitos tecnológicos que abordaremos

- Spring 3.1 para soporte de servicios web.
- Maven para la gestión de proyectos y dependencias
- Tomcat como contenedor
- SoapUI para realizar pruebas sobre nuestro servicio.
- Hibernate para trabajar en nuestra persistencia
- Mysql para nuestro motor de base de datos

Patrones de diseño para nuestra capa de negocio como siempre utilizaremos

- DAO
- Service Facade

Existen dos enfoques fundamentales para la creación de servicios web, **“First Contract”** o **“Bottom Up”** o **“Last Contract”** o **“Top Down”**.

Es decir, crear primero los XSD y los WSDL y luego generar las clases Java o bien generar las clases Java y que luego se generen los XSD y los WSDL.

El último enfoque consiste en tomar Contrato código existente y la generación de un contrato de servicio directamente desde que el código con el fin de exponerlo como una interfaz SOAP. Hay una variedad de marcos de Java por ahí (Axis2, XFire etc) que proporcionan estas herramientas Java2WSDL, para generar rápidamente los servidores proxy secundarios, clases Servlet necesarios para exponer un servicio SOAP.

El primer enfoque consiste en la definición del contrato de servicio antes de implementar el servicio. Esto significa que describe los parámetros de servicio y tipos devueltos mediante

Categorías

[.Net](#)

[Java](#)

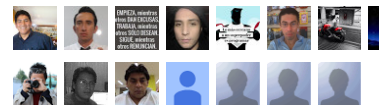
[Spring](#)

[android](#)

Recent Posts

Seguidores

Seguidores (15)



[Seguir](#)

Archivo del blog

► 2015 (8)

► 2014 (4)

▼ 2013 (33)

► diciembre (1)

► noviembre (8)

▼ octubre (6)

[Usar SVN de Google Code con Eclipse](#)

[Spring: Spring-WS con JAXB + Hibernate + DAO](#)

[Spring: Mejorando el Trabajo con Hibernate ORM](#)

[Hibernate: Dao Generic para Implementar Crud en mi...](#)

[Spring: Como manejar Transacciones y sus beneficio...](#)

[Spring: Como Mejorar JDBC](#)

XSD (XML Schema Definiciones), luego se usa el XSD para construir una WSDL (web service definition language) esto establece un contrato público o descripción del servicio. Sólo después de que el contrato de servicios se ha definido claramente , es la implementación del servicio.

Este artículo describirá un primer contrato de servicio , ya que este es el enfoque preferido por varias razones que analizaremos a lo largo del tutorial.

Conociendo un poco de XSD

XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

Los esquemas

Sería un loco si no arrancar diciendo que todo los elementos de XSD se encuentran contenido por un esquema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
.....Contenido de nuestro xsd.....
</xs:schema>
```

y que los mismos tienen atributos como:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
indica que los elementos y tipos de datos utilizados en el esquema
```

vienen del espacio de nombres "http://www.w3.org/2001/XMLSchema".

Nota: También especifica que los elementos y tipos de datos que vienen del espacio de nombres "http://www.w3.org/2001/XMLSchema" deben llevar el prefijo xs

```
targetNamespace="http://www.w3schools.com"
Indica que los elementos definidos por este esquema (note, to, from, heading, body.)
biene del namespace "http://www.w3schools.com"
```

xmlns = "mi direccion url" indica simplemente un namespace por defecto.

Un ejemplo de un esquema sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://com/company/webservices/account"
targetNamespace="http://com/company/webservices/account"
elementFormDefault="qualified" attributeFormDefault="unqualified">
.....
.....
.....
.....
..
</xs:schema>
```

Nota: en pocas palabras en el esquema vamos a proporcionar URL importantes que a lo largo del tutorial entenderemos con mayor detalle.

Definiendo un Elemento

Es un elemento que contiene únicamente texto, no contiene otro elemento o atributo. El texto puede ser de diferentes tipos.

Definiendo un ejemplo

```
<xs:element name="xxx" type="yyy"/>
```

Nota: Donde xxx es el nombre del elemento y yyy es el tipo del elemento.

Los tipos de elementos disponibles son:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Algunos ejemplos que nos ayudaran a entender el concepto

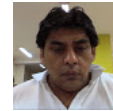
```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Definiendo Atributos

En primera instancia un elemento simple no puede tener atributos, si un elemento tiene atributos es considerado un **complex type**.

- ▶ septiembre (4)
- ▶ agosto (1)
- ▶ mayo (2)
- ▶ marzo (1)
- ▶ febrero (10)
- ▶ 2012 (6)

Datos personales



Diego Herrera



Seguir

151

[Ver todo mi perfil](#)

Definiendo un ejemplo

```
<xs:attribute name="xxx" type="yyy"/>
```

Nota: Donde xxx es el nombre del elemento y yyy es el tipo del elemento.

Los tipos de elementos disponibles son:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Algunos ejemplos

```
<xs:attribute name="lang" type="xs:string"/>
```

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Definiendo Tipos Complejos

Una Elemento complejo es un elemento que contiene otros elementos o atributos.

Definiendo un complexType contenido por medio de un elemento

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Nota: Podemos observar que el complextype esta contenido dentro de un element, ahora si recordamos lo que mencionamos anterior un element que contiene a otros es un un elemento complejo.

Definiendo un complextype relacionado a un element.

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Nota: ya aquí se tenía claro el concepto de desacoplamiento, por que definimos al tipo complejo con un nombre, el que los elementos podrán referencias de manera directa.

Cual es el beneficio ¿? Esta a la vista brindado por el desacoplamiento que puede ser referenciado por N elementos.

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Nota: Un tema para tener en cuenta que los elementos pertenecientes al tipo complejo siempre estarán contenido por una secuencia.

Definiendo Restricciones

Las restricciones se utilizan para definir los valores aceptables para elementos o atributos XML.

Definiendo un elemento con restricción.

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
</xs:simpleType>
</xs:element>
```

Nota: es claro que el elemento edad por supuesto es Integer y que como mínimo puede tener un valor 0 y como máximo 120.

Ahora es claro que voy a poder plantear el desacoplamiento del ejemplo anterior, asignándole a tipo simple un nombre que será referenciado por el elemento en cuestión.

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Para limitar el contenido de un elemento XML para un conjunto de valores aceptables, nos gustaría utilizar la restricción de enumeración.

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Nota: el claro que limite a car a una lista de posibles opciones.

Ahora la verdad es una de las formas de limitar los valores por XSD es la utilización de expresiones regulares

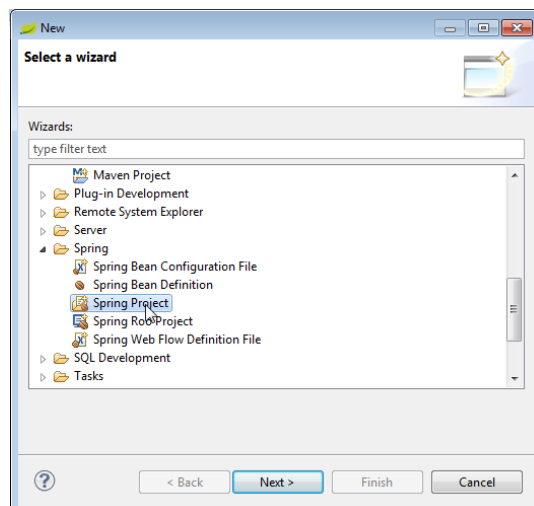
```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Nota: y es claro que limito al element letter a que pueda aceptar solo letras.

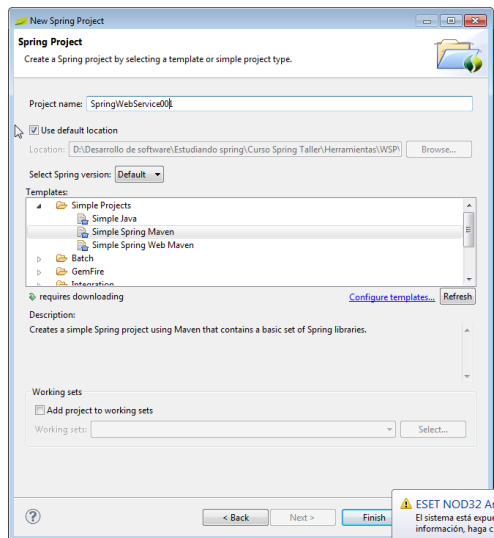
Ahora podríamos mostrar un poco de esto pero el objetivo no es convertirnos en expertos en XSD sino que puedan seguir el tutorial, pero si realmente están interesados en indagar un poco mas sobre el tema los invito a recorrer internet donde van a encontrar mucho material sobre el tema.

Primero creamos un nuevo proyecto.

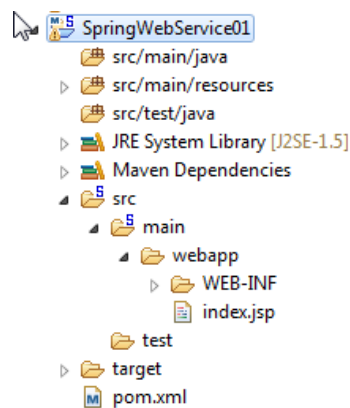
Como la mayoría de los proyectos, realizaremos el proceso de creación del proyecto.



Lo siguiente es seleccionar la template para Simple Spring Web Maven.



Lo que desde luego nos genera una estructura de base como la siguiente.



Entendiendo las dependencias de mi proyecto.

En esta oportunidad solo analizaremos las necesidades a nivel de dependencias para mi proyecto.

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>com.company</groupId>
4 <artifactId>SpringWebService01</artifactId>
5 <version>0.0.1-SNAPSHOT</version>
6 <packaging>war</packaging>
7 <properties>
8 <spring.version>3.0.5.RELEASE</spring.version>
9 <spring.ws.version>2.0.0.RELEASE</spring.ws.version>
10 <log4j.version>1.2.16</log4j.version>
11 <context.path>SpringWebService01</context.path>
12 </properties>
13 <build>
14 <plugins>
15 <plugin>
16 <groupId>org.codehaus.mojo</groupId>
17 <artifactId>jaxb2-maven-plugin</artifactId>
18 <version>1.4</version>
19 <executions>
20 <execution>
21 <goals>
22 <goal>xjc</goal>
23 </goals>
24 <phase>generate-sources</phase>
25 </execution>
26 </executions>
27 <configuration>
28 <clearOutputDir>>false</clearOutputDir>
29 <outputDirectory>src/main/java</outputDirectory>
30 <schemaDirectory>src/main/webapp/schemas</schemaDirectory>
31 <includeSchema>/**/*.xsd</includeSchema>
32 <bindingDirectory>src/main/resources/bindings</bindingDirectory>
33 <enableIntrospection>>false</enableIntrospection>
34 </configuration>
35 </plugin>
36 <plugin>
37 <groupId>org.apache.maven.plugins</groupId>
38 <artifactId>maven-war-plugin</artifactId>
39 <configuration>
40 <warName>${context.path}</warName>
41 </configuration>
42 </plugin>

```

