Com desenvolupar una aplicació que publiqui o consumeixi WebServices

A qui va dirigit

Aquest how-to va dirigit a tots aquells que hagin de desenvolupar una aplicació Canigó que publiqui o consumeixi WebServices.

Es presenta el procediment per crear i publicar un Web Service a partir de l'aplicació-plantilla de Canigó i utilitzant el servei de WebServices de Canigó. També s'inclou el procediment per a generar les classes Java client a partir d'un fitxer WSDL.

Versió de Canigó

Els passos descrits en aquest document són d'aplicació a les versions 2.x de Canigó.

Publicació d'un WebService

L'estratègia prevista a Canigó per publicar WebServices és la de generar-los a partir de mètodes d'un BO.

El servei de WebServices de Canigó simplifica el procés de generació d'un WebService ja que és suficient crear una interfície que representi el servei i publicar-ho via configuració.

A continuació es detallen els passos a seguir:

1. Configuració de l'aplicació web (només cal fer-ho un cop per aplicació)
2. Creació de la interfície (un cop per cada nou servei)
3. Configuració del nou WebService (un cop per cada nou servei)
4. Verificació. Consulta de WSDL (un cop per cada nou servei)

1. Configuració de l'aplicació Web

S'ha de crear un nou mapping per que el servlet principal gestioni les peticions als WebServices (la URL /ws/* és arbitraria, pot ser qualsevol altre que no tingui conflictes amb altres serveis).

```
<servlet-mapping>
  <servlet-name>petstore</servlet-name>
  <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

En l'exemple el `servlet-name` és `petstore` ja que s'ha partit de la aplicació-plantilla, en general és el servlet que te com a `class org.springframework.web.servlet.DispatcherServlet`.

2. Creació de la interfície

Es crea la interfície `HolaMonWS`

```
package net.gencat.ctti.canigo.samples.prototip.webservices;

public interface HolaMonWS {

        public String holaMon();

        public String echo(String texte);

}
```

Com desenvolupar una aplicació que publiqui o consumeixi WebServices

A mode d'exemple es presenta una classe (`HolaMonWSImpl`) que implementa el servei definit:

```
package net.gencat.ctti.canigo.samples.prototip.webservices;
import net.gencat.ctti.canigo.services.logging.LoggingService;
public class HolaMonWSImpl implements HolaMonWS {
    private LoggingService logService = null;
    public HolaMonWSImpl() {}
    public String echo(String texte) {
        return texte;
    }
    public String holaMon() {
        return "holaMon";
    }
    public LoggingService getLogService() {
        return logService;
    }
    public void setLogService(LoggingService logService) {
        this.logService = logService;
        logService.getLog(this.getClass()).info("Ja tinc log");
    }
}
```

3. Configuració del servei de WebServices de Canigó

S'afegeix l'exportació del nou Web Service a `canigo-services-webservices.xml`

Fitxer de configuració	**canigo-services-webservices.xml**
Ubicació proposada	<PROJECT_ROOT>/src/main/resources/spring

Per cada un dels WebServices s'ha de definir un bean dins de la llista "exportedInterfaces" del bean "webServicesService".

```
<bean abstract="true" id="ExportedInterfaceDefinition"
    class="net.gencat.ctti.canigo.services.webservices.impl.ExportedInterfaceImpl"/>
(...)
<bean name="webServicesService"
    class="net.gencat.ctti.canigo.services.webservices.impl.WebServicesServiceImpl">
  <property name="exportedInterfaces">
    <list>
      <bean parent="ExportedInterfaceDefinition">
        <property name="name" value="HolaMon"/>
        <property name="localInterface"
          value="net.gencat.ctti.canigo.samples.prototip.webservices.HolaMonWS"/>
        <property name="implementation"
```

```
            value="net.gencat.ctti.canigo.samples.prototip.webservices.HolaMonWSImpl"/>
        </bean>
      </list>
    </property>
</bean>
```

Quan es vol exposar un bean d'Spring s'haurà de fer servir:

```
<property name="instance" ref="beanHolaMon"/>
```

en lloc de:

