Tutorial Spoon

Rogliano Théo,Misse-Chanabier Pierre

Janvier 2019

1 Installation

- Faire l'étape 4 de l'url suivante: `http://www.vogella.com/tutorials/EclipseMaven/article.html#create-java-project`
- Clic droit sur le fichier XML (pom.xml) nouvellement généré
  -> Maven− > Add Dependency
  -> Remplir les 3 premières cases à l'aide des valeurs trouvées à l'adresse: `https://search.maven.org/artifact/fr.inria.gforge.spoon/spoon-core/7.2.0/jar`
  − >clic sur Ok
- clic droit sur pom.xml Run avec maven

Un main simple tel que le suivant permet de vérifier que maven importe bien spoon (exemple disponible sur le site spoon):

```
import spoon.Launcher;
import spoon.reflect.declaration.CtClass;

public class Main {

    public static void main(String[] args) {
        CtClass l = Launcher.parseClass("class A { void m() {
            System.out.println(\"yeah\");} }");
        System.out.println(l);
    }
}
```

2 Utilisation

Comme on peut le voir dans l'exemple précédent, on peut directement parseur un String grace à *Launcher.parseEntity*(), en remplaçant "Entity" par un type d'entités appartenant au méta-modèle de spoon (CtMethod, CtClass...).

Il est également possible d'extraire l'AST depuis le code source contenu dans des fichiers avec les quelques lignes suivantes, avec des commentaires les détaillants.

```
Launcher launcher = new Launcher();//Creation du launcher, qui vas
    accumuler les ressources
launcher.addInputResource("/path/to/project");//On ajoute un fichier ou
    un dossier
launcher.buildModel();//On cree le modele associe a ce projet
CtModel model = launcher.getModel();//Et on le recupere, pour le
    parcourir.
```

Puis il est possible de parcourir l'AST ainsi généré directement, en manipulant le *CtModel* et en utilisant des textitTypeFilters, où via des *Visitors*.

2.1 Accès direct le modèle

On peut donc accéder directement au élements de surfaces (packages, types...) en les demandant directement au modèle. Pour les élements encapsulés, il faut passer par le cheminement correct pour finalement arriver aux éléments associés.

Dans l'exemple suivant, on récupère toute les méthodes déclarées dans tout les types:

```
Collection<CtMethod> methods=new ArrayList<>();//resultat
for(CtType<?> t : model.getAllTypes()) {//iteration sur les classes
    for(CtMethod<?> m : t.getMethods()) {//et sur les methodes de
        chacune d'entre elles
        methods.add(m);
    }
}
```

Il existe également une façon de filtrer les éléments lorsque l'entité que l'on cherche à atteindre est dans une collection dont le contenu est polymorphique. Dans une méthode par exemple, il est possible de parser les éléments de façon différentes manières. Mais une distinction existe entre une invocation de méthode et une assignation d'objet par exemple. On applique donc un filtre sur les éléments récupérés de la méthode pour ne garder que ce qui nous intéresse (ici les CtInvocation):

```
for(CtInvocation<?> i : m.getElements(new
    TypeFilter<>(CtInvocation.class))) {
}
```

Cette méthode est évidement contraignante si on cherche à atteindre des éléments qui sont très profond dans l'arbre. C'est un problème que résout le visitor.

2.2 Visitors

Il suffit d'appliquer un visiteur sur le modèle:

```
AbstractVisitor visitor= new MyVisitor();//creation de l'objet visitor
model.accept(visitor);//Application sur le modele
visitor.getResult();//recuperation (si necessaire) d'un resultat,
    dependant du visitor utilise.
```

Le visiteur doit juste effectuer une description différentielle additionnelle de la classe CtAbstractVisitor, et réimplémenter les méthodes dont il a besoin. Par exemple, un visiteur qui récupère les déclarations de classes:

```
public class TypeVisitor extends AbstractVisitor{
    Collection<CtTypeParameter> res=new ArrayList<>();

    public void visitCtTypeParameter(CtTypeParameter typeParameter){
        res.add(typeParameter);
    }

    public Collection<CtClass> getResult(){
        return result;
    }
};
```

Les types visitable sont les classes implémentant l'interface Visitable. Leur liste est disponibles dans la page de la javadoc associée.

Note: Les visiteurs ne passent qu'une fois par noeud. Ils n'autorise pas la pré & post visit.