

L'analyseur syntaxique de fonctions

(Révision du 17 Mars 2016 pour la version 3.1)

Que fait dans les grandes lignes ce package

Le package

rr.modulog.syntaxe

permet de transformer une chaîne de caractères, qui, si elle est correctement écrite, représente une fonction mathématiques d'une ou plusieurs variables en un objet de la classe Fonction. Cet objet peut s'évaluer si on donne des valeurs aux variables. On peut afficher ses diverses caractéristiques : son nom, son arbre syntaxique, son expression algébrique sous forme d'une chaîne de caractères, son numéro dans la table dynamique des fonctions connues, le nombre de variables, son statut qui détermine si elle est utilisée par une autre fonction ou non.

Un objet de la classe Fonction une fois créé est automatiquement inséré dans une table. Cette fonction peut alors être appelée par une autre fonction. Elle peut être enlevée de la table pourvu qu'aucune autre fonction encore en vie ne fasse appel à elle. Bien entendu, comme sous-produits, on trouve diverses fonctionnalités de moindre importance.

Qu'est ce que l'écriture correcte d'une fonction

Nous définissons ici la syntaxe des chaînes de caractères qu'il conviendra de donner en pâture au constructeur de la classe Fonction.

Une chaîne correcte sera par exemple de la forme

$$f(x,y,z)=0.1*x^{\sin(x*y)}+g(z)-0.5e-3$$

où g est le nom d'une fonction d'une variable déjà entrée dans la table.

Les variables x,y,z dans notre exemple sont muettes, c'est-à-dire qu'il aurait été tout à fait équivalent de définir la fonction

$$f(u,v,t)=0.1*u^{\sin(u*v)}+g(t)-0.5e-3$$

En interne les variables sont juste numérotées, sans nom spécifique.

Voici la grammaire utilisée.

```
ECRITURE      ::= 'NomDeFonction'('('SUITEDEVAR.')' '=' EXP1
SUITEDEVAR    ::= SUITEDEVAR / Vide
SUITEDEVAR    ::= 'Variable'.STVAR
STVAR         ::= ','SUITEDEVAR / Vide
```

EXP1 ::= EXPRESSION.'\$'
 EXPRESSION ::= '+' .TERM.ST / '-' .TERM.ST / TERM.ST
 TERM ::= FACTEUR.SF
 ST ::= '+' .TERM.ST / '-' .TERM.ST / Vide
 FACTEUR ::= FACT.SFACT
 SF ::= '*' .FACTEUR.SF / '/' .FACTEUR.SF / Vide
 FACT ::= F / E / 'Nombre' / 'Variable'
 SFACT ::= '^' .FACTEUR / Vide
 F ::= 'NomDeFonction' . '(' .EXPRESSFONCT . ')'
 E ::= '(' .EXPRESSION . ')'
 EXPRESSFONCT ::= NV / Vide
 NV ::= EXPRESSION.SUITEXPRESS
 SUITEXPRESS ::= ',' .NV / Vide

Du point de vue lexical, nous devons définir ce qu'est un **nom de fonction** correct, un **nom de variable** correct, un **nom de constante** correct, les **écritures des entiers** et des **nombres réels**, les **opérateurs connus**.

Première remarque : il n'y a pas de différence entre majuscule et minuscule, avant l'analyse, la chaîne est transformée en majuscules! Donc x et X représentent le même identificateur.

Un nom de fonction est une chaîne constituée de caractères compris entre 'A' et 'Z' ou entre '0' et '9' et commençant par une lettre. Les blancs sont éliminés avant l'analyse (la chaîne est compactée avant toute chose).

C'est ce qu'on attend en premier : MaFonction(x)=

Il y a un certain nombre de fonctions prédéfinies :

ABS
 ACH
 ACOS
 ASH
 ASIN
 ATAN
 ATH
 CH
 COS
 EXP
 FRAC
 INT
 LOG
 LN
 SH
 SIN
 SQR

SQRT
TG
TH
CONJ
RE
IM
ARG
MAX
MIN

Un nom de variable est constitué d'un **seul caractère** compris entre 'A' et 'Z' sauf la lettre E. Quand on écrit $f(u,v)=u*v$, le nom de fonction est f, la première variable s'appelle u (pour le temps de l'analyse), la deuxième variable s'appelle v, après le signe = on ne doit pas trouver une variable autre que ces deux là : $f(u,v)=u*v+w$ provoque une erreur.

Un nom de constante est constitué comme un nom de fonction. Les noms de constantes se trouvent dans une table. Dans le package d'origine il n'y a que deux noms de constantes : E et PI dont les significations vont de soi.

Les nombres sont de la forme 123, +123, -123, 24.787, -24.2, +23.98, 45.321e-7, 56.78e+12, 56.78e12.

Opérateurs connus +, - (unaires et binaires), *, / (binaires), ^ (binaire), la virgule pour séparer les variables.