```

43 </plugins>
44 </build>
45 <dependencies>
46 <dependency>
47 <groupId>log4j</groupId>
48 <artifactId>log4j</artifactId>
49 <version>${log4j.version}</version>
50 </dependency>
51 <dependency>
52 <groupId>org.springframework</groupId>
53 <artifactId>spring-orm</artifactId>
54 <version>${spring.version}</version>
55 </dependency>
56 <dependency>
57 <groupId>org.springframework</groupId>
58 <artifactId>spring-core</artifactId>
59 <version>${spring.version}</version>
60 </dependency>
61 <dependency>
62 <groupId>org.springframework</groupId>
63 <artifactId>spring-context</artifactId>
64 <version>${spring.version}</version>
65 </dependency>
66 <dependency>
67 <groupId>org.springframework</groupId>
68 <artifactId>spring-beans</artifactId>
69 <version>${spring.version}</version>
70 </dependency>
71 <dependency>
72 <groupId>org.springframework</groupId>
73 <artifactId>spring-aop</artifactId>
74 <version>${spring.version}</version>
75 </dependency>
76 <dependency>
77 <groupId>org.springframework</groupId>
78 <artifactId>spring-aspects</artifactId>
79 <version>${spring.version}</version>
80 </dependency>
81 <dependency>
82 <groupId>commons-collections</groupId>
83 <artifactId>commons-collections</artifactId>
84 <version>3.2</version>
85 </dependency>
86 <dependency>
87 <groupId>org.springframework</groupId>
88 <artifactId>spring-oxm</artifactId>
89 <version>${spring.version}</version>
90 </dependency>
91 <dependency>
92 <groupId>org.springframework.ws</groupId>
93 <artifactId>spring-ws-core</artifactId>
94 <version>${spring.ws.version}</version>
95 </dependency>
96 <dependency>
97 <groupId>org.apache.ws.commons.schema</groupId>
98 <artifactId>XmlSchema</artifactId>
99 <version>1.4.3</version>
100 </dependency>
101 <dependency>
102 <groupId>cglib</groupId>
103 <artifactId>cglib</artifactId>
104 <version>2.2.2</version>
105 </dependency>
106 <dependency>
107 <groupId>org.springframework</groupId>
108 <artifactId>spring-jdbc</artifactId>
109 <version>${spring.version}</version>
110 </dependency>
111 <dependency>
112 <groupId>mysql</groupId>
113 <artifactId>mysql-connector-java</artifactId>
114 <version>5.1.17</version>
115 <type>jar</type>
116 <scope>compile</scope>
117 </dependency>
118 <!-- Hibernate framework -->
119 <dependency>
120 <groupId>org.hibernate</groupId>
121 <artifactId>hibernate-core</artifactId>
122 <version>3.6.3.Final</version>
123 </dependency>
124 <dependency>
125 <groupId>javassist</groupId>
126 <artifactId>javassist</artifactId>
127 <version>3.12.1.GA</version>
128 </dependency>
129
130 <!-- Hibernate library dependency start -->
131 <dependency>
132 <groupId>dom4j</groupId>
133 <artifactId>dom4j</artifactId>
134 <version>1.6.1</version>
135 </dependency>
136
137 <dependency>
138 <groupId>commons-logging</groupId>
139 <artifactId>commons-logging</artifactId>
140 <version>1.1.1</version>
141 </dependency>
142 <dependency>
143 <groupId>antlr</groupId>
144 <artifactId>antlr</artifactId>
145 <version>2.7.7</version>
146 </dependency>
147 <!-- Hibernate library dependency end -->
148 </dependencies>
149 </project>

```

Dentro de las dependencias registradas vamos a encontrar algunos grandes grupos y por supuesto las necesidades de

cada uno.

Generales: Gestión General de la aplicación.

- log4j: gestion de log en la aplicacion
- commons-collections: Gestión de collection
- XmlSchema: libreria necesaria para la gestion de ws.
- mysql-connector-java: Conector java para MYSQL.

Spring: Librerías necesarias para el funcionamiento de IOC.

- spring-orm: facilita la comunicación con ORM
- spring-core: el corazon de spring.
- spring-context: gestion de conexto.
- spring-beans: gestion de beans.
- spring-aop: gestion de aspectos para temas transversales.
- spring-aspects: gestion de aspectos.
- spring-oxm: permite la gestión de documento XML desde/hacia un objeto, brinda una abstracción de trabajo con JAXB.
- spring-ws-core: Liberia necesaria para la gestión de WS.
- spring-jdbc: Libreria necesaria para gestionar la conexion JDBC.

Hibernate: Librerías necesarias para el funcionamiento de Hibernate.

- hibernate-core: Corazon de hibernate
- javassist: dependencia Hibernate
- dom4j: dependencia hibernate
- commons-logging: dependencia hibernate
- antlr: dependencia hibernate

Jaxb: Librerías necesarias para el funcionamiento de Jaxb

- Plugin: Plugin del cual hablaremos mas adelante.

Nota: Es necesario entender las librerías y el significado de su inclusion por que generan sobrecarga al empaquetado final y en muchos casos ni se utilizan en el desarrollo.

Definiendo el contrato del servicio XSD

Teniendo en cuenta que estamos construyendo un servicio para recuperar simples datos bancarios vamos a empezar por la definición de nuestra entidad de negocio, una cuenta. Vamos a definir nuestra entidad cuenta

Un detalle que no debemos perder de vista que son los atributos del esquema targetNamespace, xmlns porque son las URL que utilizara JAXB para la generación de código.

src/main/webapp/schemas/AccountDetails.xsd: Definiendo la estructura de datos que utilizaremos en el retorno del servicio.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3  xmlns="http://com/company/webservices/account"
4  targetNamespace="http://com/company/webservices/account"
5  elementFormDefault="qualified" attributeFormDefault="unqualified">
6    <xs:element name="Account" type="Account"/>
7    <xs:complexType name="Account">
8      <xs:sequence>
9        <xs:element name="AccountNumber" type="xs:string"/>
10       <xs:element name="AccountName" type="xs:string"/>
11       <xs:element name="AccountBalance" type="xs:double"/>
12       <xs:element name="AccountStatus" type="EnumAccountStatus"/>
13     </xs:sequence>
14   </xs:complexType>
15   <xs:simpleType name="EnumAccountStatus">
16     <xs:restriction base="xs:string">
17       <xs:enumeration value="Active"/>
18       <xs:enumeration value="Inactive"/>
19     </xs:restriction>
20   </xs:simpleType>
21 </xs:schema>

```

En primera instancia definí un elemento Account que referencia la estructura de datos complejos para los elementos:

- AccountNumber

- AccountName
- AccountBalance
- AccountStatus

Nota: Realice una restricción en cuanto a los valores que puede aceptar proporcionando una lista de opciones. Active / Inactive. Esto se realiza con el objetivo de entender las restricciones en la definición de un archivo XSL.

src/main/webapp/schemas/AccountDetailsServiceOperations.xsd: Ahora vamos a definir nuestro Input/Output. Es decir la estructura con la que realizaremos la solicitud y la que utilizaremos para la respuesta.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3  xmlns="http://com/company/webservices/accountservice"
4  xmlns:account="http://com/company/webservices/account"
5  targetNamespace="http://com/company/webservices/accountservice"
6  elementFormDefault="qualified">
7    <xsd:import namespace="http://com/company/webservices/account"
8    schemaLocation="AccountDetails.xsd"/>
9    <xsd:element name="AccountDetailsRequest">
10      <xsd:complexType>
11        <xsd:sequence>
12          <xsd:element name="accountNumber" type="xsd:string"/>
13        </xsd:sequence>
14      </xsd:complexType>
15    </xsd:element>
16    <xsd:element name="AccountDetailsResponse">
17      <xsd:complexType>
18        <xsd:sequence>
19          <xsd:element name="AccountDetails" type="account:Account"/>
20        </xsd:sequence>
21      </xsd:complexType>
22    </xsd:element>
23  </xsd:schema>

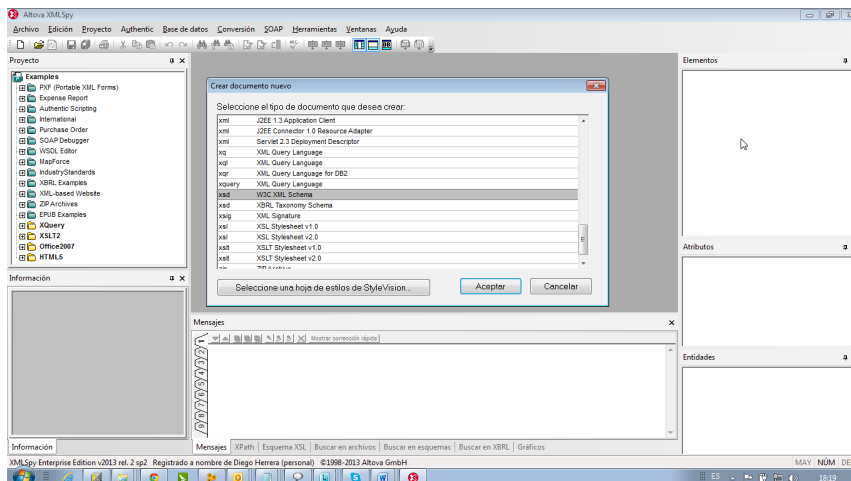
```

En primera instancia defino la estructura para para el Input o request al cual llamaremos **AccountDetailsRequest** y posterior defino los elementos que en este caso solo sera uno llamado AccountNumber de tipo string. Tenemos que recordar que vamos a buscar informacion por medio de esta entrada de datos.

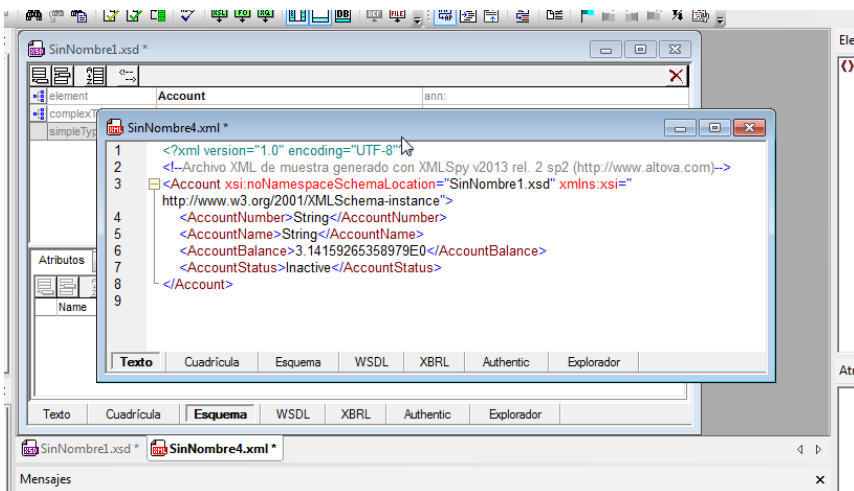
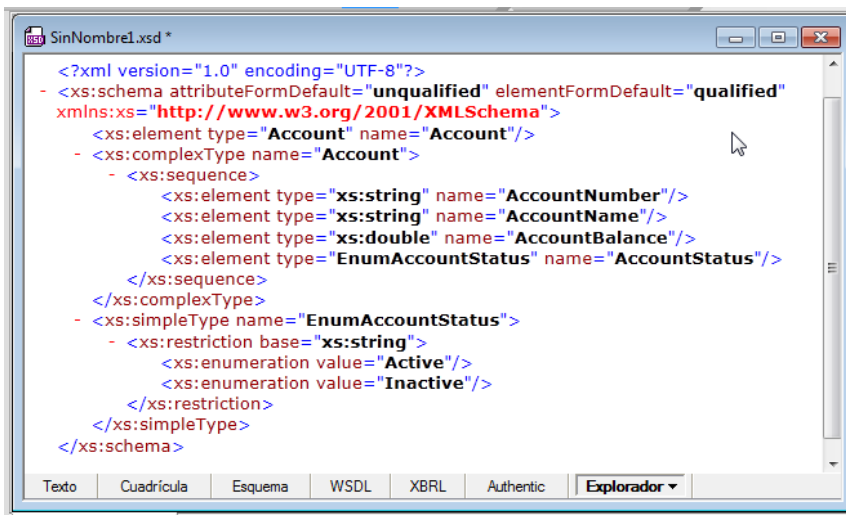
Lo siguiente es definir la estructura para mi output o response al cual lo llamaremos **AccountDetailsResponse** el cual contiene un elemento de tipo Account. Account ?? sisi retorna una estructura como la que definimos en el anterior XSD. Motivo por el cual si notamos podremos observar el import del XSD antes analizado.

Como un plus una herramienta que ayuda es XMLSPY

Es una herramienta como tantas otras para edicion de archivos xml, xsl, xsd entre otros formatos, con la particularidad que muestra una representación grafica de los elementos.



Entre algunas de sus funciones te proporciona la posibilidad de visualizar el archivo en navegador.



Nota Final: Es sin duda una potente herramienta que entre otras cosas proporciona un entorno donde se puede cargar datos de motores de base de datos, generar datos de pruebas....etc. Sin duda es una de las mejores herramientas para tratamiento de este tipo de archivos.

Conociendo un amigo JAXB

Una parte fundamental de los servicios web es la conversión de mensajes SOAP de XML a objetos Java y viceversa. Para facilitar esto vamos a utilizar el marco de trabajo JAXB para que lo realice por nosotros. JAXB es un marco poderoso y flexible, y cuenta con grandes mejoras a otros marcos como Castor.

Java Architecture for XML Binding (JAXB) específicamente con las interfaces Marshaller y Unmarshaller es responsables de dirigir el proceso de serialización de los contenidos de un objeto de alguna clase de Java a datos XML y viceversa. En palabras simples, JAXB nos permite almacenar y cargar información de nuestro modelo de objetos hacia y desde un archivo XML.

Para esta función de seguro podremos encontrar numerosas librerías con el objetivo de facilitarnos la vida a la hora de esta función, en lo personal realice algunas cosas con JAXB y me parece simple y fácil de aprender.

Como funciona ?

Para poder utilizar nuestros tipos definidos XSD en la aplicación, necesitamos generar clases Java de esos tipos. Es decir necesito crear una estructura de clases que realice el mapeo con los archivos XSD.

Como lo implemento en Proyecto?

Para esto vamos a utilizar una herramienta disponible de JAXB que es un plugin para Maven el que deberemos referencia en POM, nuestro plugin se llama **jaxb-maven-plugin**. El plugin está configurado para analizar un conjunto de XSD del generador y ejecutar la clase de JAXB para crear clases Java para cada uno de los tipos definidos.

Como Gestionar JAXB desde Maven

Hay un tema en la declaración de plugin en nuestro archivo POM cuando lo gestionamos desde M2ECLIPSE incluir el siguiente código en la sección build.

