2.4 Eclipse AST

Eclipse AST 是 Eclipse JDT 的一个重要组成部分，定义在包 org.eclipse.jdt.core.dom 中，用来表示 Java 语言中的所有语法结构。Eclipse AST 采用工厂方法模式和访问者模式（见 2.7 节）来设计和实现，这样可以减轻用户深入了解其内部结构的压力，并且方便用户利用它们构建并处理 AST。你可以打开 Eclipse 帮助页面，通过鼠标依次点击窗口左边的目录“JDT Plug-in Developer Guide”→“Reference”→“API Reference” →“org.eclipse.jdt.core.dom”，即可打开这个包的详细说明。本节将对这个包中重要的类作简要说明，为简便起见，这里省去这些类的包名，即 org.eclipse.jdt.core.dom。

在 Eclipse AST 中，与本书的课程设计相关的类主要有以下三部分：

- **ASTNode 类及其派生类**：用于描述各种 AST 节点的类，每个 AST 节点表示一个 Java 源程序中的一个语法结构，例如，一个名字、类型、表达式、语句或声明等。
- **AST 类**：创建 AST 节点的工厂类，类中包含许多创建各类 AST 节点的工厂方法，用户可以利用这些方法来构建 AST。
- **ASTVisitor 类**：AST 的访问者抽象类，类中声明了一组访问各类 AST 节点的 visit( ) 方法、endVisit( )方法和 preVisit( )方法。

2.4.1 AST 节点类

在 Eclipse AST 中，Java 源程序中的每个语法结构对应为一个 AST 节点，所有的 AST 节点按其在语法上的关系连接形成一棵 AST 树。类 ASTNode 是 AST 树中各类节点的抽象基类，其余的 AST 节点类都由它派生。在 ASTNode 类中声明有各个具体的 AST 节点类所对应的类型标识，如 ASTNode.COMPILATION_UNIT 代表 Compilation_Unit 节点类，这类节点用来表示一个 Java 源程序文件。

为便于自顶向下（从父节点到子节点）或者自底向上（从子节点到父节点）访问 AST 树，AST 节点含有指向其父节点的 parent 域以及若干关联的子节点域。在 AST 节点类中，以属性（property）来统一处理子节点以及用户自定义的节点属性，属性的访问方法有：

```
void setProperty(String propertyName, Object data)    // 设置指定属性的值
Object getProperty(String propertyName)               // 取得指定属性的值
Map properties( )                                     // 返回节点的所有属性表，这个表是不可修改的
```

在每个具体的 AST 节点类中，以类常量形式声明该类节点所拥有的基本属性（即基本的子节点）类别，并定义了存放属性值的域以及设置和访问属性的方法。例如，在一个 Java 源程序文件中，有可选的 package 声明、0 个或多个 import 声明以及至少 1 个类型声明（可以是类声明或接口声明），从而在表示 Java 源程序文件（称为编译单元）的 AST 节点类 CompilationUnit 中就声明有 final 类变量 PACKAGE_PROPERTY、IMPORTS_PROPERTY 和 TYPES_PROPERTY，分别表示 package 属性、imports 属性和 types 属性，同时还定义有如下的访问方法：

```
List imports( )          // 该节点的所有 import 声明，按在程序中的出现次序排列
```

List types( )　　// 该节点的所有顶层类型声明，按在程序中的出现次序排列

void setPackage(PackageDeclaration pkgDecl) // 设置该节点的 package 声明

PackageDeclaration getPackage( )　　// 取得该节点的 package 声明

其中，类型相同的子节点组成的序列以 java.util.List 接口类来表示，这个接口类包含 add、get、set、remove 等方法用于访问和修改序列。在实际构造和访问 AST 树时，需要注意统一所使用的 List 接口类的实现类，例如，可以统一使用类 java.util.LinkedList 或者统一使用类 java.util.ArrayList 来表示序列。

在 AST 节点类中，只提供获取父节点的方法，即

ASTNode getParent( )

