**AFCEPF – Cours du 29 octobre 2018 .**

Table des matières

[AFCEPF – Services web - Cours du 29 octobre 2018 – D. Defrance 2](#_Toc528595295)

[Présentation de base des web-services 2](#_Toc528595296)

[RMI – page 113 du pdf EJB3\_JPA\_RMI\_JMS\_CDI\_jboss7\_et\_wildFly9 3](#_Toc528595297)

[TP sur les RMI (remote method invocation) 3](#_Toc528595298)

[Projet avec EJB 6](#_Toc528595299)

[WSDL : WebService Description Language 10](#_Toc528595300)

[TP servEjb 10](#_Toc528595301)

# AFCEPF – Services web - Cours du 29 octobre 2018 – D. Defrance

## Présentation de base des web-services

SOAP : passe par XML

REST : passe par JSon

Il peut y avoir une application « cœur de métier » qui blablate avec d’autres applications, que l’entreprise va se procurer directement sur le marché (pas personnalisée),  
Le format des données peut varier, on utilise alors des adaptateurs (JSon)

On doit modéliser : UML (Unified Modeling Language) et **BPML**, ce dernier est beaucoup plus proche des webservices  
BPM : Business Process Management.  
SOA : Service Oriented Application

Un service est une URL programmée pour lancer un process, que les applications appellent pour obtenir le service.

Les classes java DTO permettent de véhiculer les données entre les serveurs d’application.  
DTO : Data Transfert Object.

Existence d’un serveur intermédiaire parfois, justifié par le côté sécurités ou le côté statistique.

CXF : technologie Java permettant de programmer de ws REST et SOAP.  
On peut aussi utiliser Spring MVC.

SOAP : les requêtes sont encodées en XML et reviennent en XML  
WSDL : Web Service Description Language

REST : utilisent http au maximum, pas de format de données (moins typé que SOAD du coup).

Les services REST sont plus simples et efficaces, ce qui explique le déclin du SOAP en faveur du REST.

SOA récents :

* Client SPA : Single Page Application 🡪 comme Angular, appelle un ws REST.
* WOA : Web Oriented Architecture
* Architecture « micro-service »

## RMI – page 113 du pdf EJB3\_JPA\_RMI\_JMS\_CDI\_jboss7\_et\_wildFly9

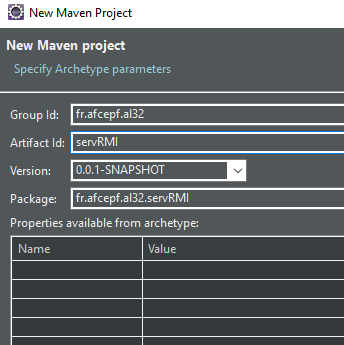
RMI : Remote Method Invocation

On va lancer un serveur de noms,

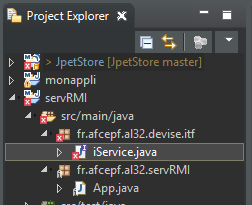
Exception technique : java.rmi.remoteException  
On peut avoir un cable mal branché par ex.

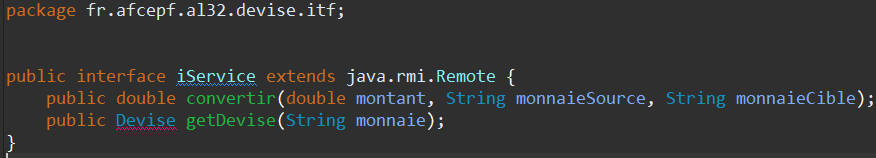
On doit implémenter dans les interfaces : remote et satellisable

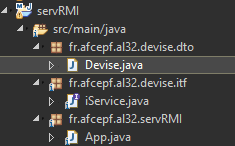
## TP sur les RMI (remote method invocation)



