Java inside - lab 3 : **Invocation API java.lang.invoke**

SORAN Altan
Repository github: https://github.com/asoran/java-inside
Lab 3 : Lookup, MethodHandle

Reflection et Invocation

On a vu que l'on pouvait accéder aux champs et méthodes des classes par reflection avec l'API java.lang.reflect. On va voir que l'on peut aussi le fait avec l'API d'Invocation, et notamment grâce aux classes MethodHandles et Lookup du package java.lang.invoke .

MethodHandler

Cette classe représente un méthode handler, c'est à dire, un pointeur typé vers une méthode.

On peut appeler la méthode pointé avec les méthodes invoke() et invokeExact().

Pour créer un méthodeHandler, il faut passer par un lookup.

insertArgument(

Lookup

Lookup est un objet de recherche, qui permet de recherche des méthodes dans une classe.

Par défaut, quand on crée un lookup, il référencie la classe courante et seulement les méthodes publiques. Pour crée un lookup d'un autre classe, il faut en créer un et le "téléporter" dans la classe cible avec MethodHandles.teleport … ou pas, mais avec MethodHandles.privateLookupIn(class, lookup) (qui fournit un lookup pour voir les méthodes privées).

Lookup::findStatic permet de récupérer les méthodes statiques, tandis que Lookup::findVirtual, les méthodes d'instance

MethodHandles

C’est un classe d’utilitaire pour gérer les lookup et les MethodHandler.

Il propose des méthodes pour notamment :

- Ajouter et enlever dynamiquement des paramètres des MethodHandler
- Créer des Lookup
- Créer des MethodHandler de toute pièce

Et d’autre choses encore ...

MethodHandles.constant et MethodHandles.guardWithTest

Il existe deux fonctions intéressantes:

MethodHandles.constant, qui permet de créer une méthode qui renvoie tout le temps la même chose (une constante quoi)

Et MethodHandles.guardWithTest, qui permet de simuler un if_else sous forme de méthode. On passe à la méthode 3 MethodeHandler, un pour exécuter le test du “if”, une pour le cas quand le test réussit, et une pour quand le test fail.

MethodTypes

Cette classe est à utiliser avec les MethodHandler, elle fournit des méthodes pour créer des signature de fonctions:

MethodType.methodType(<returnType.class>, <parameter1.class>, …, <parameterN.class>)

MethodType.methodType(String.class, String.class) représente donc une fonction qui prend en paramètre une String et retourn une String