而没有提供设置父节点的方法，这是因为对节点的 parent 域的设置是伴随着将该节点设置为其他节点的子节点而自动进行的。一个新创建的 AST 节点是没有设置其父节点的。当节点 A 通过形如 setCHILD 方法，如 A.setPackage(B)方法，或者通过序列的 add 或 set 方法，如 A.types( ).add(B)方法，将节点 B 设为自己的孩子时，B 节点的 parent 域将自动设置为对 A 节点的引用；对于那些因上述操作导致不再是 A 节点的子节点来说，其 parent 域将被自动设置为 null。

每个 AST 节点及其子节点只能归属于一棵 AST 树。如果将一棵 AST 树中的某个 AST 节点添加到另一棵 AST 树中，则必须复制这个节点及其所有的子孙节点，以保证这些节点只属于一棵 AST 树。此外，AST 树中不能含有环，如果某些操作会导致 AST 有环，则这些操作将失败。

为支持对源程序的分析和类型检查等，每个 AST 节点还含有一组位标志（用一个 int 型数表示）用来传播与该节点有关的附加信息，这些位标志可以通过节点的以下方法来存取：

void setFlags(int flags)

int getFlags( )

此外，Eclipse AST 还支持访问者模式，每个 AST 节点都含有方法：

void accept(ASTVisitor visitor)

用于统一表示对当前节点访问时所要执行的任务，这个任务由参数 visitor 来给定。你可以进一步了解 ASTVisitor 类以及访问者模式（见 2.7.2 节）来了解对 AST 树的访问。

2.4.2 AST 类

org.eclipse.jdt.core.dom.AST 是 AST 节点的工厂类，即它提供一系列形如

TYPE　newTYPE( )

的工厂方法，用来创建名为 TYPE 的 Eclipse AST 节点类的实例，新创建的节点并没有设置父节点。例如，方法

CompilationUnit newCompilationUnit( )

用来创建由这个 AST 所拥有的一个编译单元节点。

要使用这些方法，首先需要创建 AST 类的实例：

AST ast = AST.newAST(AST.JLS3);

其中，参数 AST.JLS3 指示所生成的 ast 包含处理 JLS3（Java 语言规范第 3 版）的 AST API。JLS3 是 Java 语言所有早期版本的超集，JLS3 API 可以用来处理直到 Java SE 6（即 JDK1.6）的 Java 程序。

2.4.3 ASTVisitor 类

org.eclipse.jdt.core.dom.ASTVisitor 是 AST 树的访问者类，它提供一套方法来实现对给定节点的访问。这套方法中有两组是与具体的 AST 节点类 T 相关的，即 visit 方法和 endVisit 方法，有两个是与具体的 AST 节点类无关的，即 preVisit 方法和 postVisit 方法。这些方法都通过参数接收一个 AST 节点 node，然后对这个节点进行访问以执行一些操作。

- public boolean visit(T node) 如果返回 true，则接着访问 node 的子节点；如果返回 false，则不再访问 node 的子节点。

  ASTVisitor 类提供的各个 visit 方法的缺省实现是：什么也不做，直接返回 true。子类可以根据需要重新实现这些方法中的部分或全部。
- public void endVisit(T node) 这类方法在节点 node 的子节点已经被访问或者是在 visit(node)返回为 false 之后被调用。

  ASTVisitor 类提供的各个 endVisit 方法的缺省实现是什么也不做。子类可以根据需要重新实现这些方法中的部分或全部。
- public void preVisit(ASTNode node) 这个方法在 visit(node)之前被调用。
- public void postVisit(ASTNode node) 这个方法在 endVisit(node)之后被调用。

  ASTVisitor 类提供的 preVisit 方法和 postVisit 方法的缺省实现是什么也不做。子类可以根据需要来重新实现它们。

在 Eclipse AST 中，结合 AST 节点的 accept( )方法和 ASTVisitor 实例，假设待访问的 AST 树的根节点为 root，则调用 root.accept( )就可以启动对这棵 AST 树的遍历。遍历是以深度优先搜索为基础的，你可以进一步查看 Eclipse JDT 的源代码来确认这一点，你也可以从 http://www.docjar.com/docs/api/org/eclipse/jdt/core/dom/index.html 上查看相关的源代码。为帮助大家理解对 AST 树的遍历过程，这里简要给出 accept 方法的实现。

