**Plugin Thrift – java – wsdl generator**

**Nome plugin:** **framework-thrift2Java-maven-plugin**

**<plugin>**

**<groupId>factory.framework</groupId>**

**<artifactId>framework-thrift2Java-maven-plugin</artifactId>**

**<version>2.6.0-SNAPSHOT</version>**

**<configuration>**

**<thriftExecutable>C:/Program Files (x86)/Apache/thrift/bin/thrift</thriftExecutable>**

**<thriftSourceRoot>${project.basedir}/src/main/thrift</thriftSourceRoot>**

**<!—tag non necessario, prende questa di default:**

**<outputDirectory>${project.basedir}/src/main/java/drift</outputDirectory>-->**

**</configuration>**

**<executions>**

**<execution>**

**<!--<phase>non mettere nulla</phase>-->**

**<goals>**

**<!-- entrambi le fasi verranno seguite in sequenza se si fa mvn clean install-->**

**<goal>genjava</goal>**

**<goal>Java2Ws</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

**Per eseguire il plugin una volta indicato nel pom del progetto: mvn clean install.**

Il plugin è composto da due goal:

* **genjava (**eseguito in fase di generate-sources, cioè in pre compilazione dei sorgenti):

:legge i file thrift e genera i rispettivi file java in target/generated-sources (questi vengono poi compilati nella fase di compilazione e salvati da javac nella cartella target/classes)

* **Java2Ws**: (eseguito dopo la fase di compile, ovvero dopo che le classi java sono state compilate in file class in quanto è presente una fase di modifica del bytecode, anziché che dei sorgienti; il bytecode modificato (“ricompilato”) con la libreria byteBuddy si trova nella stessa posizione sotto target/classes; dopo viene eseguito il java2ws di Apache CXF, richiamando il suo plugin, si generano così i wsdl delle interfacce presenti nelle classi appena modificate).

Il plugin vuole in input principalmente **thriftExecutable**, che indica il percorso dove è presente il file eseguibile di thrift, in tale percorso deve essere ance presente la libreria jar.

I file thrift di default sono nel progetto /src/main/thrift (se non si passa < **thriftSourceRoot>**, altrimenti si può passare un altro path.

**La versione della libreria Thrift è la 0.11.0**

**NB:** Il plugin richiama a sua volta il pugin del thrift, il cui sorgente va compilato e va installato nel repository. Il plugin è quello ufficiale di Apache ed è scaricabile su github e compilabile con il comando

mvn clean install skipTest=true

si trova al seguente indirizzo:

[**https://github.com/apache/thrift/tree/master/contrib/thrift-maven-plugin**](https://github.com/apache/thrift/tree/master/contrib/thrift-maven-plugin)

Il comando clean install serve solo ad installare nel proprio repository locale .m2 la libreria jar, che va poi quindi deploiata su Nexus.

Per quanto riguarda la generazione dei Wsdl, come indicato precedentemente, prima di passare il riferimento alle interfacce e il classpath dei file compilati ad Apache cxf , il plugin le modifica tramite la libreria byteBuddy(<http://bytebuddy.net>) aggiungendo annotazioni e applicando le seguenti regole:

1. Per le prorpietà di tipo java.util.List :

Le annotazioni inserite sulle liste sono

* XmlWrapper con attributo name
* XmlElement con attributo name
* JsonProperty con attributo value

Ad esempio il campo TAZIGENLIST verrà seguito dalla keyword del wax “List”

Perché si segue la regola <nome\_proprietà>List

Mentre XmlElement contiene solo il <nome\_proprietà> (stessa cosa per JsonProperty):

**@XmlElementWrapper(name="TAZIGENLISTList",** namespace="##default", required=**false**, nillable=**false**)

@XmlElement(name="TAZIGENLIST", type=XmlElement.DEFAULT.**class**, defaultValue="\000", namespace="##default", required=**false**, nillable=**false**)

@JsonProperty(index=-1, value="TAZIGENLISTList", access=JsonProperty.Access.AUTO, defaultValue="", required=**false**)

**public** List<tTAZIGEN> TAZIGENLIST;

\*I valori non in giallo sono quelli di default che il plugin non modifica.

1. Qualsiasi Classe che non sia di tipo Interfaccia o un enum applica le seguente annotazioni :
   * *XmlAccessorType con value = XmlAccessType.FIELD*

La classe che contiene l’interfaccia Iface è la classe che espone il servizio web, il cui metodo esposto è dichiarato all’interno dell’interfaccia stessa (su questa non applichiamo @WebService).

@XmlAccessorType(XmlAccessType.FIELD)

**public** **class** sigs534

{

**public** **static** **abstract** **interface** Iface

  {

**public** Output metodWebService (Input in ) ;

}

}

I valori non in giallo sono quelli di default che io non ho modificato.