```
<property name="implementation"
  value="net.gencat.ctti.canigo.samples.prototip.webservices.HolaMonWSImpl"/>
```

Fer servir un bean d'Spring permet que la classe que dona el servei pugui accedir a serveis Canigó i altres beans disponibles en el context. En cas de que el servei faci servir persistència s'haurà de fer servir l'exposició de bean i aquest haurà de tenir un proxy transaccional (com pot ser baseDaoProxy).

4. Verificació. Consulta del WSDL

Per comprovar que s'ha publicat correctament el servei web "`HolaMon`", podem consultar el seu WSDL. Per fer-ho, cal arrancar l'aplicació i accedir a la següent URL:

http://localhost:8080/canigo-env/ws/HolaMon?WSDL

Com desenvolupar una aplicació que publiqui o consumeixi WebServices

Consum d'un WebService

Per a consumir un servei web es presenten 3 alternatives:

Escenari		Què requereix?	Situació d'aplicació
A	Integrat amb Canigó a partir d'una URL	URL del servei Interface java del servei	Aquest cas es d'aplicació quan el proveïdor del servei dóna una interfície java. També es pot arribar aquí generant classes amb Axis i només fent us de la interfície. Avantatges: Configuració senzilla. Recomanat per serveis amb paràmetres bàsics Limitacions: Canigó farà servir Xfire per fer el binding de la interfície amb el web service. En WS complexos aquest binding pot ser erroni.
B	Integrat amb Canigó a partir d'una implementació axis	URL del servei Classes generades Axis	Aquest cas es d'aplicació quan no es disposa d'una interificie java equivalent o aquesta te tipus complexes. Avantatges: Integració amb la configuració de Canigó. Fàcil accés al servei Limitacions: Les propies d'un client Axis. Cal incorporar les dependències generades pel client.
C	Desenvolupament no integrat	Classes generades amb qualsevol eina (verificar conflicte de llibreries)	Aquest cas es d'aplicació quan cap de les dues alternatives Canigó es suficient. En aquest cas la tecnologia i el desenvolupament i configuració del client queda a elecció de l'arquitecte de l'aplicació

Abans de veure com utilitzar els serveis web des de Canigó, mostrarem com generar classes client Axis a partir d'un WSDL. Aquest procediment es d'aplicació tant per l'escenari A com B.

Generació de classes client a partir d'un WSDL

Procediment: generació de classes client Axis per un WebService

Entrada: url del webservice

Sortida: classes generades

Eines: ant + axis-ant.jar

Les darreres versions de Canigó treballen, com ja és sabut, amb XFire. De fet, el servei de Web Services del framework utilitza aquesta llibreria.

Tanmateix, per a la generació de les classes client utilitzarem Axis. El motiu pel qual s'utilitzarà Axis i no XFire és que amb aquest últim, la generació del client només és possible realitzar-la amb Java 5.0 i en conseqüència, fa que les classes generades siguin incompatibles amb la versió 1.4 (que és la versió de referència de Canigó).

El procediment que es descriu a continuació està basat en l'eina ANT, que caldrà tenir instal·lada[1].

[1] http://ant.apache.org/

Com desenvolupar una aplicació que publiqui o consumeixi WebServices

1. Estructura de directoris

Crear la següent estructura de directoris en una carpeta del vostre sistema:

- `/target`
- `/resources`
- `/lib`

I crear els fitxers a la carpeta arrel:

- `build.xml`
- `tasks.properties`

2. Llibreries requerides

Les llibreries necessàries per a generar les classes són:

- axis.jar
- axis-ant.jar
- commons-discovery.jar
- commons-logging.jar
- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar
- wsdl4j.jar

Afegir les llibreries esmentades en el directori `lib` que heu creat en el pas anterior.

3. Fitxer `build.xml`

```
<project name="Generate client (Axis)" default="generate-client">

    <property name="output.dir"   value="src"/>
    <property name="wsdl.file"    value="resources/HolaMon1.wsdl"/>
    <property name="lib"          value="lib"/>

    <path id="axis.classpath">
      <fileset dir="${lib}">
        <include name="**/*.jar"/>
      </fileset>
    </path>

    <taskdef resource="axis-tasks.properties" classpathref="axis.classpath" />

    <target name="generate-client">
            <axis-wsdl2java output="${output.dir}" testcase="true"
                    verbose="true" url="${wsdl.file}" >
        </axis-wsdl2java>
    </target>

</project>
```

Les propietats definides en el fitxer build.xml són les següents:

- output.dir. Directori on es generaran les classes
- lib. Directori on estan les llibreries

- wsdl.file. Pot ser tant un fitxer local com una url d'accés al WSDL. Exemple: resources/HolaMon.wsdl p http://localhost:8085/canigo-env/ws/HolaMon?WSDL

De la tasca `axis-wsdl2java`, que és la que s'encarrega de parsejar el WSDL i generar les classes, destacarem els atributs més rellevants:

- `output`: directori on es volen ubicar les classes Java generades
- `url`: path on es troba l'arxiu WSDL (tant pot ser un fitxer local com remot, al qual s'accedirà a partir d'una URL)

La resta i d'altres es poden consultar a la pàgina oficial d'Apache-Axis (url que indiquem en l'apartat de referències).

4. Generació del client

Finalment ja podem generar les classes Java client a partir d'un WSDL mitjançant ANT. Des de la línia de comandes i ubicant-nos en el directori arrel, executar:

```
> ant generate-client
```

D'aquesta manera s'executarà el contingut el fitxer `build.xml` i es crearan les classes generades en el directori que hem indicat a `output.dir`

NOTA: les classes generades tenen les seves pròpies dependències. Per tant si s'han d'integrar en una aplicació caldrà satisfer aquestes dependències. Per més informació sobre les versions consulta la documentació d'Axis.

Invocació de WebServices amb Canigó (Escenari A i B)

1. Importació de serveis WebServices integrats amb Canigó (Escenari A i B)

Per cada WebService importat cal afegir una entrada dins de la llista "`importedInterfaces`". Del bean "`webServicesService`" que trobarem al fitxer `canigo-services-webservices.xml`.