所有的 AST 节点都执行在 ASTNode 类中定义的 accept 方法：

```
public final void accept(ASTVisitor visitor) {
    if (visitor == null) {    throw new IllegalArgumentException(); }
    visitor.preVisit(this);     // 执行与节点类型无关的 preVisit 方法
    accept0(visitor);    // 调用 accept0，执行与节点类型相关的 visit/endVisit 方法
    visitor.postVisit(this);    // 执行与节点类型无关的 postVisit 方法
}
```

ASTNode 类中的 accept0 方法是一个抽象的方法：

```
abstract void accept0(ASTVisitor visitor);
```

每个具体的 AST 节点类中都必须实现 accept0 方法，实现该方法的通用模板如下：

```
boolean visitChildren = visitor.visit(this);    // 调用 visit( )访问本节点
if (visitChildren) {                            // 如果 visit( )返回 true，则访问子节点
```

```
            acceptChild(visitor, getProperty1());     // 访问非序列型属性
            acceptChildren(visitor, rawListProperty);     // 访问序列型属性
            acceptChild(visitor, getProperty2());
        }
        visitor.endVisit(this);            // 调用 endVisit( )执行一些节点访问后的操作
```

从上面的模板可以看出，如果节点包含多个属性，如 CompilationUnit 节点中有 imports 属性和 types 属性等，则按这些属性在源程序中的先后次序来依次访问；如果一个属性为序列型，如 CompilationUnit 节点中的 types 属性，则调用 acceptChildren 方法来依次访问序列中的各个子节点；如果一个属性是非序列的，则调用 acceptChild 方法来访问。acceptChild 和 acceptChildren 这两个方法的实现都与 AST 节点的具体类型无关，故放在 ASTNode 类中：

```
    final void acceptChild(ASTVisitor visitor, ASTNode child) {
        if (child == null) {  return; }
        child.accept(visitor);
    }
    final void acceptChildren(ASTVisitor visitor, ASTNode.NodeList children) {
        NodeList.Cursor cursor = children.newCursor();
        try {
            while (cursor.hasNext()) {
                ASTNode child = (ASTNode) cursor.next();
                child.accept(visitor);
            }
        } finally {
            children.releaseCursor(cursor);
        }
    }
```

基于上述的实现机制，当你需要对 AST 树实现特定的访问功能时，你只需要结合实际需求设计和实现 ASTVisitor 类的子类就可以了。例如，在本章的课程设计中，你需要实现 ASTVisitor 类的派生类 InterpVisitor，在其中重写（override）与需要解释执行的语法结构相对应的 visit( )方法。

2.4.4 SimpleMiniJOOL 语言涉及的 AST 节点类

虽然 Eclipse AST 有为数众多的 AST 节点类，但是为表示 SimpleMiniJOOL 语言只会用到其中的一小部分，下面简要描述 SimpleMiniJOOL 语言所涉及到的 AST 节点类，重点介绍在使用这些类时所关注的成员。

2.4.4.1 整体结构

1、CompilationUnit 类（编译单元）

它用来表示一个 Java 源程序文件，本书中用来表示一个 MiniJOOL 语言或其子语言程序的全部，是所对应的 AST 树的根节点。虽然这类节点有 package 声明、import 声明列表

和类型声明列表等基本属性，但是在表示 SimpleMiniJOOL 程序时就只有一个类声明子节点。

主要成员

List types( ) // 返回该编译单元的顶层类型声明所对应的节点序列

2、TypeDeclaration 类（类型声明）

它用于表示 Java 语言中的类声明或接口声明，本书中用来表示 MiniJOOL 语言或其子语言中的类声明。在 SimpleMiniJOOL 程序中，仅有一个名为 Program 的类声明。

主要成员

MethodDeclaration[] getMethods( ) // 返回类声明的方法声明序列

以下成员在表示 SkipOOMiniJOOL 或 MiniJOOL 程序时才会用到：

FieldDeclaration[] getFields( ) // 返回类声明的域声明序列

Type getSuperclassType( ) // 返回该类声明的超类类型或 null

void setSuperclassType(Type superclassType) // 设置或清除超类

3、MethodDeclaration 类（方法声明）

它用于表示 Java 语言中的方法声明或者是构造器声明（**注意**：在 Java 语言规范中，并不是将构造器看成是一个方法。因为后者视为是类中的成员，可以被子类继承；而前者不能被子类继承）。这类节点的基本属性包括：方法或构造器名、方法或构造器的体、返回类型、形参列表等。在一个 SimpleMiniJOOL 程序里的唯一的类中，只有一个名为 main 的无参方法，这个方法声明表示为一个 MethodDeclaration 实例。