```
1 <build>
2   <plugins>
3     <plugin>
```

```

4      <groupId>org.codehaus.mojo</groupId>
5      <artifactId>jaxb2-maven-plugin</artifactId>
6      <version>1.4</version>
7      <executions>
8        <execution>
9          <goals>
10           <goal>xjc</goal>
11         </goals>
12         <phase>generate-sources</phase>
13       </execution>
14     </executions>
15     <configuration>
16       <clearOutputDir>>false</clearOutputDir>
17       <outputDirectory>src/main/java</outputDirectory>
18       <schemaDirectory>src/main/webapp/schemas</schemaDirectory>
19       <includeSchema>/**/*.xsd</includeSchema>
20       <bindingDirectory>src/main/resources/bindings</bindingDirectory>
21       <enableIntrospection>>false</enableIntrospection>
22     </configuration>
23   </plugin>
24   <plugin>
25     <groupId>org.apache.maven.plugins</groupId>
26     <artifactId>maven-war-plugin</artifactId>
27     <configuration>
28       <warName>${context.path}</warName>
29     </configuration>
30   </plugin>
31 </plugins>
32 </build>

```

Ahora analicemos un poco el código para entender como configurarlo, lo primero que debemos entender que JAXB utiliza una herramienta de compilador de esquemas xjc para las transformaciones que realiza.

Ahora la siguiente sección que definimos es claramente la configuración

- **clearOutputDir**: indica si hay que limpiar el directorio indicado en cada ejecución.
- **outputDirectory**: indica el directorio de generación.
- **schemaDirectory**: indica el directorio donde se encuentran los esquemas.
- **includeSchema**: indico un patron de busqueda en este caso le digo que tome todo los XSD.
- **src/main/resources**: indico el directorio de generación de archivos XSD.
- **enableIntrospection**: indico la posibilidad de generación Boolean getters/setters.

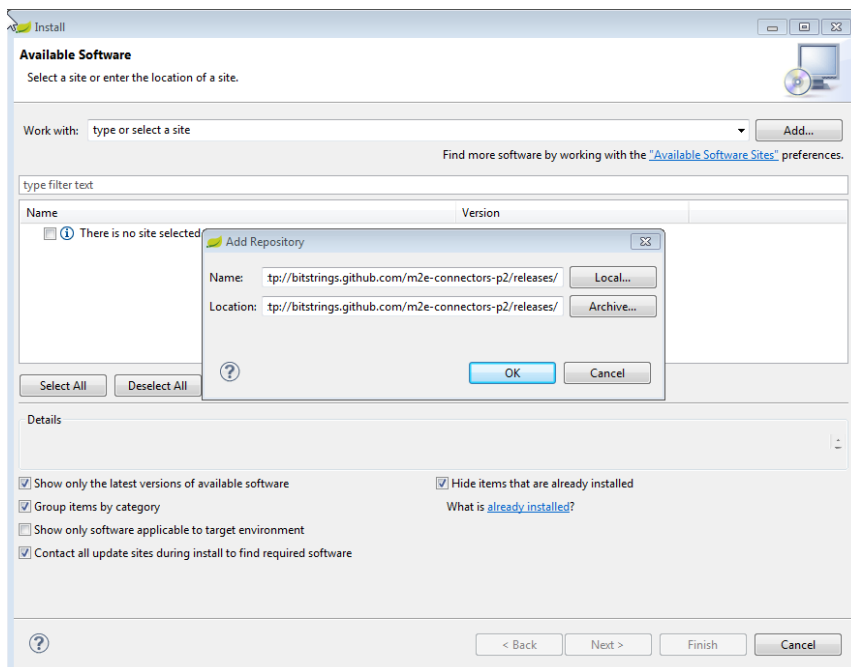
Error: Si agregas el plugin y el POM te dice flaco que esta pasando y te arroja lindos mensajes como:



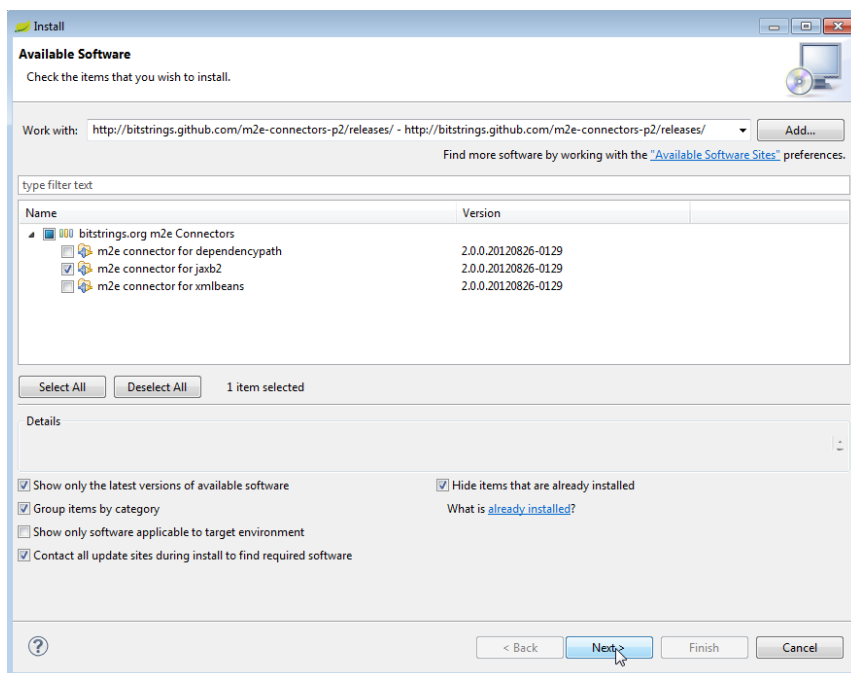
Que pasa que no le gusta ? y al parecer algunas herramientas para trabajar con Maven como M2ECLIPSE no tienen una buena interpretación de esta configuración en el POM por lo que requieren de un complemento de conexión M2eclipse para JAXB.

Me compro un buen libro de JAXB y hago a mano mis clases ? naaaa tranqui que hay una solución.

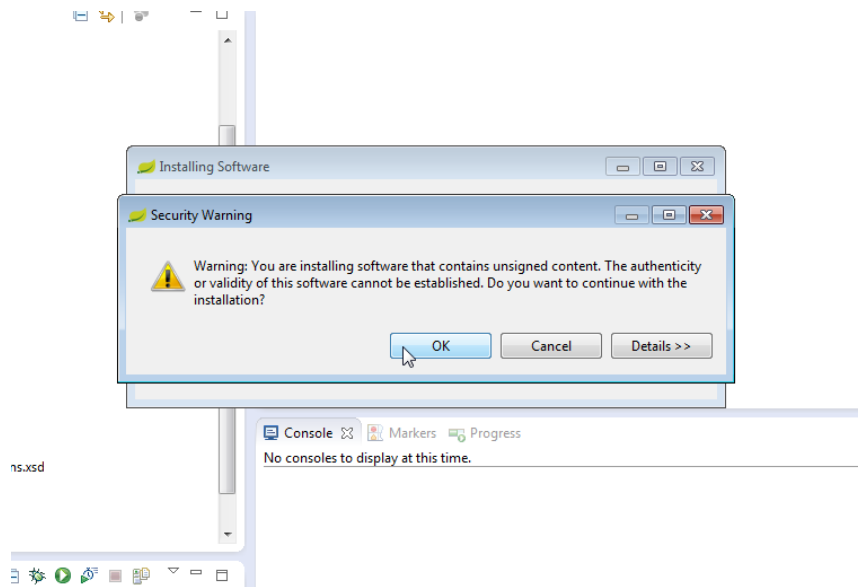
Primero vaya a "Install New Software" en nuestro eclipse y agregar la siguiente URL: <http://bitstrings.github.com/m2e-connectors-p2/releases/>



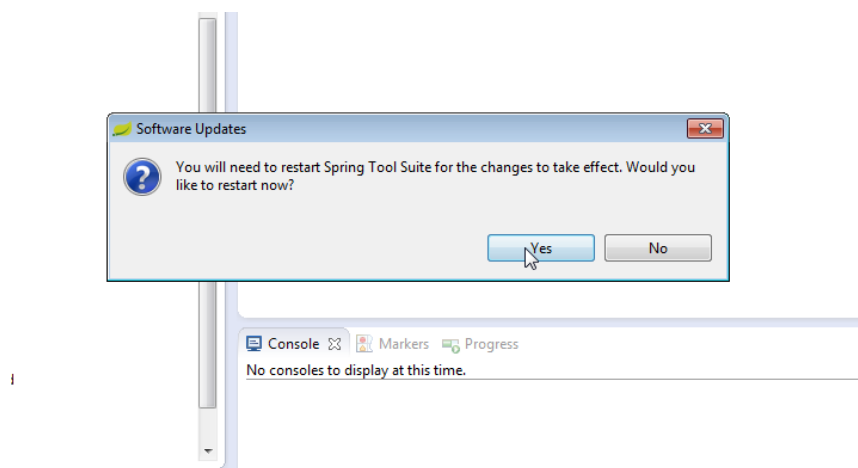
Seleccione el conector para JAXB2



En el siguiente paso nos presenta una validación del proceso de instalación a la cual le damos aceptar.

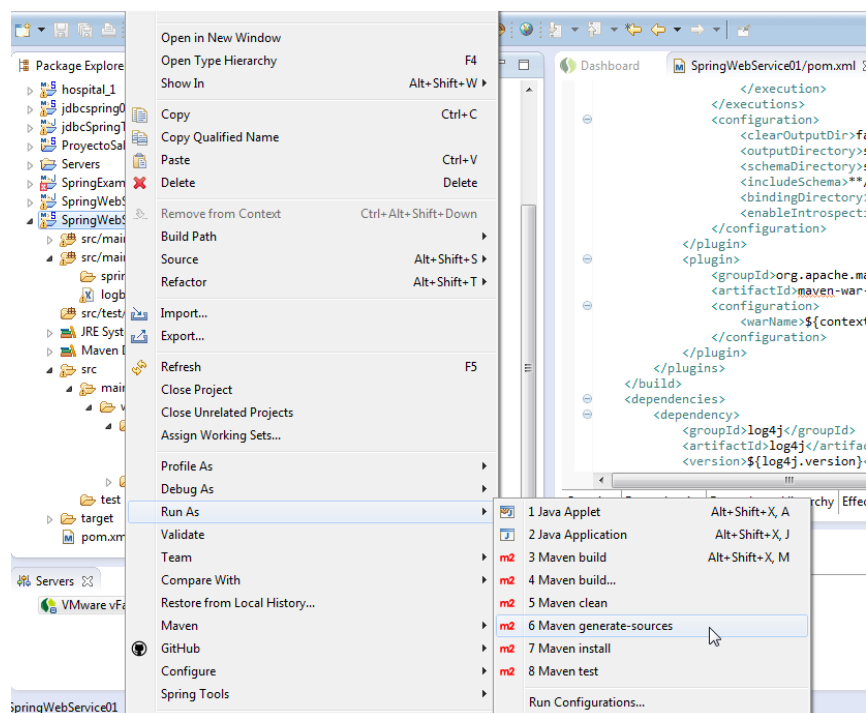


Finalmente nos indicara que debe reiniciar a lo que accederemos.



Nota: con todo esto ya tendríamos nuestro plugin listo para generar las clases de intercambio con los XSD, por lo que nuestro pom dejaría de gritar.

Ahora con todo esto debería pararme sobre el proyecto, boton derecho -> Run As ->Maven Generate Source.



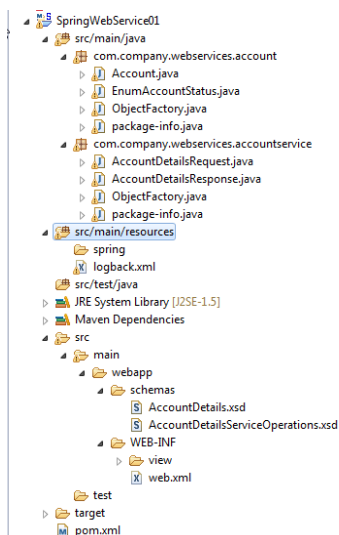
Si todo marcha bien por consola debería tener un resultado como el siguiente.

