

Servicios con JAX-WS

solución con JAX-WS

el estándar JAX-WS

JAX-WS (Java API for XML Web Services):

- Es un **estándar Java EE**, presente desde la versión 5.
- Es decir, JAX-WS **no es un framework como tal**. Cada servidor de aplicaciones lo implementa, y también existe una "Reference Implementation", llamada JAX-WS RI, como parte del proyecto "GlassFish Metro". Esta es la implementación utilizada en los ejercicios.
- Utiliza anotaciones de Java 5 para facilitar el desarrollo y despliegue, tanto del lado servidor como del lado cliente.
- JAX-WS RI contiene tanto las librerías como las herramientas de generación de código.

solución con JAX-WS

publicación del servicio

Para publicar un Web Service con JAX-WS RI, se requiere:

- Tener una aplicación Web ⁽¹⁾.
- Agregar a la aplicación las librerías de JAX-WS RI, que incluyen la API, la implementación de referencia, la API e implementación de JAXB, y el compilador XJC, entre otras.
- Declarar en el web.xml de la aplicación el Listener de JAX-WS, que lo inicializa, y el Servlet genérico de JAX-WS, que maneja la configuración y los mensajes SOAP de request y response.

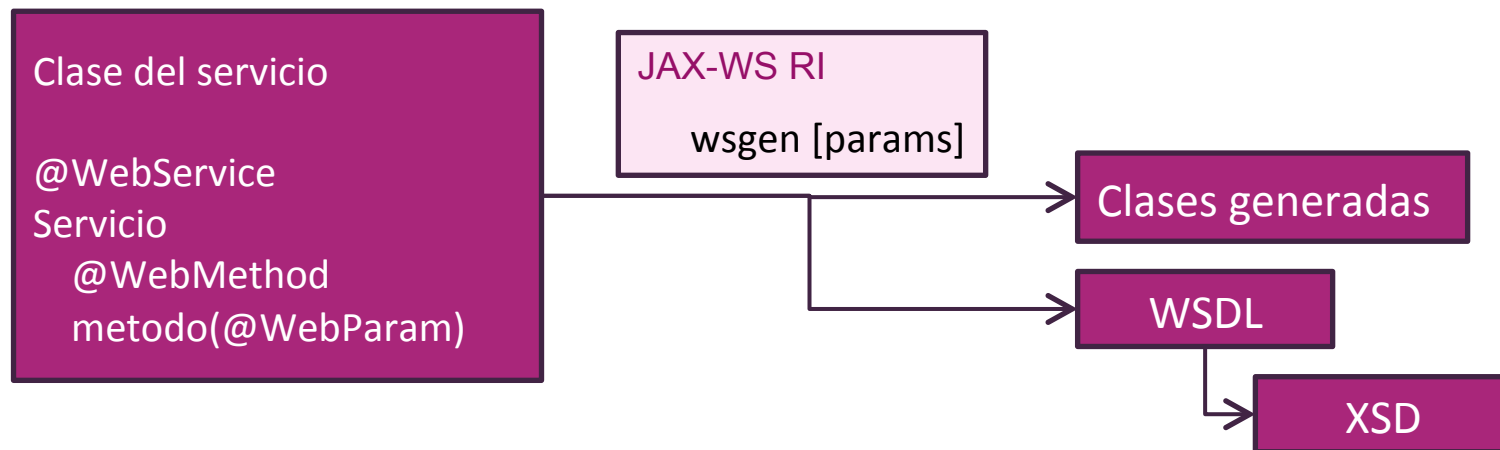
⁽¹⁾ Nota: Si se utiliza JAX-WS en un servidor de aplicaciones Java EE 5 o superior, también se puede publicar desde un módulo EJB 3. En este caso, el propio servidor provee la implementación de JAX-WS, así que no es necesario agregar las librerías de RI.

solución con JAX-WS

publicación del servicio

Si se utiliza Bottom Up:

- Implementar la clase que se expone como Web Service, y agregarle las anotaciones `@WebService`, `@WebMethod` y `@WebParam`.
- Utilizar el comando **wsgen** que viene con JAX-WS RI para generar las clases del servicio y el WSDL.

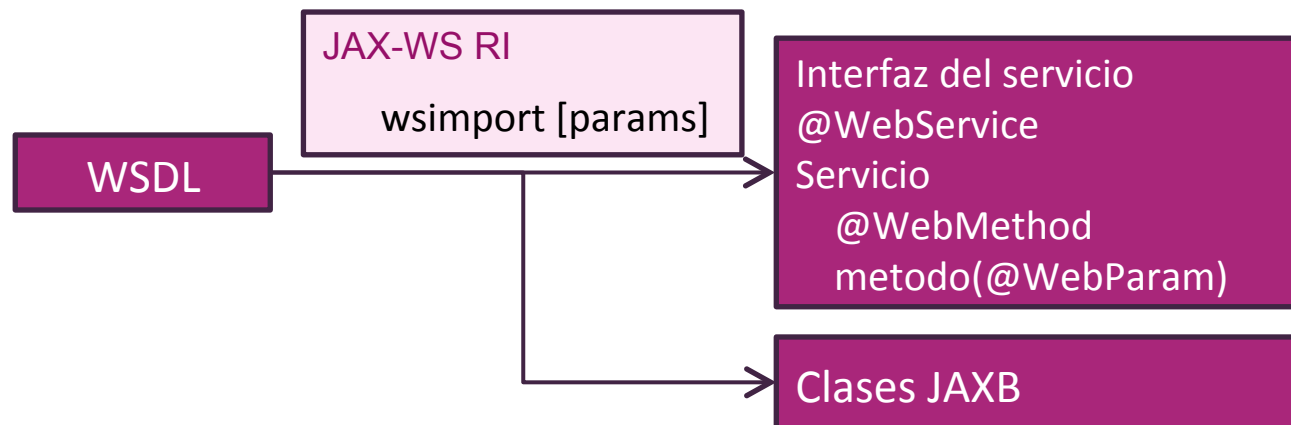


solución con JAX-WS

publicación del servicio

Si se utiliza Top Down:

- Implementar un archivo WSDL con la definición del servicio.
- Generar la interfaz java del servicio y las clases JAXB asociadas, utilizando el comando **wsimport** que viene con JAX-WS RI.



- Implementar la interfaz del servicio generada.

solución con JAX-WS

publicación del servicio

Configuraciones comunes:

- Tanto en Bottom Up como Top Down, una vez contruidos los servicios, se debe configurar un descriptor de los servicios, en un archivo sun-jaxws.xml, en el directorio WEB-INF.

