# Installation de Thrift :

Plusieurs choses à faire :

Fichier IDL :

Il faut définir le fichier IDL (Interface Definition Language ) qui servira d’interface entre client et serveur . c’est grace à ce fichier que l’ executable Thirft créera les sources des 2 cotés .

ce fichier est un fichier \*.Thirft , défini par un langage propre à lui mais très clair et bien structuré .

On peut déclarer des constantes , des structures , organiser le code grâce à des namespaces , des includes ; des commentaires peuvent êtres rajoutés ….

Les méthodes que l’on pourra utiliser seront déclarés dans le block Service :

Ex : service Twitter {

*// A method definition looks like C code. It has a return type, arguments,*

*// and optionally a list of exceptions that it may throw. Note that argument*

*// lists and exception list are specified using the exact same syntax as*

*// field lists in structs.*

**void** ping(), *//* *Confusingly, method definitions can be terminated using comma or semi-colon*

bool postTweet(1:Tweet tweet); *//* *Arguments can be primitive types or structs*

TweetSearchResult searchTweets(1:string query); *//* *Likewise for return types*

*// The 'oneway' modifier indicates that the client only makes a request and*

*// does not wait for any response at all. Oneway methods MUST be void.*

oneway **void** zip() *//* *void is a valid return type for functions*

}

on peut trouver un tutoriel ici . <http://diwakergupta.github.io/thrift-missing-guide/>

fichier exe :

Sous windows , c’est ce fichier qui génèrera les sources coté client et coté » serveurs

La sécurité internet Continental empeche de télécharger sur la page Apache . mais autorise d’aller dans les archives . il faudra suivre les mises à jour .

<http://archive.apache.org/dist/thrift/0.9.0/>

il suffit ensuite de lancer une commande DOS pour générer , grâce au fichier Thrift , les sources pour différents langages : toutes les commandes sont accessibles avec l’option -help

celles qui nous interressent:

Usage: thrift [options] file

Options:

-o dir Set the output directory for gen-\* packages

(default: current directory)

--gen STR Generate code with a dynamically-registered generator.

STR has the form language[:key1=val1[,key2,[key3=val3]]].

Keys and values are options passed to the generator.

Many options will not require values.

Available generators (and options):

**java (Java):**

beans: Members will be private, and setter methods will return void.

private-members: Members will be private, but setter methods will return

**py (Python):**

new\_style: Generate new-style classes.

twisted: Generate Twisted-friendly RPC services.

utf8strings: Encode/decode strings using utf8 in the generated code.

slots: Generate code using slots for instance members.

dynamic: Generate dynamic code, less code generated but slower.

dynbase=CLS Derive generated classes from class CLS instead of TBase.

dynexc=CLS Derive generated exceptions from CLS instead of

Dépendances :

Chaque langage a besoin d’un « noyau de code » et parfois de dépendances :

Le noyau est accessible sur github (ici branche master )

<https://github.com/apache/thrift/tree/master/lib>

pour les dépendances . il faut les trouver sur le net . (testé pour java , pour python pas de dépendances )

une liste de requirement est accessible ici :

<http://wiki.apache.org/thrift/ThriftRequirements>

# Utilisation de Thrift :

Une fois le code généré coté client et serveur :

Il faut implémenter les méthodes définissant le service , coté serveur :

Si l’on a définit le service XXX , il suffit d’hériter de la partie IFace (interface ) qui à été crée et

d’implémenter les méthodes

Puis une partie lancement de serveur doit etre lancée

Ex python :

Class XXXService (XXX.IFace)

{

Def methode1(self) :

….

Def methode2 (self, toto ) :

….

}

handler = XXXService()

processor = Processor(handler)

transport = TSocket.TServerSocket(port=3333)

tfactory = TTransport.TFramedTransportFactory()

pfactory = TBinaryProtocol.TBinaryProtocolFactory()

server = TServer.TSimpleServer(processor, transport, tfactory, pfactory)

Coté client , on utilise la partie Client du service .

Ex java :

TSocket socket = new TSocket("[::1]", 3333);

TFramedTransport transport = new TFramedTransport(socket);

TProtocol protocol = new TBinaryProtocol(transport);

XXX.Client xxx = new dSpaceManager.Client(protocol);

xxx.methode1();

xxx.methode2();

# Problèmes trouvés :

la génération du python est formatée en > 2.7 il faut modifier toutes les exceptions : “totoException as e” en totoException , e

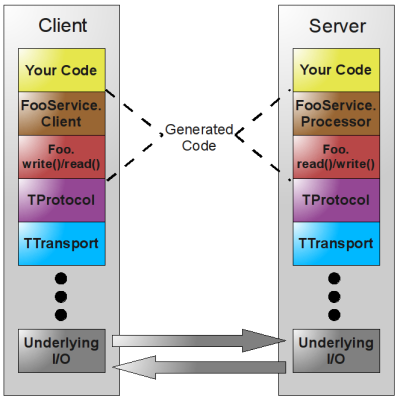
les tabulations ne sont pas au format PEP8...

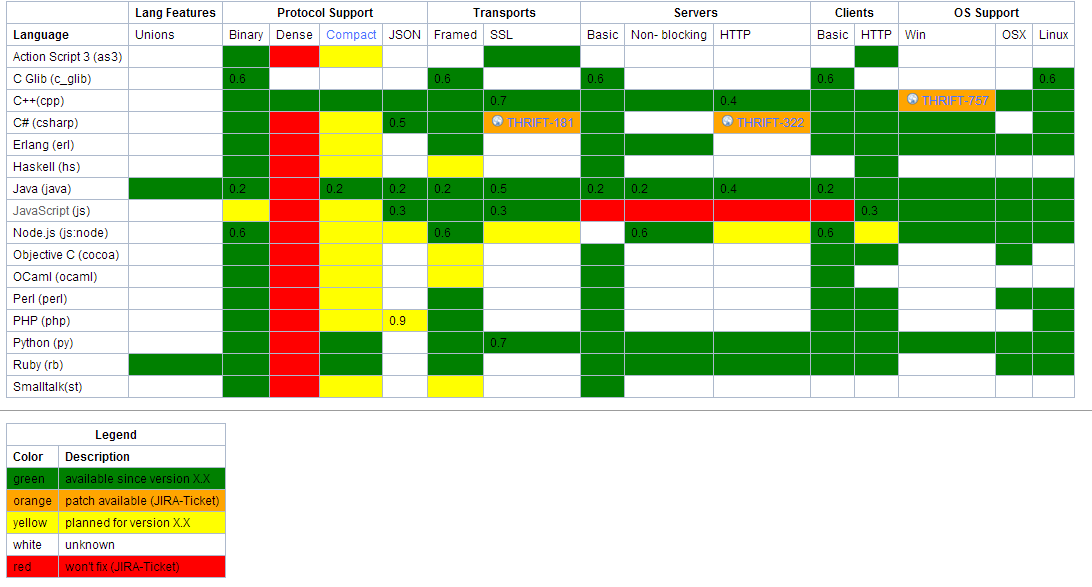
java :

pour connecter les server avec TSimpleServer .

il faut utiliser l’adresse IPV6 du localhost :“[:1]”

# refs :





<http://thrift.apache.org/>

<http://wiki.apache.org/thrift/FrontPage>

<http://diwakergupta.github.io/thrift-missing-guide/>

<http://www.thrift.pl/Thrift-tutorial-getting-started.html>

<http://jnb.ociweb.com/jnb/jnbJun2009.html>

<http://www.codeproject.com/Articles/14740/Fast-IPC-Communication-Using-Shared-Memory-and-Int>