```

Console  Markers  Progress
<terminated> C:\Program Files\Java\jre7\bin\javaw.exe (19/10/2013 02:17:29)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for org.springframework:
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-war-plugin is missing.
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of you
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed pr
[WARNING]
[INFO] -----
[INFO] Building SpringWebService01 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- jaxb2-maven-plugin:1.4:xjc (default) @ SpringWebService01 ---
[INFO] No changes detected in schema or binding files, skipping source generation.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.160s
[INFO] Finished at: Sat Oct 19 02:17:32 ART 2013
[INFO] Final Memory: 4M/15M
[INFO] -----

```

Si todo marcha bien y Maven no se enoja debería haber creado las clases necesarias para mi proyecto en correspondencia con los archivos XSD.



Que bueno me facilitó la vida la generación de código y ahora tengo las clases necesarias para poder trabajar.

Ahora como sabe donde generar las clases el JAXB ?

Bueno recuerdan que hablamos de los esquemas en los archivos XSD!. Bueno los 2 elementos que utiliza el JAXB para direccionar la generación son

- **xmlns:** indica el default namespace
- **targetNamespace:** Indica que los elementos definidos por este esquema viene del namespace

Es lo logico por que son propiedades utilizadas por los archivos XSD para conectarse.

Nota Final: en pocas palabras nos genera las clases necesarias para poder trabajar, pero analicemos un poco la generación muy en poco detalle dado que no es el objetivo del tutorial entender las anotaciones en clase que utiliza JAXB

para vincular al los XSD.

com.company.webservice.account/Account.java: Genero una clases con getter y setter para la estructura Account que definimos en nuestro XSD.

```

1 //
2 // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Imple
3 // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
4 // Any modifications to this file will be lost upon recompilation of the source schema.
5 // Generated on: 2013.10.22 at 01:22:14 AM ART
6 //
7
8
9 package com.company.webservices.account;
10
11 import javax.xml.bind.annotation.XmlAccessType;
12 import javax.xml.bind.annotation.XmlAccessorType;
13 import javax.xml.bind.annotation.XmlElement;
14 import javax.xml.bind.annotation.XmlType;
15
16
17 /**
18  * <p>Java class for Account complex type.
19  *
20  * <p>The following schema fragment specifies the expected content contained within this cl
21  *
22  * <pre>
23  * <complexType name="Account">
24  *   <complexContent>
25  *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
26  *       <sequence>

```

```

27 *      <!--element name="AccountNumber" type="{http://www.w3.org/2001/XMLSchema}string"
28 *      <!--element name="AccountName" type="{http://www.w3.org/2001/XMLSchema}string"/>
29 *      <!--element name="AccountBalance" type="{http://www.w3.org/2001/XMLSchema}double
30 *      <!--element name="AccountStatus" type="{http://com/company/webservices/account}E
31 *      <!--sequence>
32 *      <!--restriction>
33 *      <!--complexContent>
34 *      <!--complexType>
35 *      </pre>
36 *
37 *
38 */
39 @XmlAccessorType(XmlAccessType.FIELD)
40 @XmlType(name = "Account", propOrder = {
41     "accountNumber",
42     "accountName",
43     "accountBalance",
44     "accountStatus"
45 })
46 public class Account {
47
48     @XmlElement(name = "AccountNumber", required = true)
49     protected String accountNumber;
50     @XmlElement(name = "AccountName", required = true)
51     protected String accountName;
52     @XmlElement(name = "AccountBalance")
53     protected double accountBalance;
54     @XmlElement(name = "AccountStatus", required = true)
55     protected EnumAccountStatus accountStatus;
56
57     /**
58      * Gets the value of the accountNumber property.
59      *
60      * @return
61      *     possible object is
62      *     {@link String }
63      */
64     public String getAccountNumber() {
65         return accountNumber;
66     }
67
68     /**
69      * Sets the value of the accountNumber property.
70      *
71      * @param value
72      *     allowed object is
73      *     {@link String }
74      */
75     public void setAccountNumber(String value) {
76         this.accountNumber = value;
77     }
78
79     /**
80      * Gets the value of the accountName property.
81      *
82      * @return
83      *     possible object is
84      *     {@link String }
85      */
86     public String getAccountName() {
87         return accountName;
88     }
89
90     /**
91      * Sets the value of the accountName property.
92      *
93      * @param value
94      *     allowed object is
95      *     {@link String }
96      */
97     public void setAccountName(String value) {
98         this.accountName = value;
99     }
100
101     /**
102      * Gets the value of the accountBalance property.
103      */
104     public double getAccountBalance() {
105         return accountBalance;
106     }
107
108     /**
109      * Sets the value of the accountBalance property.
110      */
111     public void setAccountBalance(double value) {
112         this.accountBalance = value;
113     }
114
115     /**
116      * Gets the value of the accountStatus property.
117      *
118      * @return
119      *     possible object is
120      *     {@link EnumAccountStatus }
121      */
122     public EnumAccountStatus getAccountStatus() {
123         return accountStatus;
124     }
125
126     /**
127      * Sets the value of the accountStatus property.
128      *
129      * @param value
130      *     allowed object is
131      *     {@link EnumAccountStatus }
132      */
133

```

```

139     *
140     */
141     public void setAccountStatus(EnumAccountStatus value) {
142         this.accountStatus = value;
143     }
144 }
145 }

```

com.company.webservice.account/EnumAccountStatus.java: si recuerdan generamos una restricción en el XSD y lo logico es que jaxb genere una estructura de enumeracion para este tipo de datos.

```

1  //
2  // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implem
3  // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
4  // Any modifications to this file will be lost upon recompilation of the source schema.
5  // Generated on: 2013.10.22 at 01:22:14 AM ART
6  //
7
8
9  package com.company.webservices.account;
10
11  import javax.xml.bind.annotation.XmlEnum;
12  import javax.xml.bind.annotation.XmlEnumValue;
13  import javax.xml.bind.annotation.XmlType;
14
15  /**
16   * <p>Java class for EnumAccountStatus.
17   *
18   * <p>The following schema fragment specifies the expected content contained within this cla
19   * <p>
20   * <pre>
21   * <simpleType name="EnumAccountStatus">
22   *   <restriction base="{http://www.w3.org/2001/XMLSchema}string">
23   *     <enumeration value="Active"/>
24   *     <enumeration value="Inactive"/>
25   *   </restriction>
26   * </simpleType>
27   * </pre>
28   */
29
30  @XmlType(name = "EnumAccountStatus")
31  @XmlEnum
32  public enum EnumAccountStatus {
33
34      @XmlEnumValue("Active")
35      ACTIVE("Active"),
36      @XmlEnumValue("Inactive")
37      INACTIVE("Inactive");
38      private final String value;
39
40      EnumAccountStatus(String v) {
41          value = v;
42      }
43
44      public String value() {
45          return value;
46      }
47
48      public static EnumAccountStatus fromValue(String v) {
49          for (EnumAccountStatus c: EnumAccountStatus.values()) {
50              if (c.value.equals(v)) {
51                  return c;
52              }
53          }
54          throw new IllegalArgumentException(v);
55      }
56  }
57
58 }

```

com.company.webservice.account/ObjectFactory.java: esta es nuestra fabrica de objetos Account por codigo.

```

1  //
2  // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implem
3  // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
4  // Any modifications to this file will be lost upon recompilation of the source schema.
5  // Generated on: 2013.10.22 at 01:22:14 AM ART
6  //
7
8
9  package com.company.webservices.account;
10
11  import javax.xml.bind.JAXBElement;
12  import javax.xml.bind.annotation.XmlElementDecl;
13  import javax.xml.bind.annotation.XmlRegistry;
14  import javax.xml.namespace.QName;
15
16  /**
17   * This object contains factory methods for each
18   * Java content interface and Java element interface
19   * generated in the com.company.webservices.account package.
20   *
21   * <p>An ObjectFactory allows you to programatically
22   * construct new instances of the Java representation
23   * for XML content. The Java representation of XML
24   * content can consist of schema derived interfaces
25   * and classes representing the binding of schema
26   * type definitions, element declarations and model
27   * groups. Factory methods for each of these are
28   * provided in this class.
29

```

```

30 */
31 @XmlRegistry
32 public class ObjectFactory {
33
34     private final static QName _Account_QNAME = new QName("http://com/company/webservices/ac
35
36     /**
37      * Create a new ObjectFactory that can be used to create new instances of schema derived
38      *
39      */
40     public ObjectFactory() {
41     }
42
43     /**
44      * Create an instance of {@link Account }
45      *
46      */
47     public Account createAccount() {
48         return new Account();
49     }
50
51     /**
52      * Create an instance of {@link JAXBElement }{@code <}{@link Account }{@code >}}
53      *
54      */
55     @XmlElementDecl(namespace = "http://com/company/webservices/account", name = "Account")
56     public JAXBElement<Account> createAccount(Account value) {
57         return new JAXBElement<Account>(_Account_QNAME, Account.class, null, value);
58     }
59
60 }

```

com.company.webservices.accountservice/AccountDetailsRequest.java: si miramos podremos observar que es una clase con un propiedad accountnumber y con sus correspondientes getter y setter, ahora si recordamos el XSD es como definimos nuestra estructura request.

```

1 //
2 // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implem
3 // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
4 // Any modifications to this file will be lost upon recompilation of the source schema.
5 // Generated on: 2013.10.22 at 01:22:14 AM ART
6 //
7
8
9 package com.company.webservices.accountservice;
10
11 import javax.xml.bind.annotation.XmlAccessType;
12 import javax.xml.bind.annotation.XmlAccessorType;
13 import javax.xml.bind.annotation.XmlElement;
14 import javax.xml.bind.annotation.XmlRootElement;
15 import javax.xml.bind.annotation.XmlType;
16
17 /**
18  * <p>Java class for anonymous complex type.
19  *
20  * <p>The following schema fragment specifies the expected content contained within this cla
21  *
22  * <pre>
23  * <complexType>
24  *   <complexContent>
25  *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
26  *       <sequence>
27  *         <element name="accountNumber" type="{http://www.w3.org/2001/XMLSchema}string"/
28  *       </sequence>
29  *     </restriction>
30  *   </complexContent>
31  * </complexType>
32  * </pre>
33  *
34  *
35  */
36
37 @XmlAccessorType(XmlAccessType.FIELD)
38 @XmlType(name = "", propOrder = {
39     "accountNumber"
40 })
41 @XmlRootElement(name = "AccountDetailsRequest")
42 public class AccountDetailsRequest {
43
44     @XmlElement(required = true)
45     protected String accountNumber;
46
47     /**
48      * Gets the value of the accountNumber property.
49      *
50      * @return
51      *     possible object is
52      *     {@link String }
53      *
54      */
55     public String getAccountNumber() {
56         return accountNumber;
57     }
58
59     /**
60      * Sets the value of the accountNumber property.
61      *
62      * @param value
63      *     allowed object is
64      *     {@link String }
65      *
66      */
67     public void setAccountNumber(String value) {
68         this.accountNumber = value;
69     }
70

```


71 | }

com.company.webservices.accountservice/AccountDetailsResponse.java: si miramos podremos observar que es una clase con un objeto Account como propiedad y con sus correspondientes getter y setter, ahora si recordamos el XSD es como definimos nuestra estructura response.

```

1  //
2  // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implem
3  // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
4  // Any modifications to this file will be lost upon recompilation of the source schema.
5  // Generated on: 2013.10.22 at 01:22:14 AM ART
6  //
7
8
9  package com.company.webservices.accountservice;
10
11  import javax.xml.bind.annotation.XmlAccessType;
12  import javax.xml.bind.annotation.XmlAccessorType;
13  import javax.xml.bind.annotation.XmlElement;
14  import javax.xml.bind.annotation.XmlRootElement;
15  import javax.xml.bind.annotation.XmlType;
16  import com.company.webservices.account.Account;
17
18
19  /**
20   * <p>Java class for anonymous complex type.
21   *
22   * <p>The following schema fragment specifies the expected content contained within this cla
23   *
24   * <pre>
25   * <complexType>
26   *   <complexContent>
27   *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
28   *       <sequence>
29   *         <element name="AccountDetails" type="{http://com/company/webservices/account}A
30   *       </sequence>
31   *     </restriction>
32   *   </complexContent>
33   * </complexType>
34   * </pre>
35   *
36   */
37
38  @XmlAccessorType(XmlAccessType.FIELD)
39  @XmlType(name = "", propOrder = {
40      "accountDetails"
41  })
42  @XmlRootElement(name = "AccountDetailsResponse")
43  public class AccountDetailsResponse {
44
45      @XmlElement(name = "AccountDetails", required = true)
46      protected Account accountDetails;
47
48      /**
49       * Gets the value of the accountDetails property.
50       *
51       * @return
52       *     possible object is
53       *     {@link Account }
54       *
55       */
56      public Account getAccountDetails() {
57          return accountDetails;
58      }
59
60      /**
61       * Sets the value of the accountDetails property.
62       *
63       * @param value
64       *     allowed object is
65       *     {@link Account }
66       *
67       */
68      public void setAccountDetails(Account value) {
69          this.accountDetails = value;
70      }
71  }
72

```

com.company.webservices.accountservice/ObjectFactory.java: nuestra fabrica de objetos para los objetos antes mencionados.