Organisation globale du package

Le package s'appelle **rr.modulog.syntaxe** donc pour l'installer, il suffit de le décompresser-détarer (**tar xvfz RRSyntaxe-V3.1.tgz**). Cela crée un répertoire **RRSyntaxe-V3.1** qu'on peut renommer si on veut et qu'il faut citer dans la variable d'environnement java CLASSPATH. Tout s'installe alors dans la hiérarchie de répertoires RRSyntaxe-V3.1/rr/modulog/syntaxe/

La classe publique principale est la classe Fonction. Lors de la première utilisation un code d'initialisation s'exécute (une seule fois). Ce code permet de mettre en place la table des constantes prédéfinies, la table dynamique des fonctions qui contient dès le départ les fonctions prédéfinies, et qui s'enrichit des nouvelles fonctions au fur et à mesure qu'on crée les objets correspondants.

Voici les objets publics et leurs méthodes publiques de cette classe :

```
//Constructeur  
public Fonction(String S)  
throws FonctionException
```

```
//Méthodes de classe
public static Fonction getFonction(short i)

//Méthodes d'instance
public NomDeFonction getNomDeFonction()
public ArbreDeFonction getArbreDeFonction()
public ExpressionDeFonction getExpressionDeFonction()
public short getNumeroDeFonction()
public byte getNombreDeVariables()
public short getEnusage()
public void libereFonction()
    throws FonctionException
public double EvalFonct(double[] v)
    throws Exception
public double EvalFonct(double x)
    throws Exception
public double EvalFonct(double x,double y)
    throws Exception
public double EvalFonct(double x,double y,double z)
    throws Exception
```

Commentaires : le constructeur construit une instance de la classe Fonction à partir d'une chaîne correcte. Le lecteur attentif verra que ce constructeur peut lancer une exception dont la classe est commentée plus loin.

On voit qu'on peut récupérer divers renseignements.

Le nom de la fonction.

L'arbre de la fonction (arbre de la partie de l'analyse qui commence après le signe égale).

L'expression de la fonction (sous forme de chaîne) de caractères. Exemple :

« $f(x,y)=\sin(x*y)$ ».

Le numéro de la fonction (sa place dans la table).

Le nombre de variables de la fonction .

Le nombre d'appels à cette fonction par d'autres fonctions qui l'utilisent.

Attention, ceci ne compte pas le nombre de fonctions qui l'utilisent, car une même fonction peut l'utiliser plusieurs fois.

On peut en outre libérer la place occupée par une fonction dans la table pourvu que la fonction ne soit pas utilisée par une autre (on peut tester avec getEnusage, ou en traitant les erreurs).

Enfin, on peut l'évaluer sur un tableau de valeurs de la taille du nombre de variables. Cependant comme souvent on utilisera des fonctions de une, deux ou trois variables,

des évaluations spécifiques existent pour ces cas.

Les divers champs constituant un objet fonction sont privés, et donc ne sont accessibles qu'à travers les méthodes chargées de les récupérer.

La classe FonctionException regroupe toutes les exceptions générées par les méthodes de la classe Fonction. Voici cette classe en détail :

```
public class FonctionException extends Exception {

    private static String[] tabledesmessages = {
        "Expression correcte",
        "Il manque une parenthèse ouvrante",
        "Il manque une parenthèse fermante",
        "Un nom de variable est incorrect",
        "Il y a des caractères en trop",
        "La chaîne d'entrée est vide",
        "Il y a un nombre non valide",
        "Le nombre de variables ne correspond pas à la définition",
        "Le nom de la fonction existe déjà dans la table",
        "Il manque un signe = ",
        "Un nom de variable est déjà utilisé",
        "Un nom de variable est incorrect",
        "Le nom de la fonction est incorrect",
        "La table de fonctions est pleine",
        "La fonction est utilisée"
    };

    public FonctionException(int numero) {
        super(tabledesmessages[numero]);
    }
}
```

Les outils moins utiles

Voici de manière moins détaillées, les autres classes qui possèdent des objets publics intéressants. En principe, sauf pour des essais ou des développements ultérieurs, on n'a pas besoin de manipuler directement ces objets.

```
*****
*** La classe ArbreDeFonction ***
*****
```

```
//Champs
public byte      opertype;      // type de l'operateur (0..6)
public char      signe;         // Si type=0,4
```

```

public short      fonctcode;      // Si type=1
public short      varcode;        // Si type=2
public double     constante;      // Si type=3
public char       separateur;     // Si type=5
public short      constform;      // Si type=6
public ArbreDeFonction oper1;     // Si type=0,1,4,5
public ArbreDeFonction oper2;     // Si type=0,5

```

//Constructeurs

```

public ArbreDeFonction()
public ArbreDeFonction(byte b, char c, short fc, short vc, double ct, char sep,
                        short ctf, ArbreDeFonction P1, ArbreDeFonction P2)

```

//Méthodes d'instance

```

public void imprime(int niv)
public String toString()

```

On remarquera la présence de la méthode toString(), ce qui permet d'imprimer un arbre avec System.out.println(T);

```

*****
*** La classe ExpressionDeFonction ***
*****

```

//Champs

```

public String expression;

```

//Constructeur

```

public ExpressionDeFonction(String S)

```

//Méthodes d'instance

```

public String toString()

```

```

*****
*** La classe NomDeFonction ***
*****

```

//Champs

```

public String nom;

```

//Constructeur

```

public NomDeFonction(String S)

```

//Méthodes d'instance

```

public String toString()

```