```
<beans>
   (...)
   <bean abstract="true" id="importedInterfaceDefinition"
      class="net.gencat.ctti.canigo.services.webservices.impl.ImportedInterfaceImpl"/>

    <bean name="webServicesService"
       class="net.gencat.ctti.canigo.services.webservices.impl.WebServicesServiceImpl">
       <property name="importedInterfaces">
          <list>
             <bean parent="importedInterfaceDefinition">
                 <property name="name" value="holaMonService">
                 (...)
             </bean>
          </list>
       </property>
       (...)
    </bean>
(...)
</beans>
```

Com desenvolupar una aplicació que publiqui o consumeixi WebServices

Depenent de l'escenari en que ens trobem la definició es farà d'una forma o una altra:

Escenari A: *URL + Interfície*

Podem accedir a un servei web indicant la seva URL i la interfície que conté els mètodes publicats. D'aquesta manera, Canigó utilitzarà el Client Locator que té per defecte, amb XFire.

```
(...)
<bean parent="importedInterfaceDefinition">
   <property name="name" value="holaMonService" />
   <property name="serviceURL"
        value="http://localhost:8080/canigo-env/ws/HolaMon"/>
    <property name="localInterface"
        value="net.gencat.ctti.canigo.samples.prototip.webservices.HolaMonWS
"/>
</bean>

(...)
```

Ens trobem certes limitacions si el servei al qual accedim mitjançant la seva URL fa el binding de dades del tipus RPC, degut a què XFire no suporta aquesta codificació.

Tanmateix, si el servei utilitza en els seus mètodes tipus de dades bàsics, no hauríem de trobar cap mena de problemàtica en el binding de dades.

Escenari B: Locator + Interfície

Per evitar el problema del punt anterior, Canigó ens ofereix una altra alternativa. En lloc d'indicar la url del servei, és possible indicar el service Locator del mateix; d'aquesta manera evitem l'ús del locator de XFire.

```
(...)
<bean parent="importedInterfaceDefinition">
   <property name="name" value="holaMonService" />
   <property name="serviceLocator"
      value="net.gencat.ctti.canigo.samples.prototip.webservices.HolaMonWSL
ocator"/>
    <property name="localInterface"
        value="net.gencat.ctti.canigo.samples.prototip.webservices.HolaMonWS"
/>
</bean>

(...)
```

Aquesta classe Locator ha estat generada a partir del fitxer WSDL del servei, la qual contindrà l'adreça on es troba el servei. S'utilitzarà Jax-RPC enlloc de XFire, i per tant, evitarem els problemes de codificació.

Les limitacions que podem trobar en aquest cas són les pròpies d'Axis, ja que les classes són generades amb aquesta API. Per exemple, Axis no suporta el tipus de dades java.util.Calendar.

Com desenvolupar una aplicació que publiqui o consumeixi WebServices

2. Utilització d'un WebService Remot (Escenari A i B)

Després d'haver configurat l'aplicació indicant els Web Services que volem importar, ara és el moment d'invocar-los.

Per accedir al WebService, per exemple des d'un BO, cal injectar-li el bean `webServicesService`

```
<bean id="categoryBOTarget"
     class="net.gencat.ctti.canigo.samples.prototip.model.bo.impl.CategoryBOImpl">
  <property name="dao" ref="universalHibernateDAO"/>
  <property name="logService" ref="loggingService" />
  <property name="wss" ref="webServicesService" />
</bean>
```

i, en el codi del BO, obtenir la referència al WebService a partir del nom que li hem donat al definir els `importedInterfaces` i a continuació utilitzar el WebService cridant als seus mètodes

```
HolaMonWS hm = (HolaMonWS) wss.getWebService("holaMonService");
String resultat = hm.holaMon();
```

Invocació de WebServices sense integrar amb Canigó (Escenari C)

Tal i com s'ha descrit a l'inici d'aquest apartat, pot donar-se situacions on els sistemes previstos per Canigó no s'adaptin a les necessitats de l'aplicació. En aquest cas es pot fer servir el sistema alternatiu que es consideri adequat per generar el client i consumir el servei.

Des de Canigó es recomana avaluar la generació de client XFire. Aquesta opció queda limitada a desenvolupament realitzats amb Java 5. Si es posible aquesta es la estratègia recomanada ja que no incorpora noves dependències al projecte ja que Canigó ja disposa de les llibreries de XFire.

També s'ha de considerar la possibilitat de fer servir Axis tal i com es descriu per l'escenari B enriquint el procés amb bindings propis.

Referències

Servei de WebServices de Canigó (2.3.x)

http://canigo.ctti.gencat.net/confluence/display/CAN/Servei+de+WebServices+2.3

Apache-Axis Ant Tasks

http://ws.apache.org/axis/java/ant/ant.html

Configuració de XFire amb SpringRemoting

http://xfire.codehaus.org/Spring+Remoting