```

1  //
2  // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implem
3  // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
4  // Any modifications to this file will be lost upon recompilation of the source schema.
5  // Generated on: 2013.10.22 at 01:22:14 AM ART
6  //
7
8
9  package com.company.webservices.accountservice;
10
11  import javax.xml.bind.annotation.XmlRegistry;
12
13
14  /**
15   * This object contains factory methods for each
16   * Java content interface and Java element interface
17   * generated in the com.company.webservices.accountservice package.
18   * <p>An ObjectFactory allows you to programatically
19   * construct new instances of the Java representation

```

```

20 * for XML content. The Java representation of XML
21 * content can consist of schema derived interfaces
22 * and classes representing the binding of schema
23 * type definitions, element declarations and model
24 * groups. Factory methods for each of these are
25 * provided in this class.
26 */
27 */
28 @XmlRegistry
29 public class ObjectFactory {
30
31
32 /**
33 * Create a new ObjectFactory that can be used to create new instances of schema derived
34 *
35 */
36 public ObjectFactory() {
37 }
38
39 /**
40 * Create an instance of {@link AccountDetailsRequest }
41 *
42 */
43 public AccountDetailsRequest createAccountDetailsRequest() {
44     return new AccountDetailsRequest();
45 }
46
47 /**
48 * Create an instance of {@link AccountDetailsResponse }
49 *
50 */
51 public AccountDetailsResponse createAccountDetailsResponse() {
52     return new AccountDetailsResponse();
53 }
54 }
55 }

```

Mirando los códigos en pocas palabras genero clases con una correspondencia para los archivos XSD.

Generando nuestra clases DAO con hibernate.

Primero que nada utilizaremos Hibernate para la persistencia y por supuesto las mejoras que proporciona spring para manejo con ORM.

com.company.hbm/Account.hbm.xml: donde encontraremos los archivos de mapeo Hibernate.

```

1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4 <!-- Generated 13-sep-2013 15:59:20 by Hibernate Tools 3.4.0.CR1 -->
5 <hibernate-mapping>
6     <class name="com.company.modelo.AccountEntity" table="account" catalog="springbanco">
7         <id name="id" type="java.lang.Integer">
8             <column name="Id" />
9             <generator class="identity" />
10        </id>
11        <property name="Numero" type="string">
12            <column name="Numero" length="50" />
13        </property>
14        <property name="Nombre" type="string">
15            <column name="Nombre" length="100" />
16        </property>
17        <property name="Balance" type="string">
18            <column name="Balance" length="100" />
19        </property>
20        <property name="Estado" type="string">
21            <column name="Estado" length="50" />
22        </property>
23    </class>
24 </hibernate-mapping>

```

Nota: Es nuestro archivo de mapeo que hace referencia a la estructura de la tabla y base de datos.

com.company.modelo/AccountEntity.java: el pojo con el que se hace el mapping.

```

1 package com.company.modelo;
2
3 public class AccountEntity {
4     private Integer id;
5     private String Numero;
6     private String Nombre;
7     private String Balance;
8     private String Estado;
9
10    public Integer getId() {
11        return id;
12    }
13    public void setId(Integer id) {
14        this.id = id;
15    }
16    public String getNumero() {
17        return Numero;
18    }
19    public void setNumero(String numero) {
20        Numero = numero;
21    }
22 }

```

```

22 public String getNombre() {
23     return Nombre;
24 }
25 public void setNombre(String nombre) {
26     Nombre = nombre;
27 }
28 public String getBalance() {
29     return Balance;
30 }
31 public void setBalance(String balance) {
32     Balance = balance;
33 }
34 public String getEstado() {
35     return Estado;
36 }
37 public void setEstado(String estado) {
38     Estado = estado;
39 }
40
41
42
43 }

```

Nota: Aquí es cuando el tipo dice pero para creo que definí ya en el proyecto una clase entidad que tiene casi la misma estructura !!!! si si es verdad pero solo es una casualidad por que las clases que construimos con JAXB de XSL a Objetos las definimos como respuesta a la necesidad del input del servicio que no siempre se va corresponder directamente a una tabla del modelo de datos, en este caso si por que es con fines educaciones.

Importante: la configuración para hibernate la abordaremos al momento de analizar los spring bean configuracion.

com.company.webservice.dao/AccountDao.java: es la interfaz o contrato de los métodos que tendrá la implementación hibernate,

```

1 package com.company.webservice.dao;
2
3 import com.company.modelo.AccountEntity;
4 import com.company.webservices.account.Account;
5
6 public interface AccountDao {
7     public AccountEntity getAccountDetails(String accountNumber);
8 }

```

com.company.webservice.dao/AccountDaoImpl.java: Corresponde a la implementación hibernate.

```

1 package com.company.webservice.dao;
2
3 import org.apache.log4j.Logger;
4 import org.hibernate.SessionFactory;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
7 import org.springframework.stereotype.Repository;
8
9 import com.company.modelo.AccountEntity;
10 import com.company.webservices.account.Account;
11 import com.company.webservices.account.EnumAccountStatus;
12
13 @Repository
14 public class AccountDaoImpl extends HibernateDaoSupport implements AccountDao {
15
16     @Autowired
17     public AccountDaoImpl(SessionFactory sessionFactory) {
18         super.setSessionFactory(sessionFactory);
19     }
20
21     public AccountEntity getAccountDetails(String accountNumber) {
22
23         return (AccountEntity) getHibernateTemplate().get(AccountEntity.class, new Integer(accountNumber));
24     }
25 }
26
27 }

```

Por un lado extenderemos de `HibernateDaoSupport` a fin de optimizar el trabajo Hibernate, por otro lado podemos notar que inyectara por setter la `sessionFactory` necesaria para poder trabajar. Cuando analicemos el archivo de configuración spring podremos tener un mayor entendimiento pero quiero comentar que la `sessionfactory` tiene la información de datos para la conectividad al motor y la información necesaria para el mapeo hibernate.

Algunos conceptos de Notaciones con Spring.

@Component es el estereotipo principal, indica que la clase anotada es un component (o un Bean de Spring).

@Repository, **@Service** y **@Controller** son especializaciones de **@Component** para casos concretos (persistencia, servicios y presentación).

Nota: Esto significa que puede usarse siempre **@Component** pero lo adecuado es usar estos estereotipos ya que algunas herramientas o futuras versiones de Spring pueden añadir semántica adicional (por ejemplo Spring Roo usa estas anotaciones y genera aspectos).

En nuestra clase de implementación dao marcamos a la clase como **@Repository**. Esta es una anotación de Spring. Estamos indicando que esta es una clase relacionada con la capa de persistencia, y que debe ser un **Singleton** (sólo habrá una instancia de la clase **HibernateDaoSupport**, y todos los Threads de la aplicación la compartirán).

@Autowired. Esta es una anotación de Spring. Sirve para indicarle a Spring que cuando vaya a crear la instancia de

HibernateDaoSupport debe "inyectarle" (pasarle) en el constructor una referencia al SessionFactory (el SessionFactory sí lo configuraremos mediante XML, lo veremos más adelante).

En cuanto a código no hay grandes complicaciones, utilizamos los métodos de getHibernateTemplate para obtener un objeto AccountEntity de acuerdo a un accountnumber ingresado por parametro.

Generando nuestra clases SERVICE FACADE.

Ahora vamos a generar la interfaz y la implementacion para el servicio en primera instancia vamos a crear nuestra interfaz.

com.company.webservice.service/AccountService.java: Generaremos solo un método que reciba un numero de cuenta y retorne un objeto Account.

```
1 package com.company.webservice.service;
2
3 import com.company.webservices.account.Account;
4
5 public interface AccountService {
6
7     public Account getAccountDetails(String accountNumber);
8
9 }
```

com.company.webservice.service/AccountServiceImpl.java: es la implementacion de nuestro servicio que recibe un numero de cuenta y retorna un account que es la clase que definimos como medio de transporte para los datos que retornamos por el servicio.

```
1 package com.company.webservice.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.stereotype.Component;
6 import org.springframework.stereotype.Service;
7
8 import com.company.modelo.AccountEntity;
9 import com.company.webservice.dao.AccountDao;
10 import com.company.webservice.dao.AccountDaoImpl;
11 import com.company.webservices.account.Account;
12 import com.company.webservices.account.EnumAccountStatus;
13
14 @Service
15 public class AccountServiceImpl implements AccountService {
16
17     @Autowired
18     private AccountDao accountdao;
19
20     public Account getAccountDetails(String accountNumber) {
21         System.out.println("paso por servicio");
22
23         AccountEntity obj = accountdao.getAccountDetails(accountNumber);
24
25         Account account = new Account();
26         account.setAccountNumber(obj.getNumero());
27         account.setAccountStatus(EnumAccountStatus.ACTIVE);
28         account.setAccountName(obj.getNombre());
29         account.setAccountBalance(Double.parseDouble(obj.getBalance()));
30         return account;
31     }
32
33 }
```

Ahora si analizamos el uso de estos patrones la capa de servicio es la que utilizara todos los dao que pueda tener definido el modelo de datos para responder a necesidades especificas indicadas para la exposicion de funcionalidades.

En nuestro caso utilizara un dao para enviar un numero de cuenta y obtener la cuenta de la base de datos mysql y luego cargara el objeto account para retornarlo para que se pueda preparar el retorno del servicio a la clase endpoint.

Nota: La notacion @Service sirve como una especialización de la notacion @component, teniendo en cuenta las clases de implementación para ser detectado automáticamente por medio de escaneo classpath. Lo que significa que puedo anotar mi clases de servicio con @Component, pero anotando con @Service las clases se adaptan más adecuadamente para su procesamiento por herramientas o asociarse con los aspectos, ya que @Service es ideal para la utilizacion de pointcut. Por lo que claramente que para la definicion de nuestra implementacion del servicio debemos utilizar la notacion @service.

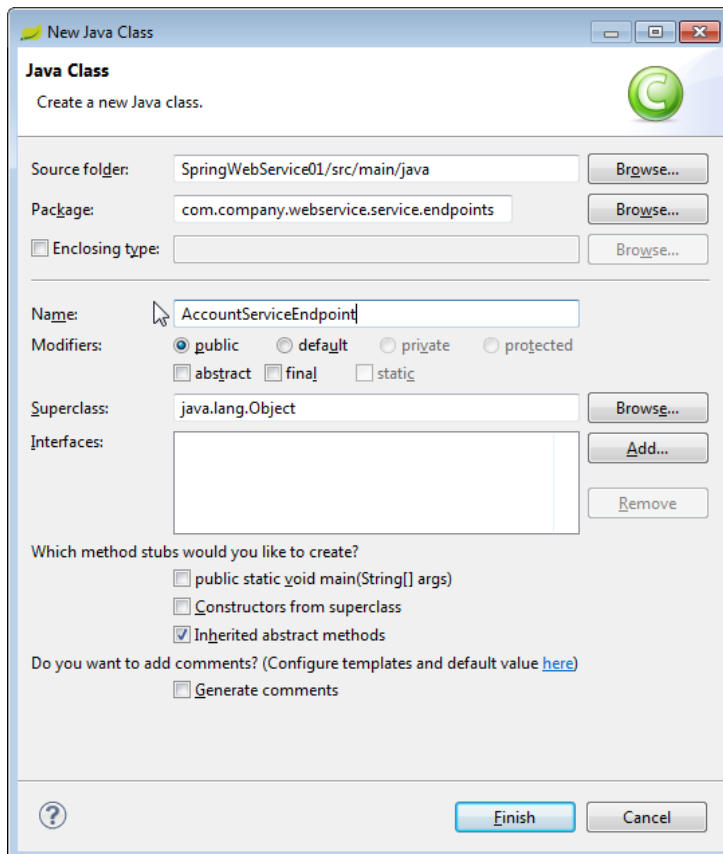
Por otro lado podemos notar la inyeccion de dependencia de la clase dao que al indicarle @Autowired buscara el bean de manera automatica para dejar la clase disponible.

Creando nuestro Service EndPoint

El endpoint es un punto de conexión donde archivos HTML o paginas active server son expuestas. Endpoint proporciona información necesaria para direccionar un Webservice. Proporciona una referencia o especificación que se utiliza para definir un grupo o familia de propiedades de direccionamiento de mensajes y dar características de los mensajes tales como la fuente y el destino del EndPoint. Posibilita el uniforme direccionamiento de los mensajes.

En la vida real es la implementacion de los metodos que un cliente puede llamar sobre un servicio web.

Nota: Una Service Endpoint Interface (SEI) declara los métodos que un cliente puede invocar sobre un web service, La interfaz no es estrictamente necesaria, ya que de no estar presente, la clase de implementación implícitamente define una interfaz. Si queremos definir una interfaz explícitamente, debemos agregar el elemento "endpointInterface" a la anotación WebService



com.company.webservice.service.endpoints/AccountServiceEndpoint.java: nuestra implementación de servicio endpoint.

