WALA チュートリアル 3
–ソースコード解析–

愛知県立大学 山本研究室
藤浦 祥雅*

はじめに

このチュートリアルでは，WALA[1] を用いた Java ソースコード解析について説明します．

WALA は，Polyglot[4] と Eclipse JDT[5](以降 JDT) の Java ソースコードフロントエンドを提供していますが，このチュートリアルでは，Eclipse JDT を用いた Java ソースコード解析について説明していきます．

1 Java ソースコードフロントエンド

バイトコード解析の場合，Shrike[2] を用いてバイトコードを解析し，WALA の IR(Intermediate Representation:中間表現) に変換していました (図 1)．

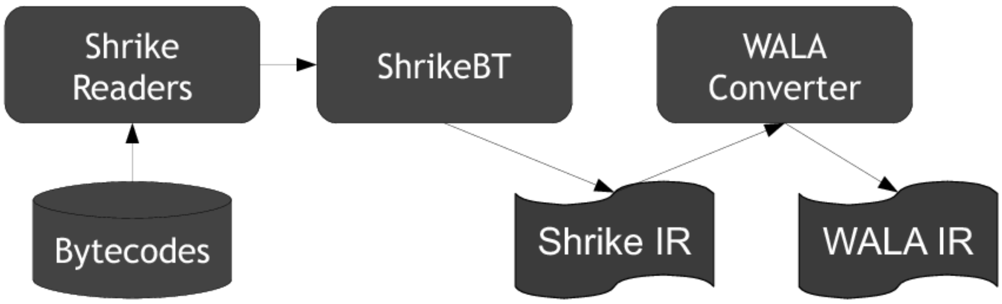

図 1: Shrike IR Construction[3]

ソースコード解析でも同様に，Polyglot や JDT を用いて AST(Abstract syntax tree:抽象構文木) を取得し，AST を WALA の IR へと変換します (2)．

ソースコードフロントエンドは 2 種類ありますが，それぞれ以下の特徴を持っています．

JDT フロントエンド

- Java 1.5 に対応 (Generic や annotation に対応)
- JDT が Eclipse Plugin であるため，Eclipse Plugin として実装しなければならない

*mail:[qtutorial@gmail.com]

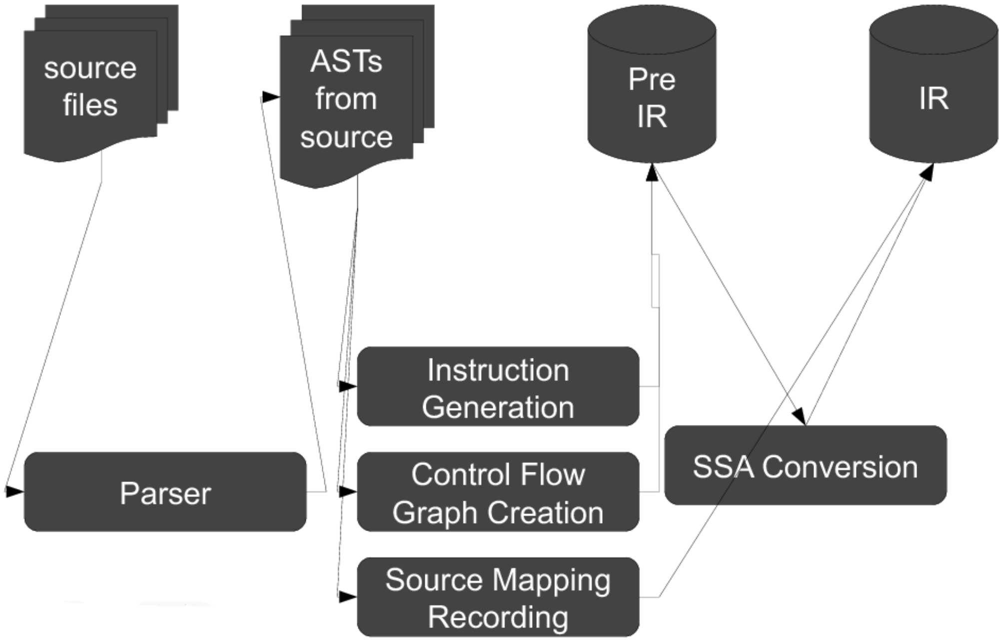

図 2: SourceCode IR Construction[3]

Polyglot フロントエンド

- Java 1.4 に対応 (Generic や annotation は非対応)
- polyglot は拡張可能な compiler framework であるため，実装形式に制限はない

2　前準備

バイトコードのときと同様に，まずプロジェクトの準備とテストデータの準備について説明します.