Para conocer los detalles de la publicación y cómo probarla, se realizan dos casos prácticos, uno Bottom Up y otro Top Down.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Resumen del ejercicio:

- Utilizar proyecto con Spring y JAX-WS RI.
- Utilizar clase de negocio, implementada con Spring.
- Definir clase de servicio que expone la clase de negocio, con consideraciones de compatibilidad SOAP y concurrencia.
- Colocar anotaciones de JAX-WS en la clase de servicio.
- Completar la implementación del servicio, utilizando un bean de Spring.
- Publicar la clase de servicio como Web Service, con procedimiento Bottom Up.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

- Se utiliza el proyecto cp-spring-logic, ya cargado en el ejercicio anterior, que contiene las clases de negocio y datos configuradas en Spring, que se quiere exponer como Web Services. Las características del proyecto son:
 - Proyecto configurado en Maven, que utiliza Spring para su configuración.
 - Tiene una capa de persistencia con Hibernate y anotaciones, que funciona en forma transparente para los ejercicios. No es necesario entender cómo está construida, pues es contenido de otro curso del programa. Sí es importante saber qué hace.
 - Para ejecutar en Tomcat los proyectos web que utilizan este proyecto, se debe tener en ejecución previamente el motor de base de datos MySQL (ejecutando el comando `mysqld`).
 - Los detalles del proyecto están en el caso práctico 1 de Axis.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

- Importar el proyecto cp-jaxws-server, del tipo dynamic web project, que es donde se publican los Web Services con JAX-WS RI. Los servicios se implementan en este proyecto, aunque utilizan la lógica del proyecto cp-spring-logic. Las características del proyecto son:
 - Proyecto con Maven, que declara la dependencia de cp-spring-logic en el pom.xml.
 - Incluye las dependencias de JAX-WS RI, las cuales importan transitivamente el resto de las dependencias, entre las cuales está JAXB:

```
jsr173_api : 1.0 [compile]
└─ jaxws-rt : 2.2.7 [compile]
   └─ jaxws-api : 2.2.8 [compile]
      └─ jaxb-api : 2.2.4 [compile]
         └─ stax-api : 1.0-2 [compile]
            └─ activation : 1.1 [compile]
               └─ saaj-api : 1.3.4 [compile]
                  └─ javax.annotation : 3.1.1 [runtime]
                     └─ jsr181-api : 1.0-MR1 [runtime]
                        └─ jaxb-impl : 2.2.6 [compile]
                           └─ saaj-impl : 1.3.19 [compile]

jaxws-rt : 2.2.7 [compile]
└─ ...
   └─ streambuffer : 1.5 [compile]
      └─ woodstox-core-asl : 4.1.2 [compile]
         └─ stax2-api : 3.1.1 [compile]
            └─ mimepull : 1.8 [compile]
               └─ policy : 2.3.1 [compile]
                  └─ stax-ex : 1.7 [compile]
                     └─ gmbal-api-only : 3.1.0-b001 [compile]
                        └─ ha-api : 3.1.8 [compile]
```

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Primero se implementa la lógica del servicio a publicar. Para ello, se pide lo siguiente:

- Se quiere publicar el método **issueTypesForScope** de IssueBS.
- Tanto la clase del servicio como todas las que utiliza se publican en el mismo paquete Java, "com.everis.bpe.ws". Con esto se evita que se utilicen múltiples namespaces en la parte "types" del WSDL, ya que cada paquete Java equivale a un namespace, y con ello no se afecta la interoperabilidad con plataformas que no soportan múltiples namespaces.
- El parámetro scopeName del método issueTypesForScope es compatible con SOAP, por ser un String, así que no es necesario realizar adaptaciones.
- El objeto de retorno List<IssueTypeDTO> en principio no es compatible con SOAP, por lo que el objeto de retorno se tendría que adaptar en la clase del servicio. Sin embargo, JAX-WS, que utiliza internamente JAXB, está preparado para manejar objetos tipo List, por lo que en este caso no es necesario realizar la adaptación.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Primero se implementa la lógica del servicio a publicar. Para ello, se pide lo siguiente:

- Para el control de excepciones, se utiliza un flag que indica si fue exitoso o no, y un mensaje de error. Para ello, la clase `IssueTypesResult` extiende de `AbstractResult`, al igual que el ejercicio anterior.
- Crear en la clase `IssueWS`, la del servicio a publicar, un método `issueTypesForScope` que reciba el mismo parámetro que el de `IssueBS`, y retorne un objeto del tipo `IssueTypesResult`, que contiene el resultado de ejecutar el método en `IssueBS`.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Continuando:

- En la implementación, se coloca IssueBS como un atributo, con su getter. Dentro del getter, se extrae la instancia desde el contenedor de Spring, con el siguiente código:

```
private IssueBS getIssueBS() {  
    WebApplicationContext ctx =  
        ContextLoader.getCurrentWebApplicationContext();  
    if (issueBS == null) {  
        issueBS = (IssueBS) ctx.getBean("issueBS");  
    }  
    return issueBS;  
}
```

- Esto externaliza IssueWS de Spring, lo que es importante, ya que issueBS es un **Singleton** pre-configurado, mientras que IssueWS es instanciado por JAX-WS **en cada invocación**.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Manejo de excepciones y mensajes de error:

- En la implementación del método `issueTypesForScope`, hacer un try-catch, y en la parte try utilizar `issueBS`. En caso de excepción, capturarla y colocar el atributo "success" de la respuesta en false, y como `errorMessage` el mensaje de la excepción. La excepción **no se propaga**, y de esa manera el servicio se desacopla del cliente.

```
try {  
    ...  
} catch (Exception e) {  
    result.setSuccess(false);  
    result.setErrorMessage(e.getMessage());  
}
```

- Colocar el resultado de `IssueBS`, de tipo `List<IssueTypeDTO>`, como atributo de `IssueTypesResult`.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Anotaciones de JAX-WS:

- Una vez implementada la clase del Web Service a publicar, se realizan los pasos para la publicación.
- Para utilizar JAX-WS en la publicación del servicio, se deben utilizar las anotaciones de JAX-WS. Se pide lo siguiente:
 - Anotar la clase IssueWS con `@WebService`. Esto marca la clase como Web Service, para que la reconozca JAX-WS.
 - Anotar el método `issueTypesForScope` con `@WebMethod`. Esto marca el método como una operación de Web Service para JAX-WS.
 - Anotar el parámetro "scopeName" de `issueTypesForScope` con la anotación `@WebParam(name="scopeName")`. Con esto, se reconoce el nombre del parámetro para efectos de la operación.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Generación de código de objetos JAXB:

- El servicio publicado utiliza JAXB para la conversión bidireccional desde Java a XML. Para ello, se deben generar las clases, para lo cual en el caso Bottom Up se utiliza la herramienta **wsgen**.
- Como se genera código fuente, para evitar que se junte con el del proyecto, se pide utilizar en el proyecto cp-jaxws-server la carpeta **src/main/java-ws**, que está asociada al **código fuente** del proyecto.
- En la raíz de cp-jaxws-server, completar el script gen-ws-server-jaxws-bottom-up.cmd, que utiliza la herramienta wsgen, con las siguientes consideraciones:
 - Se define variable JAXWS_HOME, en la ruta donde está instalado. La herramienta wsgen se ejecuta en la carpeta bin de dicha ruta.
 - Se define variable JAVA_HOME, utilizada internamente por wsgen.
 - Se regenera el directorio donde quedan temporalmente los archivos generados, en target\generated\jaxws y src\main\webapp\WEB-INF\wsdl.
 - Si la salida a internet es por proxy, se define WSGEN_OPTS con la configuración de servidor y puerto.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Configuración de la herramienta de generación:

- Si se abre una ventana de comandos en C:\BpE\jaxws-ri-2.2.7\bin, y se ejecuta:

wsgen -Xendorsed -help

se pueden ver las opciones que ofrece el comando. El parámetro **-Xendorsed** se agrega por compatibilidad, porque JDK 1.6 incluye JAX-WS 2.1.

