Analyse dynamique avec Spoon

Simon Allier(`simon.allier@inria.fr`)

20 novembre 2015

1 Présentation de Spoon

Spoon [1] vous permet de transformer et d'analyser du code source Java. Spoon construit l'AST du programme cible, et permet de le modifier (modification, ajout ou suppression de noeud), pour ensuite (re)générer le code java correspondant.

Dans ce tp, nous allons utiliser Spoon afin d'analyser dynamiquement un programme (analyse de l'exécution d'un programme). Pour cela vous devez créer des `Processor` qui vont instrumenter le code de programmes.

1.1 Exemple

Le processor `LogProcessor` remplace tout les `System.out.println(String)` par un logger qui redirige toutes les sorties console vers un fichier log. Après compilation (`mvn package`), vous pouvez utiliser le processor sur un projet *example* de cette manière :

```
$ java -cp .:target/tpSpoon-1.0-SNAPSHOT-jar-with-dependencies.jar \
    vv.spoon.MainExample src/main/resources/example \
    src/main/resources/example-instrumented
```

La figure 1 montre la classe `C` du projet *example* une fois instrumentée. Vous pouvez ensuite vous rendre dans le répertoire `src/main/resources/example-instrumented` pour compiler le projet *example* instrumenté, puis pour l'exécuter de cette façon :

```
$ java -cp .:target/example-1.0-SNAPSHOT.jar example.A 2
```

Vous obtiendez alors un fichier nommé `log` avec le contenu de la figure 2.

1. http ://spoon.gforge.inria.fr/

```java
package example;

public class C  {
    private int i;

    public C(int i) {
/**        System.out.println("C.C(int i)");
**/
    vv.spoon.logger.LogWriter.out("C.C(int i)");
        this.i = i;
    }

    public int mth1() {
/**       System.out.println("C.mth1()");
**/
    vv.spoon.logger.LogWriter.out("C.mth1()");

        int result;
       try{
            result = 100/i;
       } catch (Exception e) {
/**           System.out.println("error in C.mth1()");
**/
    vv.spoon.logger.LogWriter.out("error in C.mth1()");
            result = -1;
       }
        return result;
    }
}
```

FIGURE 1 – Code de la classe `C` après intrumentation

```
INFO: A.main(String[] args)
INFO: A.mth1(int count)
INFO: B.mth1(int i)
INFO: C.C(int i)
INFO: C.mth1()
ERROR: error in A.mth1(int count)
INFO: B.mth1(int i)
INFO: C.C(int i)
INFO: C.mth1()
INFO: result = 100
INFO: B.mth2()
```

FIGURE 2 – Fichier de log obtenu après exécution du projet *example* intrumenté

2 Instrumentation de programmes

Dans cette partie, vous devez écrire deux processors qui modifient des programmes Java. Le pattern à suivre pour écrire un processor est le suivant : héritage de la classe AbstractProcessor, redéfinition des méthodes `isToBeProcessed` (si filtrage) et `process`, et création d'une classe Main pour lancer le processor. Lors de l'héritage, vous devez fournir un type Spoon (CtBlock, CtLoop, CtInvocation, etc.) correspondant à l'unique type d'élément de l'AST que votre processor va analyser. Tout cela peut se faire sur le modèle du LogProcessor qui est fourni. De nombreux exemples sont également disponibles sur le site de Spoon.

Pour une liste des types de l'AST, consultez la javadoc de Spoon :

`http://spoon.gforge.inria.fr/mvnsites/spoon-core/apidocs/spoon/reflect/code/package-summary.html`

`http://spoon.gforge.inria.fr/mvnsites/spoon-core/apidocs/spoon/reflect/declaration/package-summary.html`

Programming 1 *Utilisez Spoon afin de compter tous les appels de méthode d'un programme donné. Pour le projet example, le résultat doit être semblable à ceci :*

```
A.main(String[] args): 1
A.mth1(int count): 1
B.mth1(int i): 5
C.C(int i): 5
C.mth1(): 5
B.mth2(): 1
```

Programming 2 *Utilisez Spoon afin de construire l'arbre d'appel des méthodes d'un programme donné. Pour le projet example, le résultat doit être semblable à ceci :*

```
A.main(String[] args)
 |  A.mth1(int count)
 |  | B.mth1(int i)
 |  |  | C.C(int i)
 |  |  | C.mth1()
 |  | B.mth1(int i)
 |  |  | C.C(int i)
 |  |  | C.mth1()
 |  | B.mth2()
```