```

1  package com.company.webservice.service.endpoints;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.ws.server.endpoint.annotation.Endpoint;
5  import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
6  import org.springframework.ws.server.endpoint.annotation.RequestPayload;
7  import org.springframework.ws.server.endpoint.annotation.ResponsePayload;
8
9  import com.company.webservice.service.AccountService;
10 import com.company.webservice.service.AccountServiceImpl;
11 import com.company.webservices.account.Account;
12 import com.company.webservices.accountservice.AccountDetailsRequest;
13 import com.company.webservices.accountservice.AccountDetailsResponse;
14
15 @Endpoint
16 public class AccountServiceEndpoint {
17
18     private static final String TARGET_NAMESPACE = "http://com/company/webservices/accountserv
19
20     @Autowired
21     private AccountService accountservice;
22
23     @PayloadRoot(localPart = "AccountDetailsRequest", namespace = TARGET_NAMESPACE)
24     public @ResponsePayload AccountDetailsResponse getAccountDetails(@RequestPayload Ac
25     {
26
27         System.out.println("Entro al metodo");
28         AccountDetailsResponse response = new AccountDetailsResponse();
29
30         System.out.println(request.getAccountNumber());
31         Account account = accountservice.getAccountDetails(request.getAccountNumber());
32         System.out.println("Encontro " + account.getAccountName());
33         response.setAccountDetails(account);
34         System.out.println("salgo del metodo");
35         return response;
36     }
37 }
38
39 }
```

Ahora el código antes mencionado hace uso del soporte de anotaciones de spring para Servicios web, vamos a explicar un poco el código.

@ Enpoint es una versión especializada de la notacion standar de spring @ Component y permite esta clase del servicio ser leida y registrada en los escaneos componente Springs.

Spring proporciona dos anotaciones para la inyección de dependencias `@Autowired` y `@Qualifier`.

`@Autowired` funciona por tipo, y es que ella sola se encarga de buscar un *bean* de la clase correspondiente definida.

`@Autowired`

```
private AccountService accountService_i;
```

La gran limitación de esta anotación es que no es posible hacer inyección por nombre (¿qué pasa si tenemos varios *beans* del mismo tipo?), por lo que la solución pasa por complementarla con `@Qualifier`. Que de esta forma buscaría el tipo y que tenga un nombre particular.

`@Autowired`

```
@Qualifier("minombreespecifico")
```

```
private AccountService accountService_i;
```

Perdón me fui del tema.!!!

Lo siguiente es definir el namespace definido en nuestro XSD, esto es usado para el Endpoint para mapear la solicitud con el metodo a procesar.

```
private static final String TARGET_NAMESPACE = "http://com/company/webservices/accountservice";
```

A continuación trabajamos sobre la solicitud o Request, `@PayloadRoot` indica que este método va a procesar solicitudes de servicio con el elemento XML coincidente el definido por el atributo LocalPart. En el ejemplo anterior, nuestro método procesará las solicitudes de entrada de tipo AccountDetailsRequest con espacio de nombres `http://com/company/webservices/accountservice`. Recuerda que hemos definido este tipo XSD y espacio de nombres antes.

```
@PayloadRoot(localPart = "AccountDetailsRequest", namespace = TARGET_NAMESPACE)
```

A continuación llego el momento de trabajar sobre la respuesta o Request `@ResponsePayload` indica el tipo de ser devuelto en la respuesta SOAP. En este ejemplo, el objeto AccountDetailsResponse se convierte a XML y devuelve a la aplicación cliente como una respuesta SOAP. `@RequestPayload AccountDetails` indica a spring convertir las solicitudes entrantes AccountDetails a partir de XML a clase Java.

```
public @ResponsePayload AccountDetailsResponse getAccountDetails(@RequestPayload
AccountDetailsRequest request)
```

En Tucumano!! la respuesta del metodo se convierte a XML y el input que llega en XML a clase para que lo pueda procesar por java.

Finalmente nuestro código del método realiza las siguientes operaciones.

Genero una instancia de la respuesta.

```
AccountDetailsResponse response = new AccountDetailsResponse();
```

Obtengo la cuenta utilizando el numero de cuenta a buscar de la solicitud, procesandolo con la implementacion de mi servicio.

```
Account account = accountService_i.getAccountDetails(request.getAccountNumber());
```

Finalmente cargo la cuenta en el objeto respuestas.

```
response.setAccountDetails(account);
```

Retorno la respuesta

```
return response;
```

Podemos notar claramente que realizaremos una inyección de la clase servicio para trabajar en todo el procesamiento del metodo implementado del servicio.

Configurando mi Spring Bean Configuration

Finalmente la parte interesante y que por supuesto cierra el circuito de funcionamiento nuestro Tutorial.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xmlns:context="http://www.springframework.org/schema/context"
5  xmlns:sws="http://www.springframework.org/schema/web-services"
6  xmlns:jdbc="http://www.springframework.org/schema/jdbc"
7  xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/
8  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/sp
9  http://www.springframework.org/schema/web-services http://www.springframework.org/schema/w
10 http://www.springframework.org/schema/context http://www.springframework.org/schema/contex
11
12
13
14      <sws:annotation-driven />
15      <context:component-scan base-package="com.company.webservice" />
16
17      <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
18          <property name="driverClassName">
19              <value>com.mysql.jdbc.Driver</value>

```

```

20 </property>
21 <property name="url">
22 <value>jdbc:mysql://localhost:3306/springbanco</value>
23 </property>
24 <property name="username">
25 <value>root</value>
26 </property>
27 <property name="password">
28 <value></value>
29 </property>
30 </bean>
31
32 <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean
33 <property name="dataSource">
34 <ref bean="dataSource"/>
35 </property>
36 <property name="mappingResources">
37 <list>
38 <value>com/company/hbm/Account.hbm.xml</value>
39 </list>
40 </property>
41 <property name="hibernateProperties">
42 <props>
43 <prop key="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</prop>
44 <prop key="hibernate.show_sql">true</prop>
45 </props>
46 </property>
47 </bean>
48
49
50
51
52
53
54 <bean id="AccountDetailsService" class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Def
55 <property name="schemaCollection">
56 <bean class="org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection">
57 <property name="inline" value="true" />
58 <property name="xsds">
59 <list>
60 <value>schemas/AccountDetailsServiceOperations.xsd</value>
61 </list>
62 </property>
63 </bean>
64 </property>
65 <property name="portTypeName" value="AccountDetailsService"/>
66 <property name="serviceName" value="AccountDetailsService" />
67 <property name="locationUri" value="/endpoints"/>
68 </bean>
69
70 </beans>

```

El `<sws:annotation-driven />` busca todos los beans con la anotación `@EndPoint` y mapea el mapeo a todos anotados con `@PayloadRoot`.

¿Cómo le decimos a Spring que por notaciones genere los beans de manera dinamica para las clases marcadas como `@Repository`, `@Service`, si tuvieramos `@Component` definidos en nuestro proyecto ? Debemos hacer uso de la etiqueta `component-scan`, a la que podemos pasar como atributo uno o varios paquetes (separados por coma):

en nuestro ejemplo lo utilizamos

```
<context:component-scan base-package="com.company.webservice" />
```

Con esto haremos que Spring escanee el paquete `com.company.webservice` en busca de clases anotadas con las notaciones antes mencionadas. Este proceso se realiza al levantar el contexto y hay que tener en cuenta que **Spring escaneará tanto los sub-paquetes como las dependencias Jar**. En el peor de los casos el proceso tardará segundos, pero lo recomendable es evitar que la búsqueda sea demasiado general. Por lo que una posible solución es incluir los diferentes paquetes de manera directa separados por comas.

Nota: te aviso que si no colocas la sentencia para que realice el scaneo la generación de Bean dinámicas no se realizará por lo que anivel código en la ejecución tus clases no estarán instanciadas por lo que garantizo que rompe por todos lados.

El siguiente Bean es el datasource y que en nuestro caso tiene información de la base de datos.

```

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName">
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property name="url">
    <value>jdbc:mysql://localhost:3306/springbanco</value>
  </property>
  <property name="username">
    <value>root</value>
  </property>
  <property name="password">
    <value></value>
  </property>
</bean>

```

El siguiente es el sessionFactory que se relaciona con el datasource, contiene el mapeo de clases y tablas y la definición

de propiedades de Hibernate, este es muy importante dado que es quien se inyecta en la clase dao de implementacion dao que extiende de hibernatedaosupport.

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource">
    <ref bean="dataSource"/>
  </property>
  <property name="mappingResources">
    <list>
      <value>com/company/hbm/Account.hbm.xml</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
```

Nuestro ultimo Beans es AccountDetailsService que es el beans para el spring ws.

Uso de DefaultWsd11Definition permite la generación automática de WSDL. Spring utiliza las definiciones de esquemas listados en la propiedad schemaCollection, así como el portType, serviceName y locationUri para generar un archivo WSDL la primera vez que se solicita.

```
<bean id="AccountDetailsService" class="org.springframework.ws.wsd11.DefaultWsd11Definition" lazy-
init="true">
  <property name="schemaCollection">
    <bean class="org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection">
      <property name="inline" value="true" />
      <property name="xsds">
        <list>
          <value>schemas/AccountDetailsServiceOperations.xsd</value>
        </list>
      </property>
    </bean>
  </property>
  <property name="portTypeName" value="AccountDetailsService"/>
  <property name="serviceName" value="AccountDetailsService" />
  <property name="locationUri" value="/endpoints"/>
</bean>
```

Nota: Aunque esta es una característica de gran alcance que se debe utilizar con precaución en la producción como el proceso de generación de WSDL puede ser muy lento.

Un enfoque que se utilizaba en el pasado es copiar el WSDL generado por el navegador para su proyecto y exponerlo de manera estática.

Configurando Web.xml

Si no configuramos esto no vamos para ningun lado

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xmlns="http://java.sun.com/xml/ns/javaee"
4          xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5          http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
6          id="WebApp_ID" version="2.5">
7
8      <display-name>SpringWebService01</display-name>
9
10     <!--
11     - Location of the XML file that defines the root application context.
12     - Applied by ContextLoaderListener.
13     -->
14     <context-param>
15       <param-name>contextConfigLocation</param-name>
16       <param-value>
17         /WEB-INF/config/spring.xml
18       </param-value>
19     </context-param>
20
21     <listener>
22       <listener-class>org.springframework.web.context.ContextLoaderListener</listener-clas
23     </listener>
24
25
26     <!--
27     - Servlet that dispatches request to registered handlers (Controller implementations).
28     -->
29     <servlet>
```



```

30     <servlet-name>webservices</servlet-name>
31     <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>
32     <init-param>
33         <param-name>transformWsdlLocations</param-name>
34         <param-value>true</param-value>
35     </init-param>
36     <init-param>
37         <param-name>contextConfigLocation</param-name>
38         <param-value></param-value>
39     </init-param>
40     <load-on-startup>1</load-on-startup>
41 </servlet>
42
43 <servlet-mapping>
44     <servlet-name>webservices</servlet-name>
45     <url-pattern>*.wsdl</url-pattern>
46 </servlet-mapping>
47
48 <servlet-mapping>
49     <servlet-name>webservices</servlet-name>
50     <url-pattern>/endpoints/*</url-pattern>
51 </servlet-mapping>
52
53 </web-app>

```

Primero que nada tengo que decirle al contexto donde se encuentra el archivo de configuración spring para que lo pueda levantar en el inicio.

```

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/config/spring.xml
    </param-value>
</context-param>

```

El siguiente paso es cargar el contexto de aplicación de Spring con el archivo de configuración se ha definido anteriormente.

```

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

```

El siguiente paso es trabajar sobre el Servlet del servicio Web que intercepta las peticiones HTTP entrantes. Asegura WSDL es consciente del contexto. Transforma dirección de SOAP a localhost: 8080. ContextConfigLocation con un conjunto de parámetros vacío significa que la Spring no intentará cargar la configuración webservices-servlet.xml defecto.

```

<servlet>
    <servlet-name>webservices</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>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value></param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
<servlet-name>webservices</servlet-name>
<url-pattern>*.wsdl</url-pattern>
</servlet-mapping>

<servlet-mapping>
<servlet-name>webservices</servlet-name>
<url-pattern>/endpoints/*</url-pattern>
</servlet-mapping>

```

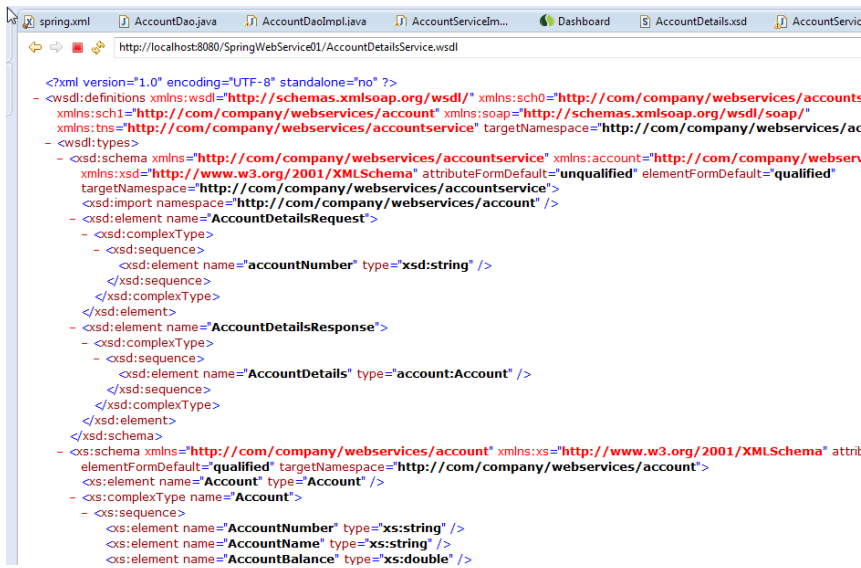
Probando nuestro Servicio con SOAPUI

SoapUi es una herramienta que entre otras cosas nos permite realizar un intercambio de mensajes con servicios web, es decir nos permite cargar datos al input y nos muestra la respuesta del servicio.