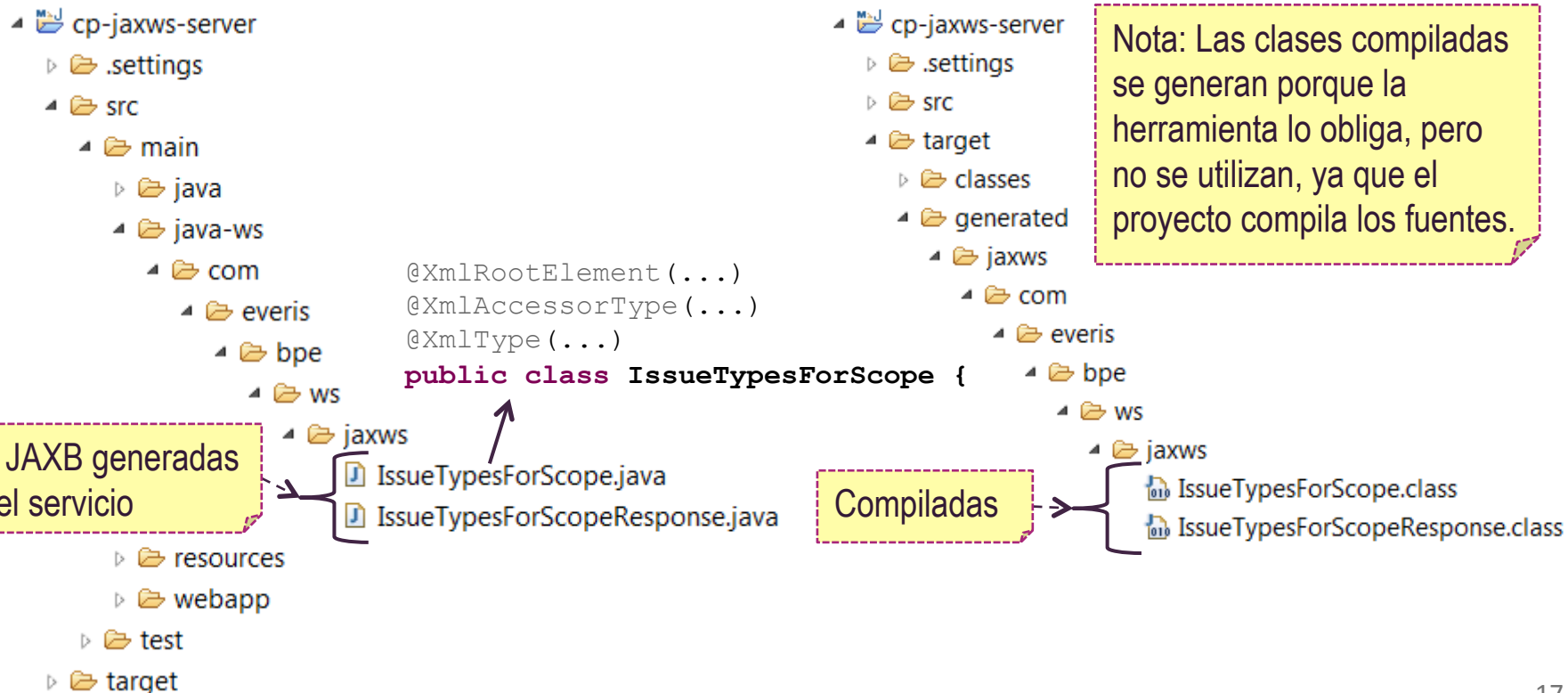
- Se pide agregar a la línea de comandos las siguientes configuraciones:
 - -Xendorsed: para compatibilidad con JAX-WS 2.2 desde JDK 1.6.
 - classpath del proyecto, utilizando el directorio de compilados del propio proyecto y también el de cp-spring-logic. Utilizar rutas relativas.
 - Directorio de código generado: target\generated\jaxws.
 - Directorio de código fuente generado: src\main\java-ws.
 - Marcar que se conserven los archivos generados.
 - Directorio de destino de los recursos: src\main\webapp\WEB-INF\wsdl.
 - Marcar que genere el WSDL, con la opción default (versión 1.1).
 - Se puede utilizar -verbose para ver mensajes del compilador de JAXB.
 - Al final se coloca la clase del servicio: com.everis.bpe.ws.IssueWS

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Continuando:

- Una vez configurado el script, ejecutarlo. Se deberían crear los siguientes archivos:



solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Adicionalmente, se debe completar el archivo WEB-INF/sun-jaxws.xml, que configura los servicios en JAX-WS RI:

```
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'  
  version='2.0'  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/jax-ws/ri/runtime  
    http://java.sun.com/webservices/docs/2.0/jaxws/sun-jaxws.xsd">  
  
  <endpoint name="IssueWS" ←----- Nombre del servicio  
    implementation="com.everis.bpe.ws.IssueWS"  
    url-pattern="/service/IssueWS" />  
</endpoints>
```

URL del servicio

Clase del servicio

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Para utilizar JAX-WS, se debe además declarar sólo una vez para todos los servicios el Listener y el Servlet para el manejo de las URL, en web.xml. Si se utiliza un servidor que soporta Servlet 3.0 (Java EE 6), esto no es necesario, aunque conviene colocarlo por compatibilidad. La nomenclatura es la siguiente:

```
<listener>
  <listener-class>
    com.sun.xml.ws.transport.http.servlet.WSServletContextListener
  </listener-class>
</listener>
<servlet>
  <servlet-name>JaxWsServlet</servlet-name>
  <servlet-class>
    com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>JaxWsServlet</servlet-name>
  <url-pattern>/service/*</url-pattern>
</servlet-mapping>
```

"service" debe corresponder con url-pattern de sun-jaxws.xml.

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Después de la publicación:

- Iniciar el servidor. En el caso de JAX-WS, no hay una URL genérica que publique la lista de servicios. Por lo tanto, hay que abrir directamente la URL del WSDL del servicio:

`http://localhost:8080/cp-jaxws-server/service/IssueWS?wsdl`

- Esto muestra el WSDL del servicio, el cual tiene los detalles de la implementación.
- Obsérvese que JAX-WS no coloca los detalles del XSD en el WSDL, sino que utiliza un XSD externo, importado desde el schema de la parte "types" del WSDL. La URL del XSD es:

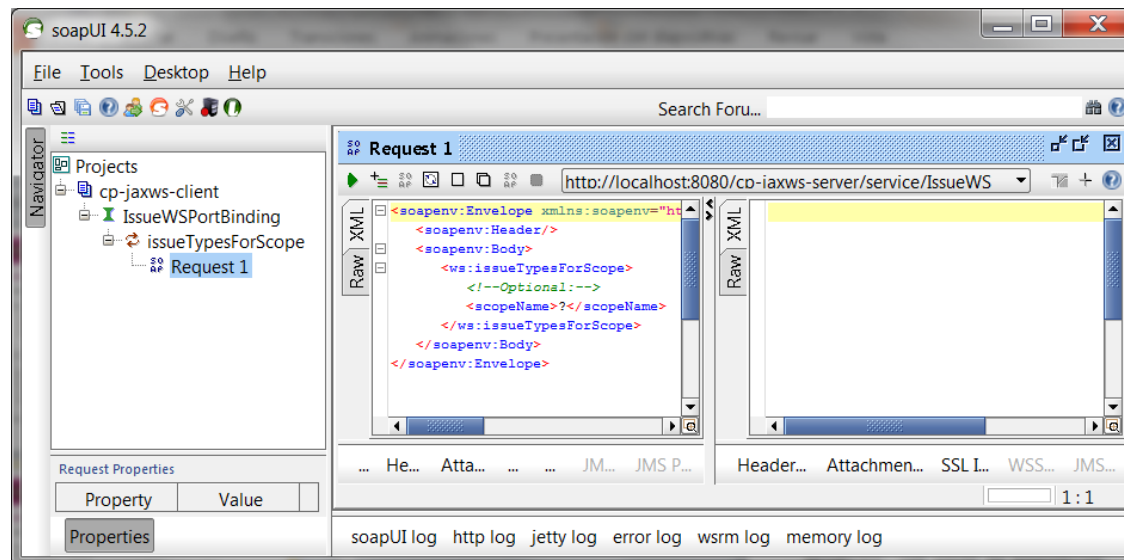
`http://localhost:8080/cp-jaxws-server/service/IssueWS?xsd=1`

solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Para probar el Web Service se utiliza la herramienta SOAP-UI, al igual que el ejercicio anterior. Se pide lo siguiente:

- Abrir SOAP-UI, y crear un nuevo proyecto llamado "cp-jaxws-client".
- Colocar la URL del WSDL del Web Service implementado.
- Abrir el elemento "Request 1", que genera un mensaje XML en formato SOAP con los datos a enviar al Web Service:

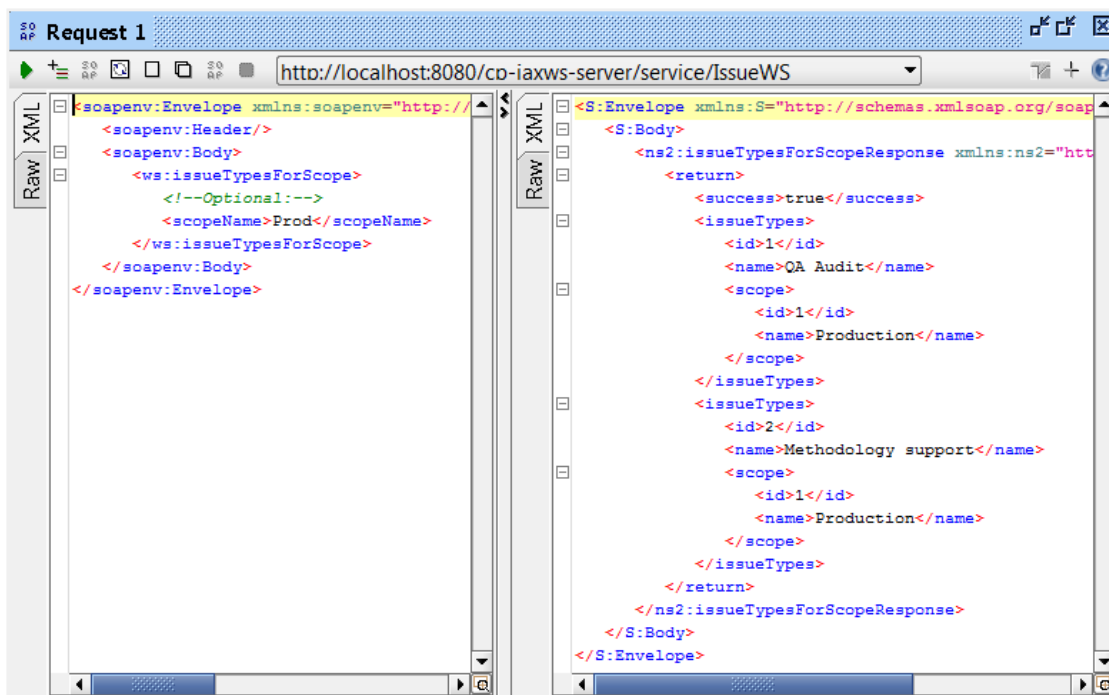


solución con JAX-WS

Caso práctico 5-1: Publicación Bottom Up de Web Service con JAX-WS

Continuando:

- Colocar en el campo scopeName el valor "Prod":
- Enviar el mensaje y obtener la respuesta:



The screenshot shows a web service client interface with two panels. The left panel, titled 'Request 1', shows a SOAP request XML. The right panel shows the corresponding SOAP response XML.

Request XML:

```
<?xml version='1.0' encoding='UTF-8'>
<soapenv:Envelope xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/'>
  <soapenv:Header/>
  <soapenv:Body>
    <ws:issueTypesForScope>
      <!--Optional:-->
      <scopeName>Prod</scopeName>
    </ws:issueTypesForScope>
  </soapenv:Body>
</soapenv:Envelope>
```

Response XML:

```
<?xml version='1.0' encoding='UTF-8'>
<S:Envelope xmlns:S='http://schemas.xmlsoap.org/soap/envelope/'>
  <S:Body>
    <ns2:issueTypesForScopeResponse xmlns:ns2='http://schemas.xmlsoap.org/soap/envelope/'>
      <return>
        <success>true</success>
        <issueTypes>
          <id>1</id>
          <name>QA Audit</name>
          <scope>
            <id>1</id>
            <name>Production</name>
          </scope>
        </issueTypes>
        <issueTypes>
          <id>2</id>
          <name>Methodology support</name>
          <scope>
            <id>1</id>
            <name>Production</name>
          </scope>
        </issueTypes>
      </return>
    </ns2:issueTypesForScopeResponse>
  </S:Body>
</S:Envelope>
```

Si se recibe un response con success en "true", la ejecución es exitosa. En la base de datos existen dos registros que cumplen el criterio. Se puede comprobar en la base de datos, tabla T_ISSUE_SCOPES y T_ISSUE_TYPES.

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Resumen del ejercicio:

- Utilizar proyecto con Spring y JAX-WS.
- A partir de una definición de servicio dado su WSDL, generar las clases que lo publican, utilizando la herramienta wsimport.
- Completar la implementación del servicio, utilizando la clase de negocio configurada en Spring.
- Probar el servicio utilizando la herramienta SOAP-UI.

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

A continuación, se publica un Web Service a partir de la definición dada por un WSDL, es decir, la opción Top Down. Se utiliza el mismo servicio que los ejercicios anteriores.

- El WSDL ya está construido, y se encuentra en el proyecto cp-spring-logic, en:
src/main/wSDL/IssueSimpleWS.wsdl
- Según se observa en el WSDL, el servicio recibe el código de "importance" y retorna su descripción.
- Internamente define un "simpleType" llamado "Importance", asociado a un string, con un conjunto de valores posibles:

```
<simpleType name="Importance">  
  <restriction base="xsd:string">  
    <enumeration value="CRITICAL" />  
    <enumeration value="MAJOR" />  
    <enumeration value="LOW" />  
  </restriction>  
</simpleType>
```

Esto equivale a un "enum". JAX-WS maneja este tipo de objetos, como se ve más adelante en el código generado.