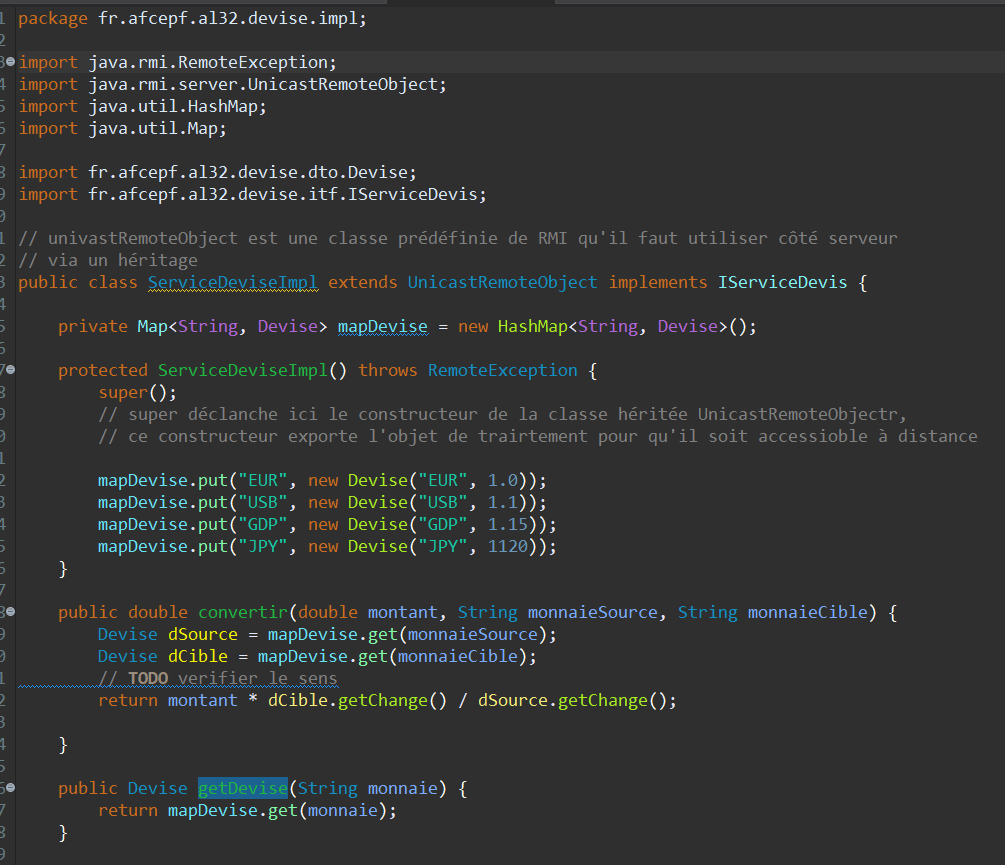
On configure le pom.xml

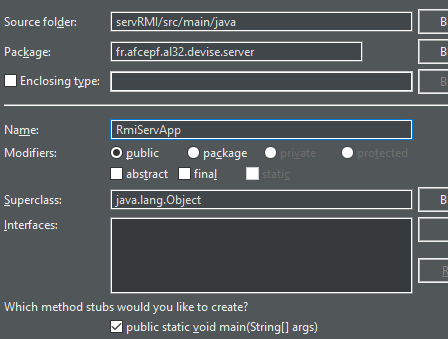
On ajoute un package pour les interfaces, et on y crée notre première interface qui extends l'interface java.rmi.remote :  




Création de la classe Devise dans un autre package :  


Implémentation :



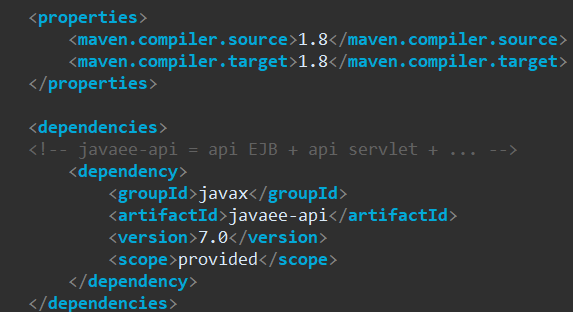
Création ensuite dans un package fr.afcepf.al32.devise.server d'un main  


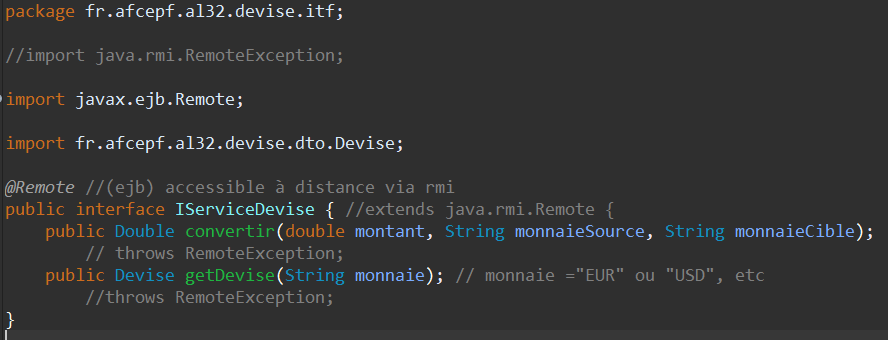
Il manquait dans notre code des throws exception etc.

TP dispo ainsi que sa solution.