2.1　プロジェクトの準備

サンプルコードを記述するためのプロジェクトを作成します．JDT を用いたソースコード解析は Eclipse プラグインとして実装する必要があります．プロジェクトはバイトコード解析同様，Plugin プロジェクトを作成します.

2.1.1　プロジェクト作成

1. メニューの [File] → [New] → [Other] を選択
2. (Plug-in 開発内の) Plug-in プロジェクトを選択し，[Next] を選択
3. Project name を入力し，[Next] を選択

Finish を選択

バイトコード解析では手順 4 で Activator を作成しないよう設定しましたが，Plugin として実装するため，Activator を作成するように設定します．Activator は Plugin そのもののライフサイクルの管理を行うためのクラスである，Plugin は必ず Activator を持たなければなりません．

そして，テンプレートを利用せずにプロジェクトを作成します．このチュートリアルでは Project name を `jp.yamamotolab.tutrial.wala.srcana` とします．

2.1.2 必要なファイルのインポート

次に，Java ソースコード解析に必要な WALA プロジェクトをインポートします．インポートに関しては，本チュートリアルの第 1 回を参考にしてください．

今回必要なのは以下の 5 つのプロジェクトです．

1. wala.core
2. wala.ide
3. wala.cast
4. wala.cast.java
5. wala.cast.java.jdt

2.1.3 プロジェクトの設定

「Plug-in Dependencies」(プラグイン依存関係) に先ほどインポートした 5 つのプロジェクト (wala.core, wala.ide, wala.cast, wala.cast.java, wala.cast.java.jdt) と，4 つの Eclipse Plugin を追加します．

1. <PROJECT_TOP>/META-INF/MANIFEST.MF を開く
2. 画面下にある，[Dependencies] タブを選択
3. 「Required Plug-ins」に以下を追加
   (a) com.ibm.wala.core
   (b) com.ibm.wala.ide
   (c) com.ibm.wala.cast
   (d) com.ibm.wala.cast.java
   (e) com.ibm.wala.cast.java.jdt
   (f) org.eclipse.core.runtime
   (g) org.eclipse.core.resources
   (h) org.eclipse.ui
   (i) org.eclipse.jdt.core

2.1.4　拡張ポイントへの変更

今回は単一の Java ファイルに対して解析を行うこととします．またワークスペース上の.java ファイルを選択時に右クリックすると，ポップアップメニューに「HelloWALA」というメニューが表示され,「HelloWALA」の中にクラス階層を解析する「Type」，コールグラフを生成する「CallGraph」, SDG を作成する「SDG」をいう三つのメニューを表示させるようにします.

以下の方法でポップアップメニューを変更します.

1. <PROJECT_TOP>/META-INF/MANIFEST.MF を開く
2. Extensions タブを選択
3. 左側の「All Extensions」に，org.eclipse.ui.popupMenus を追加する
4. すると，<PROJECT_TOP>/plugin.xml が作成されるので，リスト 1 のように編集する.

リスト 1: plugin.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?eclipse version="3.5"?>
3  <plugin>
4     <extension point="org.eclipse.ui.popupMenus">
5          <objectContribution adaptable="true"
6                  id="HelloWALA.contribution"
7                  nameFilter="*.java"
8                  objectClass="org.eclipse.core.resources.IFile">
9                  <menu
10                         label="HelloWALA"
11                         path="additions"
12                         id="HelloWALA.menu">
13                         <separator name="group">
14                         </separator>
15                 </menu>
16                 <action
17                         label="SDG"
18                 class="jp.yamamotolab.tutrial.wala.srcana.SDGAction"
19                 menubarPath="HelloWALA.menu/group"
20                 enablesFor="1"
21                 id="HelloWALA.SDGAction">
22                 </action>
23                 <action
24                 label="CallGraph"
25                 class="jp.yamamotolab.tutrial.wala.srcana.CGAction"
26                 menubarPath="HelloWALA.menu/group"
27                 enablesFor="1"
28                 id="HelloWALA.CGAction">
29                 </action>
30                 <action
31                         label="Type"
32                         class="jp.yamamotolab.tutrial.wala.srcana.TypeAction"
33                         menubarPath="HelloWALA.menu/group"
34                         enablesFor="1"
35                         id="HelloWALA.typeAction">
36                 </action>
37         </objectContribution>
38   </extension>
39 </plugin>
```

リスト 1 は，ポップアップメニューに HelloWALA を追加し，さらにその中に Type, CallGraph, SDG というメニューを追加する拡張です．

また，7 行目の nameFilter により Java ファイルを選択したときのみ，このメニューは表示され，選択された Java ファイルは `IFile` クラスのオブジェクトとして扱われます (8 行目)．IFlie クラスは JDT のクラスで，Eclipse 内のファイルを表現するクラスです．

各メニューが押されたときに実行されるクラスは，TypeAction(32 行目), CGAction(25 行目), SDGAction(18 行目) のように指定されています．これらについては本チュートリアルの第 4 回以降に詳しく説明します．今回は三つに共通する部分について説明します．

2.1.5 Exclusion ファイルの準備

バイトコード解析同様，解析除外対象を指定するファイル (Exclusions.txt) を作成します．場所は作成したプロジェクトの直下に配置します．(<PROJECT_TOP>/Exclusions.txt のように)

以上で，プロジェクトの準備は完了です．

2.2 テストデータの準備

このチュートリアルでは，以下の Java コードを解析することとします．

リスト 2: Hello.java