- Como se genera código desde el WSDL, para evitar que se junte con el del proyecto, se pide crear en el proyecto cp-jaxws-server la carpeta **src/main/java-ws**, y asociarla al **código fuente** del proyecto.

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Generación de código de objetos JAXB:

- El servicio que se quiere publicar utiliza JAXB para la conversión bidireccional desde Java a XML. En este caso, se deben generar las clases desde el WSDL, o caso Top Down, para lo cual se utiliza la herramienta **wsimport**.
- Como se genera código fuente, para evitar que se junte con el del proyecto, se utiliza la carpeta **src/main/java-ws**, al igual que el ejercicio con Bottom Up.
- En la raíz de cp-jaxws-server, completar el script gen-ws-server-jaxws-top-down.cmd, que utiliza la herramienta wsimport, con las siguientes consideraciones:
 - Se define variable JAXWS_HOME, en la ruta donde está instalado. La herramienta wsimport se ejecuta en la carpeta bin de dicha ruta.
 - Se define variable JAVA_HOME, utilizada internamente por wsimport.
 - Se regenera el directorio donde quedan temporalmente los archivos generados, en target\generated\jaxws.
 - Si la salida a internet es por proxy, se define WSIMPORT_OPTS con la configuración de servidor y puerto.

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Configuración de la herramienta de generación:

- Si se abre una ventana de comandos en C:\BpE\jaxws-ri-2.2.7\bin, y se ejecuta:

```
wsimport -Xendorsed -help
```

se pueden ver las opciones que ofrece el comando. El parámetro **-Xendorsed** se agrega por compatibilidad, porque JDK 1.6 incluye JAX-WS 2.1.

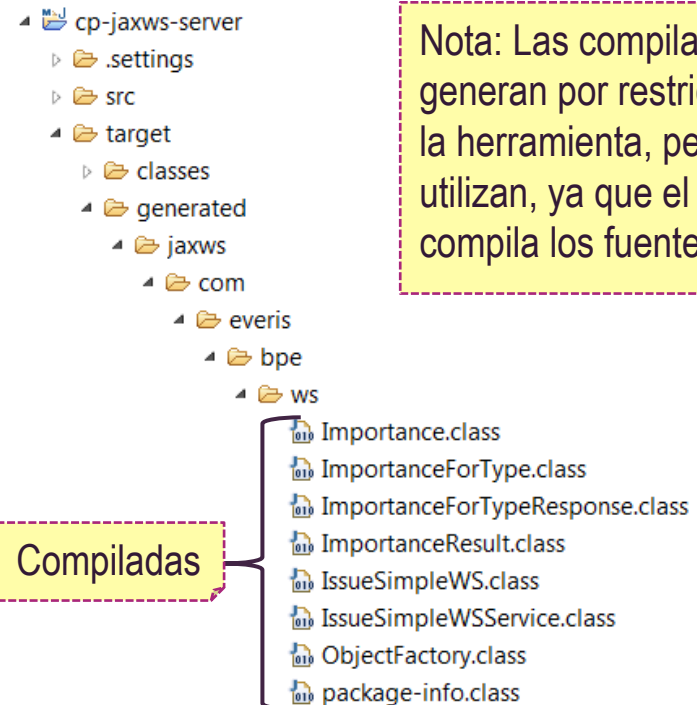
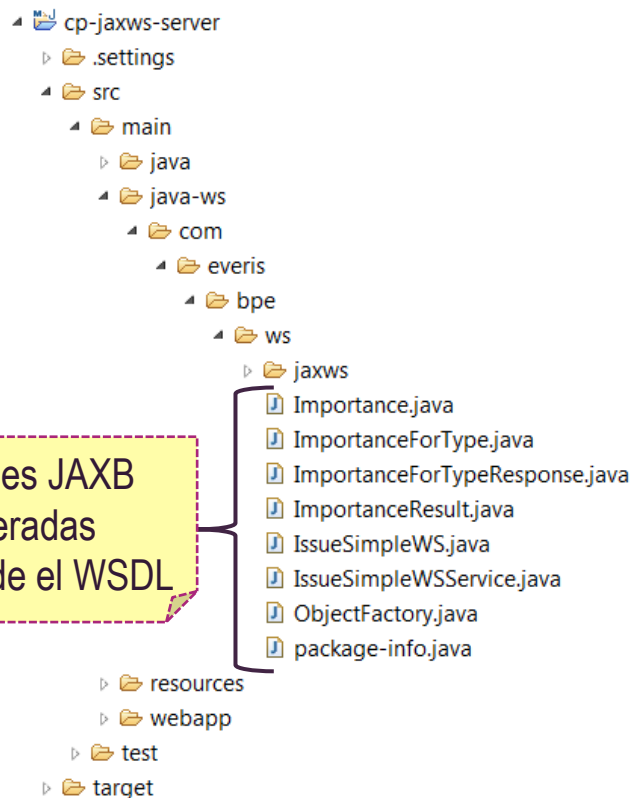
- Se pide agregar a la línea de comandos las siguientes configuraciones:
 - **-Xendorsed**: para compatibilidad con JAX-WS 2.2 desde JDK 1.6.
 - Directorio de código generado: `target\generated\jaxws`.
 - Directorio de código fuente generado: `src\main\java-ws`.
 - Marcar que se conserven los archivos generados, con **-keep**.
 - Se puede utilizar **-verbose** para ver mensajes del compilador de JAXB.
 - Al final se coloca la ruta relativa a `IssueSimpleWS.wsdl`:
`..\cp-spring-logic\src\main\wsdl\IssueSimpleWS.wsdl`

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Continuando:

- Una vez configurado el script, ejecutarlo. Se deberían crear los siguientes archivos:



Nota: Las compiladas se generan por restricciones de la herramienta, pero no se utilizan, ya que el proyecto compila los fuentes.

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Implementación del servicio:

- La herramienta wsimport genera la interfaz del servicio a publicar, IssueSimpleWS, con las anotaciones respectivas. Declara el método importanceForType.
- La herramienta wsimport **no** genera la clase que implementa el servicio. Esta clase debe construirse, agregando la anotación @WebService y referenciando al endpoint interface.
- Se pide crear la clase IssueSimpleWSImpl (en src/main/**java**, no en java-ws) con la anotación que referencia a la interfaz:

```
@WebService(endpointInterface="com.everis.bpe.ws.IssueSimpleWS")  
public class IssueSimpleWSImpl implements IssueSimpleWS { ... }
```

- Implementar en la clase el método importanceForType, utilizando la funcionalidad provista por IssueBS. Se puede reutilizar la del ejercicio de Axis.