Nota: La primera prueba de fuego de nuestro servicio es que debe responder al wsdl, si esto no sucede no vamos para ningún lado. Y ojo por que vamos a necesitar el WSDL.

En nuestro caso el wsdl sera

<http://localhost:8080/SpringWebService01/AccountDetailsService.wsdl>

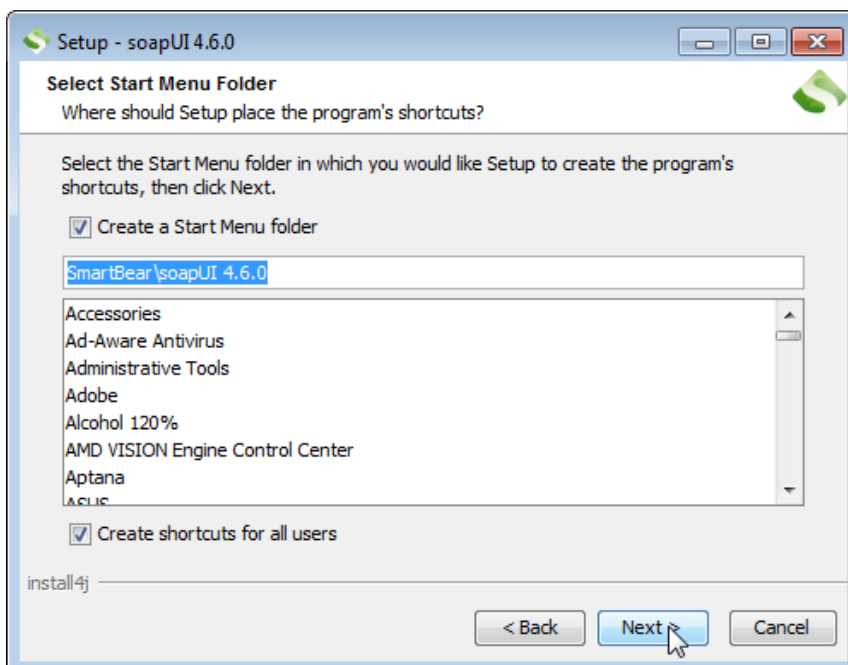
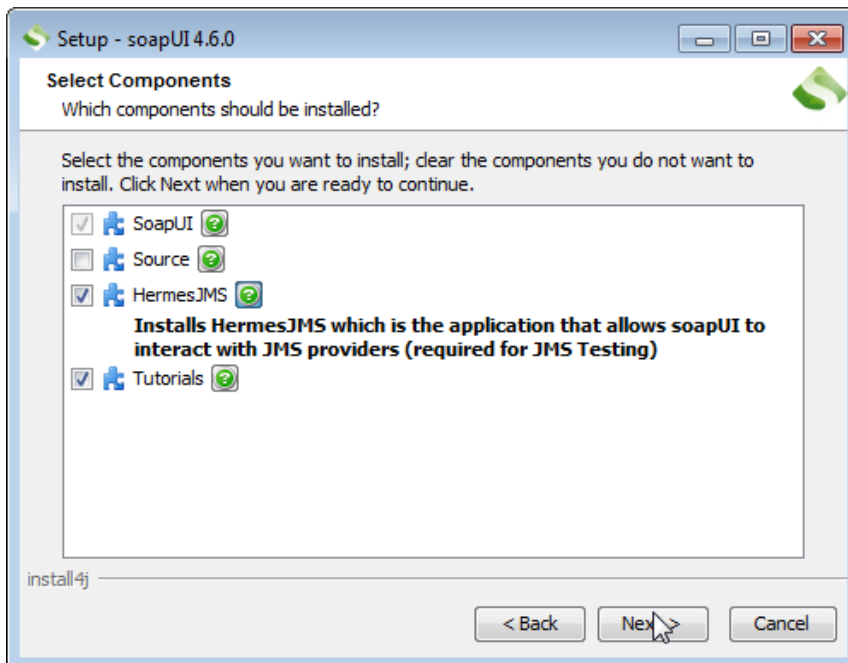