主要成员

Block getBody( ) // 返回该方法声明的方法体，如果没有体则为 null

void setBody(Block body) // 设置或清除该方法声明的方法体

SimpleName getName( ) // 返回所声明的方法名

void setName(SimpleName methodName) // 设置方法名

Type getReturnType2( ) // 返回所声明的方法的返回类型

void setReturnType2(Type type) // 设置返回类型

以下成员在表示 SkipOOMiniJOOL 或 MiniJOOL 程序时才会用到：

List parameters( ) // 返回所声明的方法的参数声明序列

boolean isConstructor( ) // 返回该声明是否是在声明构造器

void setConstructor(boolean isConstructor) // 设置是否是在声明构造器

2.4.4.2 语句

Statement 类是所有 Eclipse AST 中语句节点类的基类，由它派生出许多具体的语句节点类。在表示 SimpleMiniJOOL 程序时，只涉及到 Block、ExpressionStatement、IfStatement、WhileStatement、EmptyStatement、BreakStatement、ContinueStatement、ReturnStatement 等语句节点类。

1、Block 类（语句块）

它表示用花括号括起来的语句序列，可以用来表示一个方法体。其基本属性是语句序列。

主要成员

List statements( )　　// 返回该语句块中的语句序列

2、ExpressionStatement 类（表达式语句）

它表示由表达式形成的合法语句，包括表达式这一基本属性。在 SimpleMiniJOOL 语言中，可以用这个类来表示 print/read 语句或赋值表达式语句。

主要成员

Expression getExpression( )　// 返回该语句中的表达式

void setExpression(Expression expression) // 设置该语句中的表达式

3、IfStatement（if 语句）

它表示 if 或 if-else 语句，节点中包括 if 语句的表达式、then 分支以及可选的 else 分支这些基本属性。

主要成员

Expression getExpression( )	// 返回 if 语句的表达式
void setExpression(Expression expression)	// 设置 if 语句的表达式
Statement getThenStatement( )	// 返回 if 语句的 then 分支
void setThenStatement(Statement statement)	// 设置 then 分支
Statement getElseStatement( )	// 返回 if 语句的 else 分支
void setElseStatement(Statement statement)	// 设置 else 分支

4、WhileStatement 类（while 语句）

它表示 while 语句，节点中包括 while 语句的表达式和体这些基本属性。

主要成员

Expression getExpression( )	// 返回 while 语句的表达式
void setExpression(Expression expression)	// 设置 while 语句的表达式
Statement getBody( )	// 返回 while 语句的体
void setBody(Statement statement)	// 设置 while 语句的体

5、EmptyStatement 类（空语句）

它表示由分号组成的语句。

6、BreakStatement 类和 ContinueStatement 类

这两个类分别表示 break 语句和 continue 语句。在 Java 语言中，这两个语句是允许带有标号的，即形如“break label; ”或“continue label;”，因此提供 getLabel( )和 setLabel( )成员来访问标号。但是，在 MiniJOOL 语言或其子语言中，不支持标号。

7、ReturnStatement 类

它表示 return 语句。SimpleMiniJOOL 语言中的 return 语句是不允许含有表达式的，但是 MiniJOOL 语言和 SkipOOMiniJOOL 语言中的 return 语句允许含有表达式以支持返回值。故以下成员在表示 MiniJOOL 语言和 SkipOOMiniJOOL 语言时才会用到：

主要成员

```
Expression getExpression( )                    // 返回 return 语句的表达式
void setExpression(Expression expression)      // 设置 return 语句的表达式
```

2.4.4.3 表达式

Expression 类是所有 Eclipse AST 中表达式节点类的基类，由它派生出许多类。在表示 SimpleMiniJOOL 程序时，只涉及到 MethodInvocation、Assignment、InfixExpression、PrefixExpression、ParenthesizedExpression、NumberLiteral、Name 这些表达式节点类。