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Para que el servicio sea registrado en JAX-WS RI, se debe agregar al archivo de configuración WEB-INF/sun-jaxws.xml:

```
<endpoint name="IssueSimpleWS"  
  implementation="com.everis.bpe.ws.IssueSimpleWSImpl"  
  url-pattern="/service/IssueSimpleWS" />
```

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Manejo de enum en JAX-WS:

- Abrir la clase generada Importance, asociada al "simpleType" de tipo string con valores predefinidos. Nótese que la clase generada asociada a Importance es un enum de Java 5, con anotaciones de JAXB y métodos de compatibilidad.

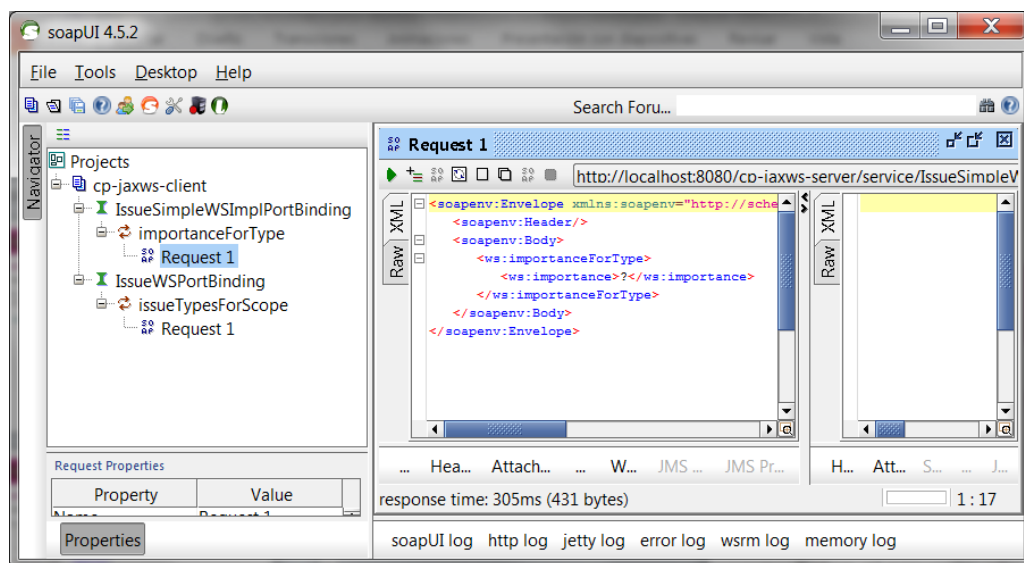
```
@XmlType(name = "Importance")
@XmlEnum
public enum Importance {
    CRITICAL,
    MAJOR,
    LOW;
    public String value() {
        return name();
    }
    public static Importance fromValue(String v) {
        return valueOf(v);
    }
}
```

solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Para probar el Web Service se utiliza nuevamente SOAP-UI. Se pide lo siguiente:

- Abrir SOAP-UI, y utilizar el proyecto llamado "cp-jaxws-client".
- Sobre el proyecto, clic derecho, opción "Add WSDL". Colocar la URL del WSDL del Web Service IssueSimpleWS.
- Abrir el elemento "Request 1", que genera un mensaje XML en formato SOAP con los datos a enviar al Web Service:

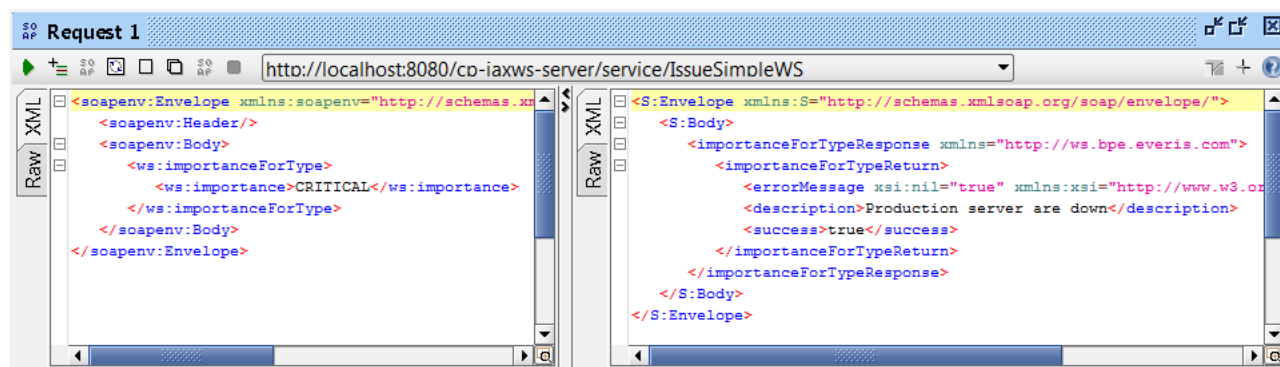


solución con JAX-WS

Caso práctico 5-2: Publicación Top Down de Web Service con JAX-WS

Continuando:

- Colocar el valor del campo importance, considerando que admite los valores especificados en el simpleType llamado "Importance".
- Enviar el mensaje y obtener la respuesta, que contiene la descripción asociada:



The screenshot shows a SOAP client interface with two panels. The left panel, titled 'Request 1', shows the XML request for the 'IssueSimpleWS' service. The right panel shows the XML response. The response indicates a successful execution with 'success=true' and a description of the production server status.

```
Request 1
http://localhost:8080/co-iaxws-server/service/IssueSimpleWS

Raw XML
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:importanceForType>
      <ws:importance>CRITICAL</ws:importance>
    </ws:importanceForType>
  </soapenv:Body>
</soapenv:Envelope>

Raw XML
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <importanceForTypeResponse xmlns="http://ws.bpe.everis.com">
      <importanceForTypeReturn>
        <errorMessage xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <description>Production server are down</description>
        </errorMessage>
        <success>true</success>
      </importanceForTypeReturn>
    </importanceForTypeResponse>
  </S:Body>
</S:Envelope>
```

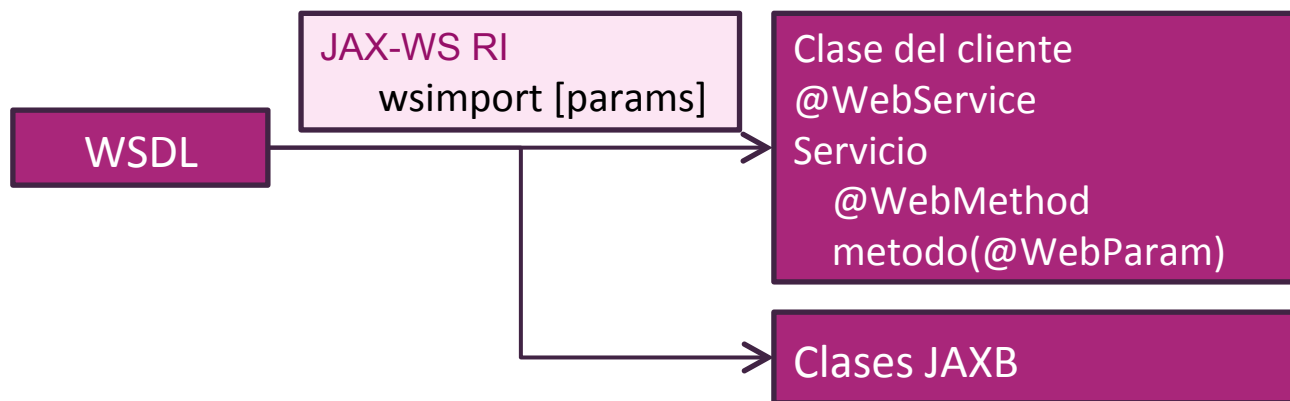
Si se recibe un response con success en "true", la ejecución es exitosa. Además, se puede comprobar la descripción en la base de datos, tabla T_ISSUE_IMPORTANCES.

solución con JAX-WS

consumo del servicio

Para consumir un Web Service con JAX-WS, se tiene en cuenta lo siguiente:

- El consumo es Java SE, es decir, no requiere ejecutarse en un servidor de aplicaciones o entorno particular. Esto permite probarlo con JUnit.
- Agregar las librerías de JAX-WS RI que se requieran como dependencias.
- Generar las clases del cliente, utilizando la herramienta **wsimport**.



