L'API Reflexion

Approfondissement de Java

X.BLANC & J. DANIEL

Une API réflexive !

- Java permet d'obtenir des informations de descriptions sur les classes, les opérations, les types, etc…..
- Grâce à la réflexion, on peut analyser dynamiquement un classe, mais également l'utiliser.
- Offre une grande souplesse d'utilisation des applications java telle que la mobilité du code.

Obtenir la description d'une classe Java

- Une classe est symbolisée par la classe « ***java.lang.Class*** »
- Chaque instance de classe peut retourner un descripteur de sa classe par l'intermédiaire de l'opération « ***getClass*** »

```
maClasse m = new maClasse();

java.lang.Class cl = m.getClass();
```

- Chaque classe comporte également un attribut appelé « ***class*** » pour récupérer sa description :

```
java.lang.Class cl = maClass.class;
```

Obtention dynamique d'une description !

- A partir du nom d'une classe, on peut récupérer dynamiquement sa description en utilisant l'opération statique « ***forName*** » :

```
java.lang.Class cl = java.lang.Class.forName("exemple.maClasse");
```

- Attention, cette opération retourne l'exception « ***ClassNotFoundException*** »

La classe « java.lang.Class »

- La classe « Class » retourne toutes les informations permettant de connaître la description d'une classe :
  - son nom (***getName***),
  - son package (***getPackage***) ,
  - ses constructeurs (***getConstructors***),
  - ses méthodes (***getMethods***),
  - ses attributs (***getFields***),
  - son ancêtre (***getSuperclass***) ,
  - les interfaces qu'elle implante (***getInterfaces***).

Description d'un Package

- La description d'un package est offert par la classe « ***java.lang.Package*** » au moyen de l'opération « getPackage »

```
java.lang.Package getPackage();
```

- Cette classe propose les opérations suivantes :
  - String getName() : retourne le nom du package
  - Package getPackage() : retourne le package de ce package
  - Package [] getPackages() : retourne la liste des packages de ce package.

Description des constructeurs

- Pour décrire un constructeur, on utilise classe « ***java.lang.reflect.Constructor*** ». Les descriptions de constructeurs sont retournés par « getConstructors » :

```
java.lang.reflect.Constructor [ ] getConstructors();
```

- Les opérations de « java.lang.reflect.Constructor » sont :
  - String getName() : retourne le nom du constructeur
  - int getModifiers() : retourne les modificateurs qui s'appliquent au constructeur
  - java.lang.Class getExceptionType() : retourne les descriptions des exceptions lancées par le constructeur

Les modificateurs

- La classe « java.lang.reflect.Modifier » permet la manipulation des modificateurs retournés par les descriptions.

- Cette classe propose un ensemble d'opérations permettant de tester la validité d'un modificateur (***final, abstract, public, private, protected, static, transient, synchronized***). Ces opérations respectent la syntaxe suivante :

```
public static boolean isXXXXX( int modifier )
```

Description des méthodes

- Afin de décrire une méthode, la classe « ***java.lang.reflect.Method*** » est utilisée. Les descriptions de méthodes sont retournées par « getMethods »

```
java.lang.reflect.Method [ ] getMethods( );
```

- Les principales opérations de cette classe sont :
  - String getName() : retourne le nom de la méthode
  - int getModifier() : retourne les modificateurs de la méthodes
  - Class [] getExceptionType() : la liste des exceptions
  - Class [] getParamType() : la liste des paramètres
  - Class getReturnType() : retourne le type de retour de la méthode

Description des attributs

- Pour décrire un attribut, la classe « ***java.lang.reflect.Field*** » est utilisée.

```
Java.lang.reflect.Field [ ] getFields();
```

- Les principales opérations sont :
  - String getName() : nom de l'attribut
  - int getModifier() : modificateur de l'attribut
  - Class getType() : type de l'attribut

- Pour savoir, si un type est un type « primitif », on utilise l'opération « ***isPrimitive*** ».

Description des interfaces

- Afin de décrire une interface, on emploi la classe « java.lang.Class » !

```
java.lang.Class [ ] getInterfaces();
```

- Pour savoir si une classe « Class » décrie une interface, on utilise l'opération « ***isInterface*** ».

Description des tableaux

- Une classe « Class » peut décrire un tableau. Pour savoir, si c'est un tableau on utilise l'opération « ***isArray*** »

- Pour connaître le type du tableau, on emploi l'opération « ***getComponentType*** »

```
if ( cl.isArray() )
{
        java.lang.Class type = cl.getComponentType();
}
```

Création dynamique d'une instance de classe !

- On peut créer dynamiquement une instance de classe à l'aide de l'opération « ***newInstance*** » :

```
try
{
    java.lang.Class clz = java.lang.Class.forName("maClasse");

    java.lang.Object obj = clz.newInstance();

    maClasse maC = ( maClasse ) obj;
}
catch ( java.lang.Exception ex )
{
    ex.printStackTrace( );
}
```

La classe à instancier doit impérativement comporter un constructeur sans paramètre.

Utilisation dynamique de constructeurs

- Si une classe comporte un constructeur avec paramètres, on doit utiliser l'opération « ***newInstance*** » de la classe « Constructor » afin d'en créer une instance :

```
java.lang.Object newInstance ( java.lang.Object [ ] params )
```

Utilisation dynamique d'une classe !

- L'API reflexion permet une utilisation dynamique des classes :
  - auto-description,
  - instanciation à la demande.

- Il est impératif de pouvoir effectuer des invocations dynamiques sans utiliser directement le type de la classe cible.

```
java.lang.Class clz = java.lang.Class.forName("maClasse");

java.lang.Object obj = clz.newInstance();

maClasse maC = ( maClasse ) obj;
```

Appel dynamique d'une opération

- Pour appeler dynamiquement une opération, on emploi « ***invoke*** » :

  java.lang.Object invoke (java.lang.Object cible, java.lang.Object [] params)

- Si une exception se produit lors de l'invocation, l'opération « invoke » génère l'exception « ***java.lang.reflect.InvocationTargetException*** »

- Parmi les opérations de cette exception, on a :
  - void printStackTrace()
  - Throwable getTargetException()

Utilisation dynamique d'un attribut

- La classe « **Field** » comporte plusieurs opérations permettant l'accès aux valeurs des attributs.

- Ainsi, les opérations suivantes sont définies pour les types primitifs :

```
void setXXXX( java.lang.Object cible, XXXXX value );
XXXX getXXXX( java.lang.Object cible );
```

- Pour les types complexes on a:

```
void set( java.lang.Object cible, java.lang.Object value );
java.lang.Object get( java.lang.Object cible );
```

Mise à l'épreuve...

- *Développez une méthode qui décrie une classe :*
  - *affiche ses constructeurs*
  - *affiche ses méthodes*
  - *affiche ses attributs.*