1、MethodInvocation 类（方法调用）

它用来表示 Java 程序中的方法调用。MiniJOOL 语言中的方法调用形如：

[Expression .] Identifier ([Expression { , Expression }])

开头的“Expression .”是限制被调用方法的受限表达式，可以是类名、实例表达式或者没有。在 SkipOOMiniJOOL 语言中，方法调用不存在受限表达式。虽然 SimpleMiniJOOL 语言中没有方法调用，但是程序中的“print/read(<参数>)”用方法调用节点来表示。

主要成员

```
SimpleName getName( )                    // 返回所调用的方法名
void setName(SimpleName methodName)      // 设置方法名
List arguments( )    // 返回方法调用表达式中的实参表达式序列
```

以下成员在表示 MiniJOOL 程序时才会用到：

```
Expression getExpression( )                    // 返回方法调用的受限表达式
void setExpression(Expression expression)      // 设置方法调用的受限表达式
```

2、Assignment 类（赋值表达式）

它用于表示赋值表达式，包含运算符、左部和右部等基本属性。赋值运算符是在内部类 Assignment.Operator 中定义。MiniJOOL 语言或其子语言中支持的赋值运算符包括 =、+=、−=、*=、/=、%=。

主要成员

```
Expression getLeftHandSide( )    // 返回赋值表达式的左部
void setLeftHandSide(Expression expr)    // 设置赋值表达式的左部 Expression
getRightHandSide( )    // 返回赋值表达式的右部
void setRightHandSide(Expression expr)    // 设置赋值表达式的右部
Assignment.Operator getOperator( ) // 返回赋值表达式的运算符
void setOperator(Assignment.Operator op) // 设置赋值表达式的运算符
```

赋值运算符

在内部类 Assignment.Operator 中，定义有许多类型为 static Assignment.Operator 的赋值运算符，其中与 MiniJOOL 语言及其子语言有关的有：

ASSIGN：表示 =，可以用 Assignment.Operator.ASSIGN 来访问，以下类似

PLUS_ASSIGN：表示+=

MINUS_ASSIGN：表示 -=

TIMES_ASSIGN：表示 *=

DIVIDE_ASSIGN：表示 /=

REMAINDER_ASSIGN：表示 %=

3、InfixExpression 类（中缀表达式）

它用于表示中缀表达式，包含运算符、左操作数和右操作数等基本属性。运算符是在内部类 InfixExpression.Operator 中定义。MiniJOOL 语言或其子语言中支持的中缀运算符包括包括+、−、*、/、%、==、!=、<、<=、>、>=、&&、||等。

主要成员

```
Expression getLeftOperand( )   // 返回中缀表达式的左操作数
void setLeftOperand(Expression expr)        // 设置左操作数
Expression getRightOperand( )               // 返回中缀表达式的右操作数
void setRightOperand(Expression expr)       // 设置右操作数
InfixExpression.Operator getOperator( )     // 返回中缀表达式的运算符
void setOperator(InfixExpression.Operator op) // 设置运算符
```

中缀运算符

本书涉及内部类 InfixExpression.Operator 中定义的以下 static InfixExpression.Operator 类型的实例：

```
+   PLUS          -   MINUS         *   TIMES         / DIVIDE
%   REMAINDER     ==  EQUALS        != NOT_EQUALS
<   LESS      > GREATER       <=  LESS_EQUALS   >=  GREATER_EQUALS
&&  CONDITIONAL_AND     ||  CONDITIONAL_OR
```

4、PrefixExpression 类（前缀表达式）

它用于表示前缀表达式，包含运算符、操作数等基本属性。运算符是在内部类 PrefixExpression.Operator 中定义。在 SimpleMiniJOOL 中，包括−、+、!等前缀运算符。

主要成员

```
Expression getOperand( )   // 返回前缀表达式的操作数
void setOperand(Expression expr)   // 设置操作数
PrefixExpression.Operator getOperator( ) // 返回前缀表达式的运算符
void setOperator(PrefixExpression.Operator op) // 设置运算符
```