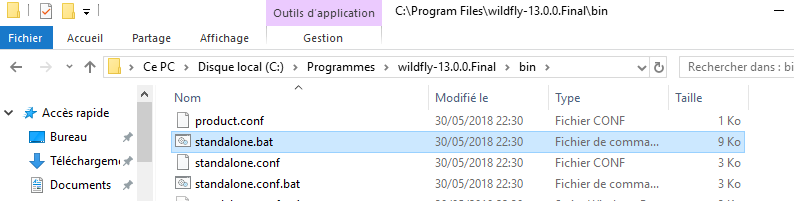
## Projet avec EJB

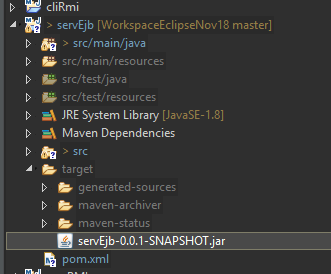
Dans le pom.xml on ajoute toujours le compiler / target en 1.8, mais on ajoute l'ejb

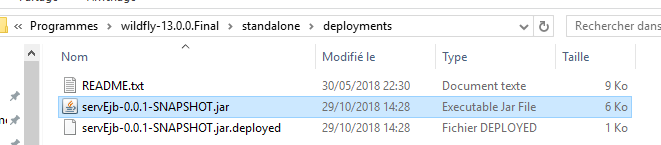


Dans iServiceDevise récupéré du projet précédent, on rajoute @Remote et on enlève les throws  


Idem dans l'implémentation :  

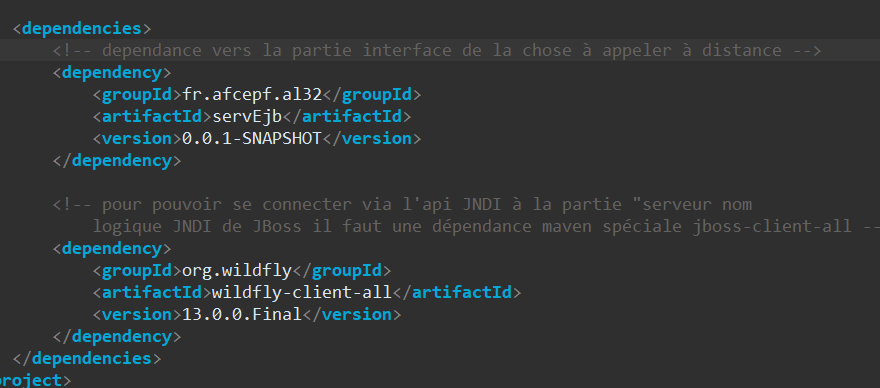

On installe Wildfly  


On peut ensuite tester que tout est ok,  
On run le standalone.bat (ci-dessus)  
On récupère le .jar compilé par Eclipse (dans les target)  


On le copie-colle dans le dossier pathWildly 🡪 standalone 🡪 deployments  


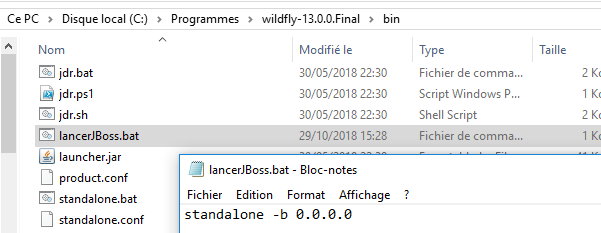
Le coller là après avoir lancé le standalone.bat (donc le serveur) permet de direct vérifier qu'on a bien géré.

Création d'un autre projet maven, son pom :



Voici un p'tit bayou utile :  


On peut créer un .bat spécial pour lancer jboss avec moins de sécurité



Payload : charge utile, comme le DTO (data transfert object), c'est ce qui est facilement transportable via le réseau

## WSDL : WebService Description Language

On n'a pas forcément le même langage du côté client et serveur, les types de données ne sont pas forcément non plus les mêmes (mêmes noms),  
Il faut se mettre d'accord au niveau des types compatibles ;  
WSDL est une norme reconnue.  
C'est une sorte de contrat entre l'appelant et l'appelé.

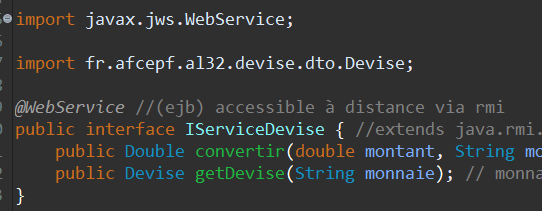
EndPoint : reçoit une requête SOAP sur une URL, transforme en java, et renvoie la réponse en SOAP (XML).

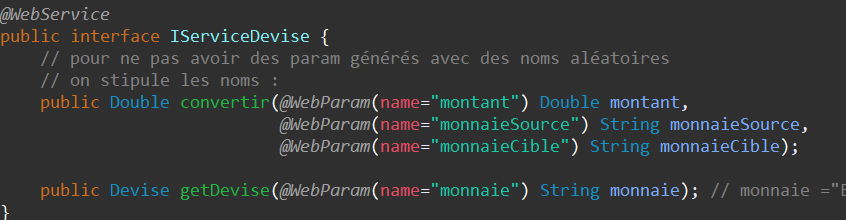
Même principe qu'avec RMI, sauf que communique au milieu en java.

SEI : Service Endpoint Interface

On crée un nouveau projet

## TP servEjb



Utilisation de @webParam(name="nomParam") :  


@WebService(targetNameSpace="", endPointInterface="")