Nota: Se generan las mismas clases del caso publicación Top Down. La utilización es la que cambia.

solución con JAX-WS

consumo del servicio

- Para utilizar la clase cliente, se crea una nueva instancia, y se obtiene el *port* del servicio, con el método getter respectivo. La ejecución del *port* realiza la invocación del servicio.
- Se recomienda seguir un patrón de desacoplamiento para evitar que se propaguen clases que utiliza JAX-WS RI, que tienen anotaciones JAX-WS o JAXB.

Para conocer los detalles, se realiza un caso práctico.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Resumen del ejercicio:

- Utilizar proyecto con Spring y JAX-WS.
- Con el servicio publicado, obtener la URL del WSDL.
- Generar las clases del cliente a partir del WSDL, utilizando la herramienta wsimport.
- Implementar una solución que desacopla el cliente de JAX-WS, utilizando Spring.
- Configurar la URL del servicio en un archivo properties.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

- Importar el proyecto cp-jaxws-client, que es donde se implementan las clases de los clientes de los Web Services del proyecto cp-jaxws-server. Las características del proyecto son:
 - Proyecto con Maven. No tiene dependencias del proyecto servidor, ya que está desacoplado de éste.
 - Incluye las dependencias de JAX-WS RI, las cuales importan transitivamente el resto de las dependencias, entre las cuales está JAXB.
- Levantar el servidor Tomcat que contiene el proyecto cp-jaxws-server. Esto es necesario ya que el WSDL se lee directamente desde la URL del servicio.
- Como se genera código desde un WSDL, para evitar que se junte con el del proyecto, se pide utilizar en el proyecto cp-jaxws-client la carpeta **src/main/java-wsclient**, que ya está asociada al **código fuente** del proyecto.

Nota: Aunque en este ejercicio se consume con cliente JAX-WS desde servidor JAX-WS, se podría también consumir con servidores de otras tecnologías.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Generación de código de objetos JAXB:

- El cliente que se quiere generar utiliza JAXB para la conversión bidireccional desde Java a XML. En este caso, se deben generar las clases desde el WSDL, para lo cual se utiliza la herramienta **wsimport**. Es un caso muy parecido a la publicación tipo Top Down con JAX-WS.
- En la raíz de cp-jaxws-client, completar el script gen-ws-client-jaxws.cmd, que utiliza la herramienta wsimport, con las siguientes consideraciones:
 - Se define variable JAXWS_HOME, en la ruta donde está instalado. La herramienta wsimport se ejecuta en la carpeta bin de dicha ruta.
 - Se define variable JAVA_HOME, utilizada internamente por wsimport.
 - Se regenera el directorio donde quedan temporalmente los archivos generados, en target\generated\jaxws.
 - Si la salida a internet es por proxy, se define WSIMPORT_OPTS con la configuración de servidor y puerto.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Configuración de la herramienta de generación:

- Si se abre una ventana de comandos en C:\BpE\jaxws-ri-2.2.7\bin, y se ejecuta:

```
wsimport -Xendorsed -help
```

se pueden ver las opciones que ofrece el comando. El parámetro **-Xendorsed** se agrega por compatibilidad, porque JDK 1.6 incluye JAX-WS 2.1.

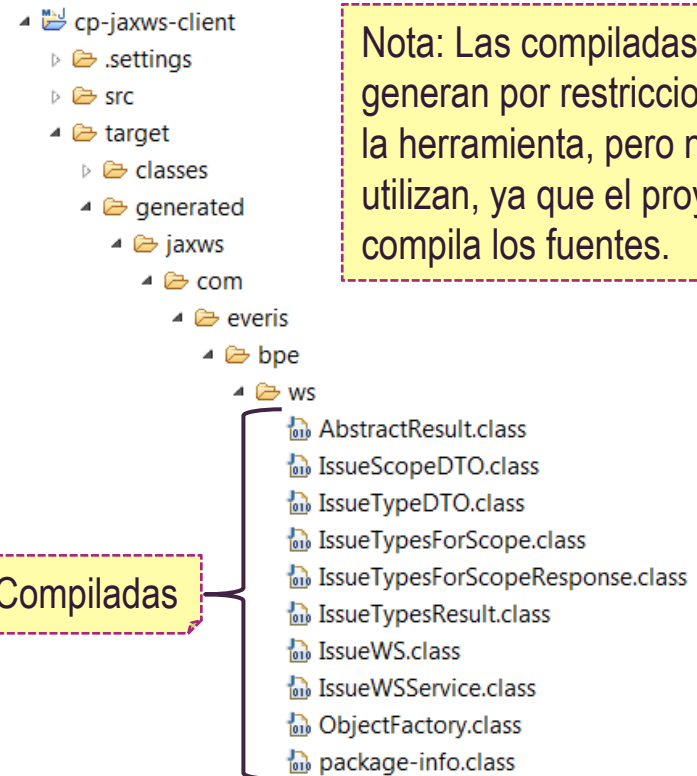
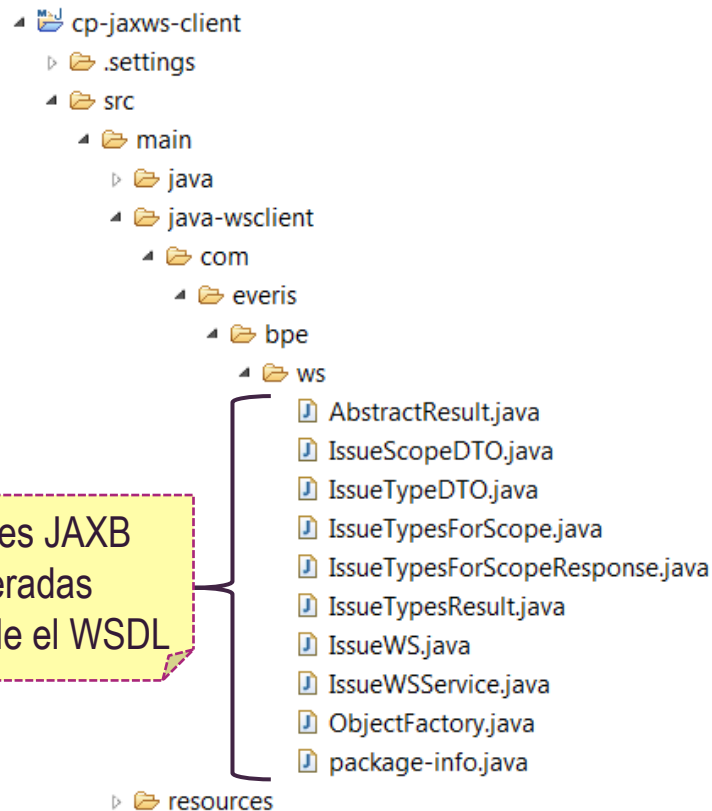
- Se pide agregar a la línea de comandos las siguientes configuraciones:
 - **-Xendorsed**: para compatibilidad con JAX-WS 2.2 desde JDK 1.6.
 - Directorio de código generado: `target\generated\jaxws`.
 - Directorio de código fuente generado: `src\main\java-wsclient`.
 - Marcar que se conserven los archivos generados, con **-keep**.
 - Se puede utilizar **-verbose** para ver mensajes del compilador de JAXB.
 - Al final se coloca la URL del WSDL:
`http://localhost:8080/cp-jaxws-server/service/IssueWS?wsdl`