前缀运算符

本书涉及内部类 PrefixExpression.Operator 中定义的以下 static PrefixExpression.Operator 类型的实例：

+ PLUS - MINUS ! NOT

5、ParenthesizedExpression 类（带括号的表达式）

主要成员

Expression getExpression( ) // 返回括号内的表达式

void setExpression(Expression expression) // 设置括号内的表达式

6、NumberLiteral 类（整数）

主要成员

String getToken( ) // 返回对应的整数串

void setToken(String token) // 设置整数串

7、Name 类

它用于表示一个名字，由它派生出 QualifiedName 和 SimpleName 两个类，前者表示一个形如 a.b 的受限名，后者表示一个简单名。在 SimpleMiniJOOL 程序中只会出现简单名。

SimpleName 的主要成员

String getIdentifier( ) // 返回标识符

void setIdentifier(String expression) // 设置标识符

boolean isDeclaration( ) // 该标识符是否定义过

QualifiedName 的主要成员

SimpleName getName( ) // 返回受限名中的名字部分

void setName(SimpleName name) // 设置受限名中的名字部分

Name getQualifier( ) // 返回受限名中的受限部分

void setQualifier(Name qualifier) // 设置受限名中的受限部分

2.4.5 Eclipse AST 使用示例

在这一节中，我们将演示如何利用 Eclipse AST 手工构建如下的 SimpleMiniJOOL 程序的 AST 中间表示。

```
class Program {
    static void main() {
        i = 10;
    }
}
```

首先，你需要通过 Eclipse AST 工厂类中的方法 newAST()建立一个 AST 实例：

AST ast = AST.newAST(JLS3);

利用这个 AST 实例，就可以按如下的方法创建各种 AST 节点，并构建完整的抽象语法树。

然后，利用 Eclipse AST 工厂类中的各种创建方法按如下步骤创建所需要的 AST 节点：

1） 整个 SimpleMiniJOOL 程序构成一个 CompilationUnit：

```
CompilationUnit cu = ast.newCompilationUnit();
```

2） 在 CompilationUnit 实例中包含一个 TypeDeclaration，表示程序中的类 Program：

```
TypeDeclaration type = ast.newTypeDeclaration( );
type.setName(ast.newSimpleName(“Program”));      // 定义类的名字
```

3） 在这个 TypeDeclaration 实例中添加类 Program 中的方法 main()：

```
MethodDeclaration method = ast.newMethodDeclaration( );
method.setName(ast.newSimpleName(“main”));
type.bodyDeclarations().add(method);
// 设置方法 main()的 modifier 为 static
method.modifiers().add(
    ast.newModifier(Modifier.ModifierKeyword.STATIC_KEYWORD));
// 设置方法 main()的返回类型为 void
method.setReturnType2(ast.newPrimitiveType(PrimitiveType.VOID));
```

4） 构造 main 函数的函数体 mainBody

```
Block mainBody = ast.newBlock();
method.setBody(mainBody);
```

5） 向方法 main 函数体 mainBody 中添加语句

```
// 构建赋值表达式
Assignment assign = ast.newAssignment();
// 设置赋值表达式的左值为 i
assign.setLeftHandSide(ast.newSimpleName("i"));
// 设置赋值表达式的赋值算符为=
assign.setOperator(Assignment.Operator.ASSIGN);
// 设置赋值表达式的右值为数字 10
assign.setRightHandSide(ast.newNumberLiteral("10"));
// 由赋值表达式构建语句，并把这个语句加入方法 Main()的函数体
ExpressionStatement statement = ast.newExpressionStatement(assign);
mainBody.statements().add(statement);
```

至此，用 Eclipse AST 表示的 SimpleMiniJOOL 程序的抽象语法树就构建完毕了。

在 lab1/src/edu/ustc/cs/compile/interpreter/TestCase.java 中的 createSampleAST( )方法给出了构建一个简单 SimpleMiniJOOL 程序对应的 AST 的完整示例。

需要再次强调的是，使用 Eclipse AST 构建的抽象语法树在拓扑结构上必须是无环的。无论是手工构建 AST 还是自动构建 AST，你都需要小心的检查自己的代码，避免违反这个原则。