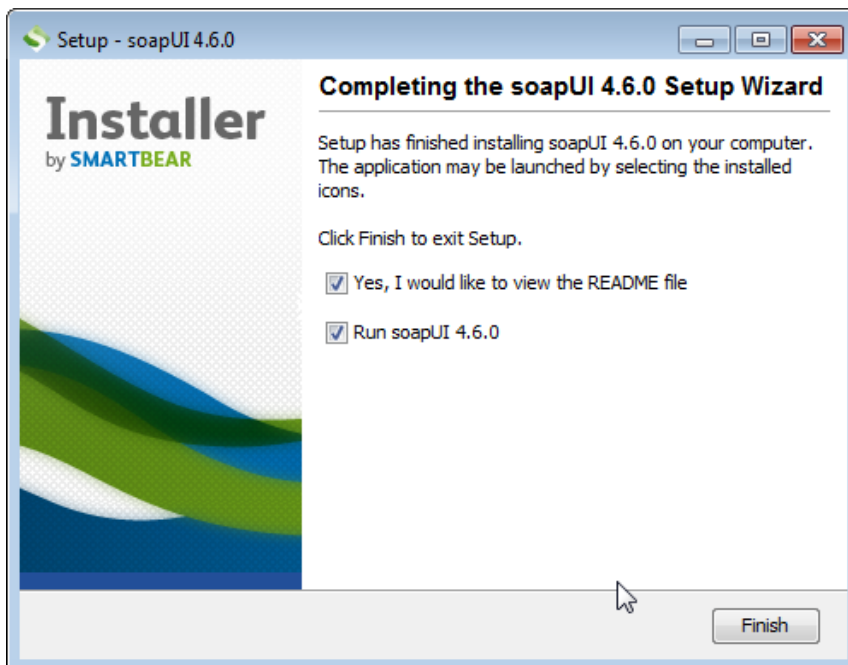
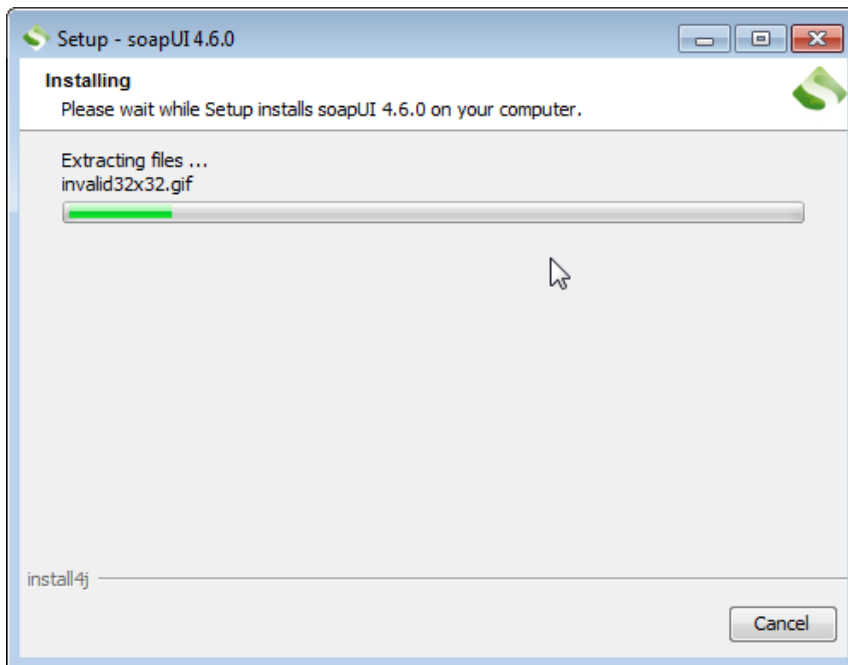
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sch0="http://com/company/webservices/accounts"
xmlns:sch1="http://com/company/webservices/account" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://com/company/webservices/accountservice" targetNamespace="http://com/company/webservices/ac"
<wsdl:types>
<xsd:schema xmlns="http://com/company/webservices/accountservice" xmlns:account="http://com/company/webserv"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://com/company/webservices/accountservice">
<xsd:import namespace="http://com/company/webservices/account" />
<xsd:element name="AccountDetailsRequest">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="accountNumber" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="AccountDetailsResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="AccountDetails" type="account:Account" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<xs:schema xmlns="http://com/company/webservices/account" xmlns:xs="http://www.w3.org/2001/XMLSchema" attrit
elementFormDefault="qualified" targetNamespace="http://com/company/webservices/account">
<xs:element name="Account" type="Account" />
<xs:complexType name="Account">
<xs:sequence>
<xs:element name="AccountNumber" type="xs:string" />
<xs:element name="AccountName" type="xs:string" />
<xs:element name="AccountBalance" type="xs:double" />

```

El siguiente paso es intalar el SOAPUI que no es nada complicado

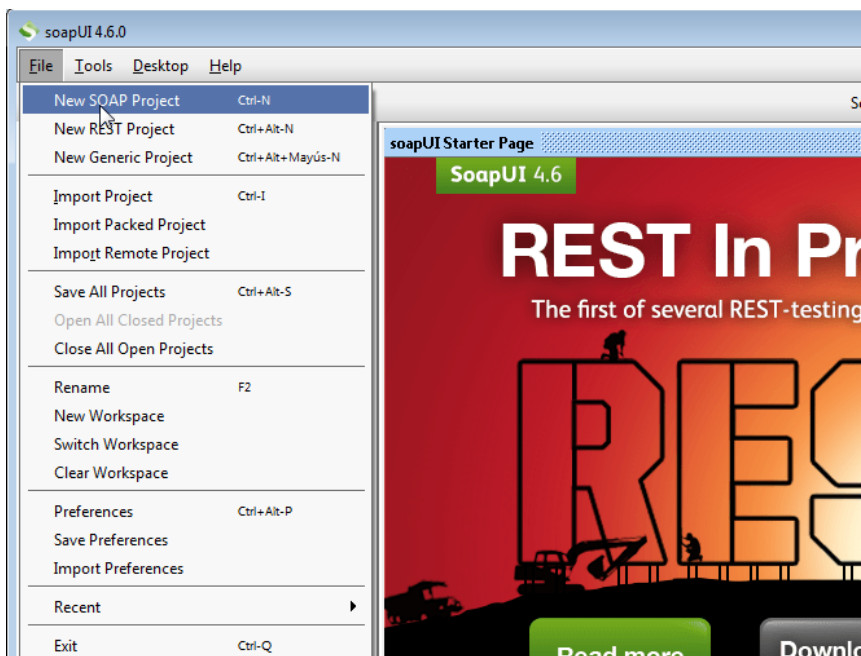




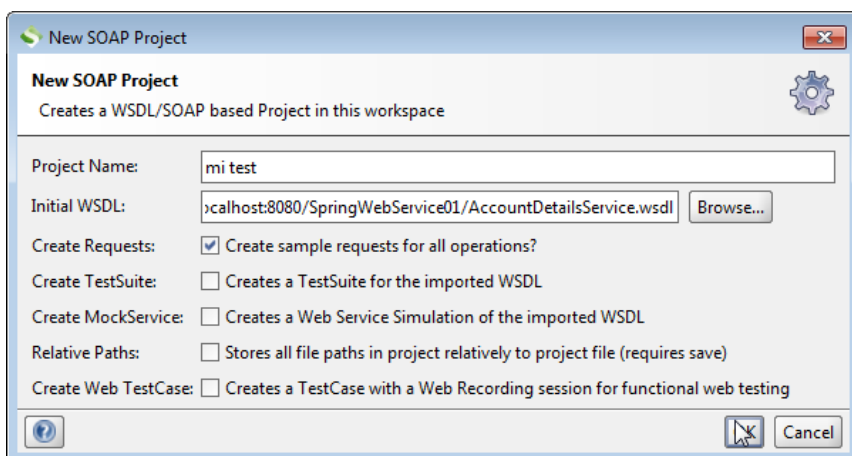


El siguiente paso es crear nuestro proyecto SOAP con nuestro WSDL y realizar las pruebas.

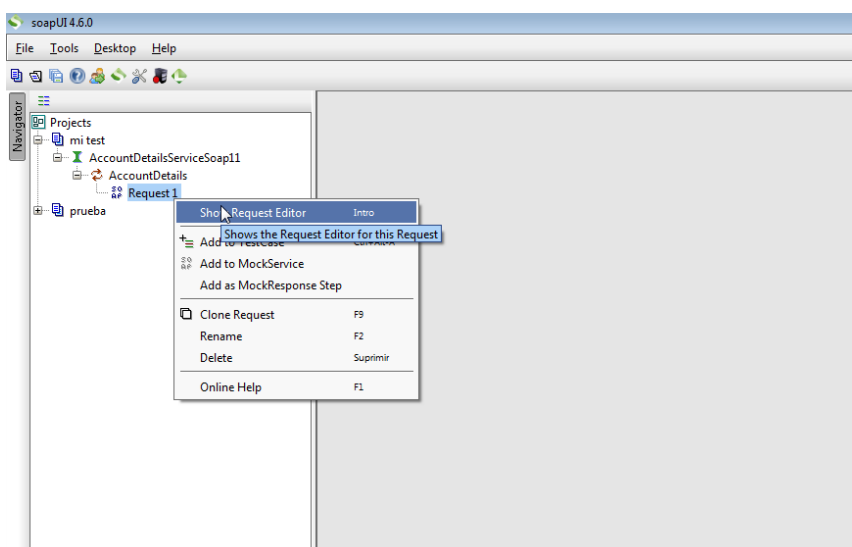
Primero vamos a crear nuestro proyecto



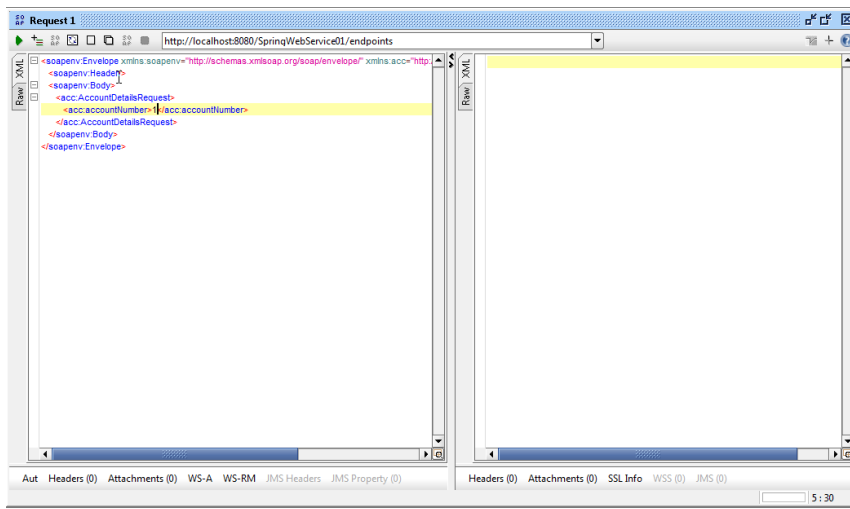
El siguiente paso es cargar nuestro WSDL.



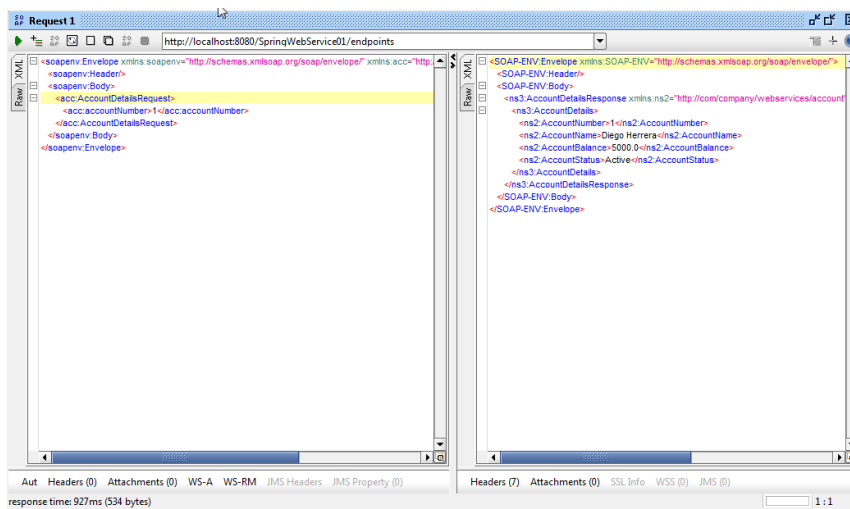
El siguiente paso es abrir el editor de solicitudes.



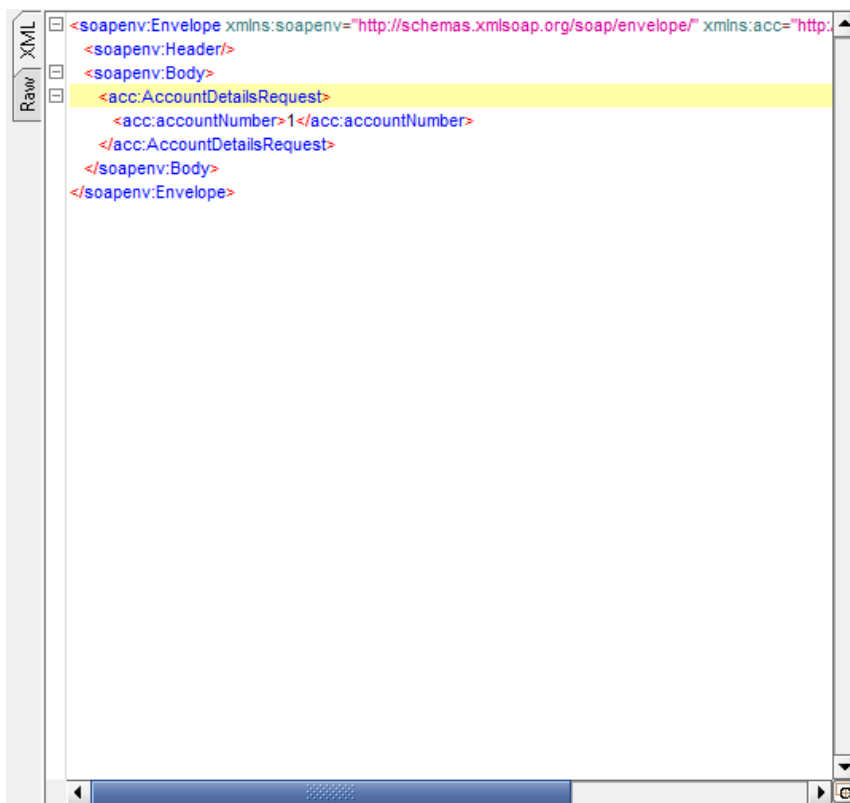
El siguiente paso es cargar el input de la solicitud que realizaremos al Servicio.



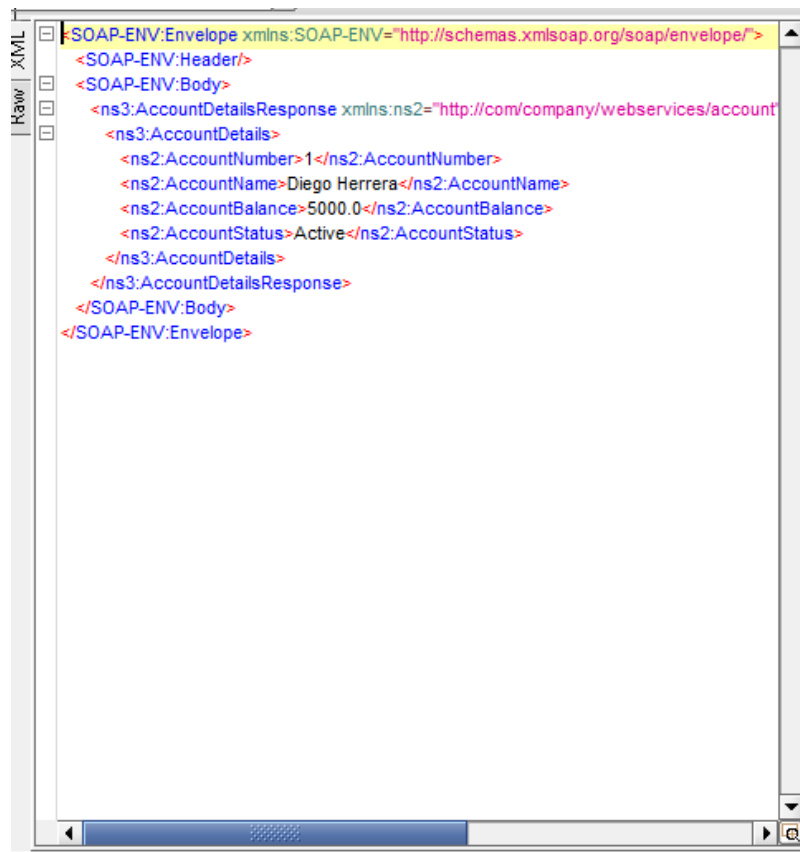
Le damos el boton verde de arriba a la izquierda para ejecutar la consulta al Servicio y obtener la respuesta.



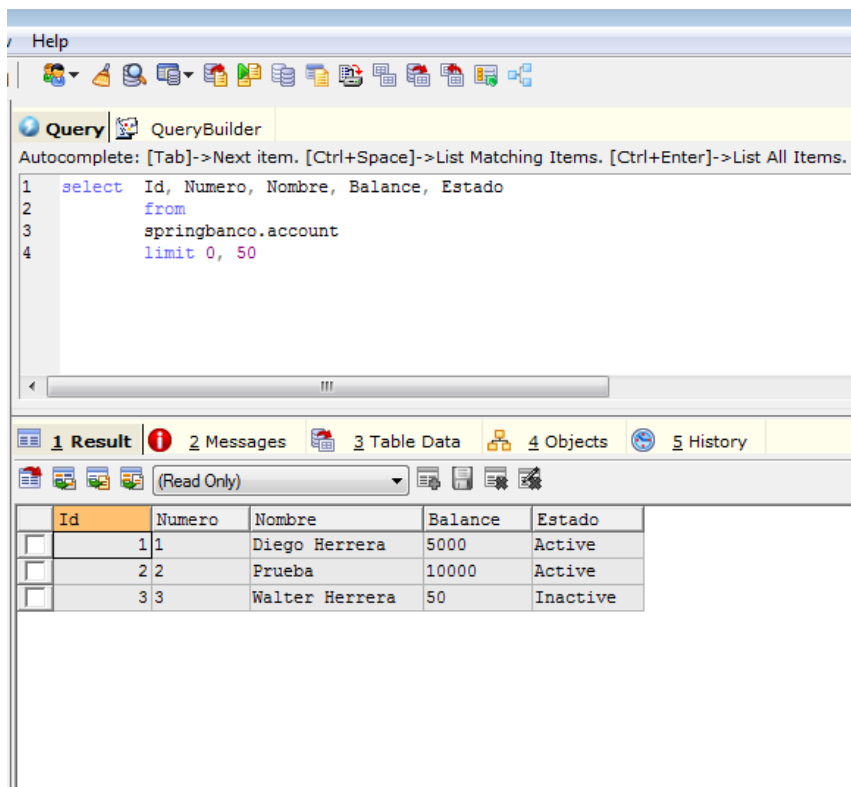
Esta fue la solicitud que mandamos.



Esta fue la respuesta del servicio.



A perdón un ultimo detalle, esta es una captura de la base de datos.como para antender que estamos consultando de mysql.



Código fuente del proyecto

El código fuente del proyecto lo podemos descargar. [Codigofuente.zip](http://codigofuente.zip)

El script de la base de datos mysql lo pueden descargar. [script.sql](#)

Publicado por [Diego Herrera](#) en [20:50](#)



Etiquetas: [Spring](#)

No hay comentarios:

Publicar un comentario

Introduce tu comentario...

Comentar como: [Javier Martín A](#) ▼

Cerrar sesión

Publicar

Vista previa

☐ Avisarme

[Entrada más reciente](#)

[Página principal](#)

[Entrada antigua](#)

Suscribirse a: [Enviar comentarios \(Atom\)](#)

Tema Fantástico, S.A.. Con la tecnología de [Blogger](#).