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Continuando:

- Una vez configurado el script, ejecutarlo. Se deberían crear los siguientes archivos:



Nota: Las compiladas se generan por restricciones de la herramienta, pero no se utilizan, ya que el proyecto compila los fuentes.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Ejecución del cliente:

- La ejecución básica del cliente se realiza instanciando la clase de servicio generada, llamada IssueWSService. Como primer paso, se pide construir un test.
- La clase IssueWSTest debe estar en src/test/java, paquete "com.everis.bpe.ws". No necesita levantar contexto de Spring porque instancia directamente el cliente.
- La clase IssueWSService no permite recibir un String con la URL del servicio, como en los casos anteriores. Lo que utiliza es un objeto URL **del WSDL**. Por lo tanto, para construir la instancia en el método marcado con @Before, se utiliza:

```
try {  
    URL url = new URL("http://localhost:8080/cp-jaxws-server/service/IssueWS?wsdl");  
    IssueWSService service = new IssueWSService(url);  
    issueWS = service.getIssueWSPort();  
} catch (MalformedURLException ex) {  
    throw new WebServiceException(ex);  
}
```

Nota: Se utiliza la URL del WSDL, no la del end point.

- Esto es necesario para poder especificar la URL del servicio, y no utilizar la generada internamente, siguiendo las buenas prácticas.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Ejecución del cliente:

- Crear el método de test para el método `issueTypesForScope`, y probar con el valor "Prod" para el parámetro `scopeName`.
- En el test, hacer un `assertTrue` sobre el atributo `success` de la respuesta, para verificar que la ejecución es correcta.
- Comprobar que el número de registros obtenido es mayor que cero.

solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Monitorización de mensajes:

- Abrir la herramienta "**tcpmon**", crear un nuevo y configurar un monitor con:
 - Local port: 18080
 - Server name: 127.0.0.1 (localhost)
 - Server port: 8080
- Ejecutar "Add monitor", y se abre una pantalla que captura y muestra los mensajes SOAP.
- Modificar el test de IssueWS, colocando como servidor y puerto de la URL del Web Service "localhost" y el valor de Local port.
- Ejecutar el test, y observar los mensajes que se muestran en tcpmon.

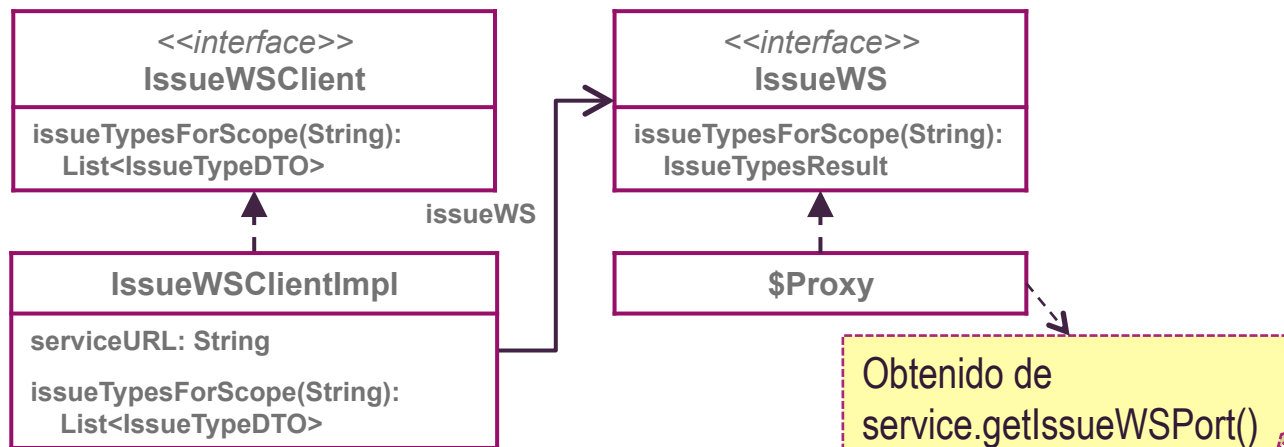
solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Desacoplamiento:

- La solución presentada tiene el problema que está acoplada a JAX-WS, tanto en la interfaz IssueWS (anotaciones de JAX-WS) como en el retorno IssueTypesResult (anotaciones de JAXB). Es decir, si una clase de negocio utilizara este cliente, tendría dependencias de JAX-WS en ejecución, y si se cambiara JAX-WS, tendría que cambiarse la clase de negocio.
- Una forma de desacoplarla es colocar una interfaz e implementación de wrapper, que adapte los parámetros, de modo que la clase que la invoque no conozca de JAX-WS.
- Se pide completar la implementación IssueWSCClientImpl, de acuerdo al diagrama:

Por simplicidad, desacoplar sólo de JAX-WS y no de JAXB, es decir, utilizar las mismas clases de retorno del cliente generado (IssueTypeDTO).



solución con JAX-WS

Caso práctico 5-3: Consumo de Web Service con JAX-WS

Desacoplamiento :

- La clase IssueWSCClientImpl hace un try-catch de la ejecución del cliente, y si el atributo success de la respuesta es false, lanza una excepción. El resultado que viene dentro de IssueTypesResult es el retornado por el método. De esta manera, su firma es similar al método de la interfaz de negocio original, y también se desacopla de la solución de manejo de errores del Web Service.
- Luego, se pide configurar el cliente desacoplado como bean de Spring en el archivo applicationContext-wsclient.xml, en src/main/resources. Inyectarle el atributo serviceURL desde el archivo spring-config-client.properties, para mayor flexibilidad.
- Construir un test src/test/java, paquete "com.everis.bpe.wsclient", IssueWSCClientTest, que levante un contexto de Spring, con el archivo applicationContext-wsclient.xml.
- Inyectar el cliente IssueWSCClient al test, con @Resource.
- Crear el método de test, y probar la clase del cliente con datos válidos, de la misma manera que el test anterior. Verificar con un assert el número de registros obtenidos.
- Utilizar tcpmon para observar los mensajes SOAP.