```
public class Hello {
        public void run() {
                String s = "Hello, WALA!!";
                System.out.println(s);
        }

        public static void main(String[] args) {
                Hello h = new Hello();
                h.run();
        }
}
```

今回はソースコード解析なので，リスト 2 を含んだプロジェクトを作成します．新たに jp.yamamotolab.tutrial.wala.srcar というプロジェクトを作成し，リスト 2 を追加します．

以上で，テストデータの準備は完了です．

3 ソースコード解析をする

準備が整ったところで，本題のソースコード解析について説明したいと思います．

第 2 回では `AnalysisScope` クラスを使って，解析対象を管理していると説明しましたが，ソースコード解析では AnalysisScope やコールグラフ作成クラスなどを含めたエンジンである `JavaSourceAnalysisEngine` クラスが提供されています．

そのため，JavaSourceAnalysisEngine オブジェクトを生成することから始めます．

リスト 3: JavaSourceAnalysisEngine の生成

```
JavaSourceAnalysisEngine engine = new JDTJavaSourceAnalysisEngine();
```

以後この engine に設定を加えていきます．

リスト 4: Exclusion ファイルの設定

```
engine.setExclusionsFile(
 FileProvider.getFileFromPlugin(
                    Activator.getDefault(),
                    "Exclusions.txt").getAbsolutePath());
```

setExclusionsFile() の引数は String 型の Exclusions ファイルへのパスです．この例では Plugin として実装しているため，Plugin(Activator.getDefault()) のパスと Plugin に対する Exclusions ファイルの相対パス (”Exclusions.txt”) から Exclusions ファイルの絶対パス (getAbsolutePath()) を取得し，引数としています．FileProvider クラスは WALA が提供する Plugin 内のファイルを扱うユーティリティクラスです．

リスト 5: Java 標準ライブラリの設定

```
String rt = "C:\\Java\\jdk1.5.0_22\\jre\\lib\\rt.jar";

engine.addSystemModule(
   new JarFileModule(new JarFile(new File(rt))));
```

次に Java 標準ライブラリに関して設定します.

WALA では，解析ファイルのまとまりを `Module` として扱います．この例では Jar ファイルを扱っているため，`JarFileModule` クラスを利用して engine に追加されています.

Module には他にも，クラスファイルを表現する `ClassFileModule` やディレクトリツリーを表現する `DirectoryTreeModule` があります.

リスト 6: 解析ファイルの設定

```
private IFile selectedItem;

engine.addSourceModule(
            EclipseSourceFileModule.
                   createEclipseSourceFileModule(selectedItem));
```

最後に解析対象のファイルを設定します．先ほどの JarFileModule のように，Eclipse 内のファイルは `EclipseSourceFileModule` として扱われます．EclipseSourceFileModule オブジェクトは createEclipseSourceFileModule(IFile) を用いて生成します.

以上でソースコード解析を行うことができます.

例として，クラス階層の取得方法について説明します.

リスト 7: クラス階層の取得

```
IClassHierarchy cha = engine.buildClassHierarchy();
```

4 おわりに

今回はソースコード解析に関する以下のことを行いました.

- WALA の IR Constraction について
- プロジェクトの作成
- テストデータの作成
- JavaSourceAnalysisEngine について

次のチュートリアルでは，第 2 回，第 3 回で解析例として出てきたクラス階層について説明します.

参考文献

[1] WALA wiki：*http://wala.sourceforge.net/wiki/*

[2] Shrike : *http://wala.sourceforge.net/wiki/index.php/Shrike_technical_overview*

[3] WALA tutorial : *http://wala.sourceforge.net/files/PLDI_WALA_Tutorial.pdf*

[4] Polyglot : *http://www.cs.cornell.edu/projects/polyglot/*

[5] Eclipse Java development tools(JDT) : *http://www.eclipse.org/jdt/*

環境

- Windows XP
- Eclipse 3.5.2
- Java 1.5.22
- WALA 1.3.1M1

変更履歴

2010/12/8 : Ver 1.0 : 公開開始