Dopo che il plugin aggiunge le annotazioni mancanti viene invocato il plugin **cxf-java2ws-plugin** di Apache cxf (che esegue il goal **java2ws)**:

<**plugin>**

**<groupId>org.apache.cxf</groupId>**

**<artifactId>cxf-java2ws-plugin</artifactId>**

**<version>${cxf.version}</version>**

**<dependencies>**

**<dependency>**

**<groupId>org.apache.cxf</groupId>**

**<artifactId>cxf-rt-frontend-jaxws</artifactId>**

**<version>3.1.15</version>**

**</dependency>**

**<dependency>**

**<groupId>org.apache.cxf</groupId>**

**<artifactId>cxf-rt-frontend-simple</artifactId>**

**<version>${cxf.version}</version>**

**</dependency>**

**</dependencies>**

**<executions>**

**<execution>**

**<id>process-classes</id>**

**<phase>process-classes</phase>**

**<configuration>**

**<className>nome class in canonical form </className>**

**<genWsdl>true</genWsdl>**

**<verbose>true</verbose>**

**</configuration>**

**<goals>**

**<goal>java2ws</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

Al plugin CXF viene passato anche il riferimento alla dipendenza alla libthrift perché le classi e le interfacce generate dipendono dalle classi contenute all’interno di quella libreria; inoltre viene passato il percorso di output <outputFile> indicante la posizione in cui verrà generato il wsdl: essendo il plugin eseguito all’interno del classpath ${project.base.build} i wsdl verranno generati all’interno della cartella target/classes e non nella src/ del progetto. Quindi il plugin poi li sposterà copiandoli in una fase successiva all’interno della cartella src/main/resources.

La copia dei file in src/resources è dovuta anche perché è durante la fase di “packaging” (creazione del jar/war o dell’artefatto del progetto) successiva alla fase di processamento delle classi (fase in cui questa goal è eseguito), il maven copia la cartella wsdl che troverà nella cartella src/resources e non quella in target/classes.

Il file **src/main/resources/META-INF/drift-service.list** conterrà la lista dei servizi che dovrà essere così composta, insieme ad altre convenzioni:

contenuto file drift-service.list (in questo caso un servizio, se più servizi, uno per riga)

[***META-INF/wsdl/*drift-thrift@**<nome\_classe>Service.wsdl](mailto:META-INF/wsdl/drift-thrift@%3cnome_classe%3eService.wsdl)

“drift-thrift” è il <nome del modulo>, ogni namespace deve iniziare con drift,

quindi in ogni file .thrift il namespace sarà

***namespace java drift.drift.thrift.api.nome\_classe***

la classe sarà dichiarata come :

*Service* ***nome\_classeService*** *{..}*

Riassumendo, in generated-sources il nostro servizio si troverà nella seguente alberatura:

***- generated-sources***

***- drift***

***- drift***

***- thrift***

***- api***

***-nome\_classe (profondità package)***

***\* nome\_classeService-.java\****

**(stessa cosa in target/classes)**

Il wsdl generato dopo il comando mvn clean install si troverà in

[**/src/main/resources/META-INF/wsdl/drift-thrift@nome\_classeService.wsdl**](mailto:/src/main/resources/META-INF/wsdl/drift-thrift@nome_classeService.wsdl)

**Più in dettaglio il nome file così composto:**

[**modulo@<nome\_classe>Service.wsdl**](mailto:modulo@%3cnome_classe%3eService.wsdl) **dove la parte variabile è il nome classe.**

La dipendenza per eseguire i plugin richiamati all’interno del plugin è questa:

**<dependency>**

**<groupId>org.twdata.maven</groupId>**

**<artifactId>mojo-executor</artifactId>**

**<version>2.3.0</version>**

**</dependency>**

[**https://github.com/TimMoore/mojo-executor**](https://github.com/TimMoore/mojo-executor)

**La versione di Byte Buddy:**

**<dependency>**

**<groupId>net.bytebuddy</groupId>**

**<artifactId>byte-buddy</artifactId>**

**<version>1.4.26</version>**

**</dependency>**

Altre dipendenze incluse nel plugin:

**<dependency>**

**<groupId>org.apache.maven</groupId>**

**<artifactId>maven-plugin-api</artifactId>**

**<version>3.5.3</version>**

**</dependency>**

**<dependency>**

**<groupId>org.apache.maven.plugin-tools</groupId>**

**<artifactId>maven-plugin-annotations</artifactId>**

**<version>3.2</version>**

**</dependency>**

**<dependency>**

**<groupId>org.apache.thrift</groupId>**

**<artifactId>libthrift</artifactId>**

**<version>0.11.0</version>**

**</dependency>**

**<dependency> <!—per il JsonProperty 🡪**

**<groupId>com.fasterxml.jackson.core</groupId>**

**<artifactId>jackson-annotations</artifactId>**

**<version>2.8.8</version>**

**</dependency>**