第 33 卷 第 19 期 计 算 机 工 程 2007 年 10 月
Vol.33 No.19 Computer Engineering October 2007
·软件技术与数据库· 文章编号：1000—3428(2007)19—0051—03 文献标识码：A 中图分类号：TP311

一种构建 Java 语言扩展的编译器框架

张 昱
(中国科学技术大学计算机科学技术系，合肥 230027)

摘 要: Polyglot 是一种避免代码复制的、高度可扩展的编译器前端框架。用户只需扩展该框架，定义对抽象语法树、语义分析等的必要修改，即可实现 Java 语言扩展。为指导用户在 Polyglot 上快速开展工作，该文总结 Polyglot 的主要流程和类体系，剖析其利用委托、访问者、抽象工厂等实现可升级扩展的方法，简述了 Polyglot 的已有应用及应用步骤。

关键词: 可升级扩展；遍；访问者；委托

Compiler Framework for Building Java Language Extensions

ZHANG Yu
(Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027)

【Abstract】 Polyglot is a highly extensible compiler frontend framework, while avoiding code duplication. Users may extend the framework to define any necessary changes on abstract syntax tree and semantic analysis for implementing language extensions. In order to direct users to work with Polyglot fleetly, the main flow and class hierarchy of Polyglot are summarized, and the methodologies of the scalable extensibility are anatomized, in which the delegation, visitors and abstract factories are employed. And the use steps of Polyglot and the related application are stated in a nut shell.

【Key words】 scalable extensibility; pass; visitor; delegation

在软件安全、新型语言设计等研究中，往往需要扩展现有的程序设计语言，构造编译器。构造和维护编译器是一项繁杂的工作，要快速实现扩展语言比较困难。

为支持类Java语言的编译器构造，2003 年Cornell大学研制了高度可扩展的编译器前端框架Polyglot，并为基语言Java 1.4 实现了一个可扩展编译器[1]。在该框架下，通过定义对编译过程的必要修改(抽象语法树AST和语义分析等)可实现语言扩展。Polyglot 的主要目标是可升级扩展(scalable extensibility)，即为实现扩展所需的编码量只与扩展语言和基语言间的差异成比例。

本文通过剖析 Polyglot 源码，总结其设计和实现体系及可升级扩展的方法，期望能帮助用户快速地在 Polyglot 上开展工作。

1 Polyglot 概览

图 1 是Polyglot的总体结构。扩展语言代码输入到分析器后产生AST。对AST依次执行遍调度器预设的各编译遍，可得最终的Java AST和序列化的类型信息；再调用如javac的编译器生成字节码。其中词法分析器由JFlex生成，语法分析器由CUP[2]生成；遍负责执行各种编译行为。

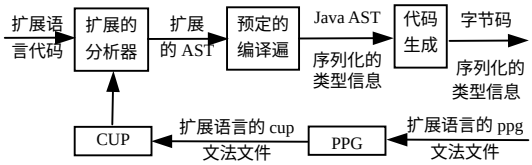

图 1 Polyglot 的总体结构

一般来说，扩展语言与基语言在词法上相同；而在语法上会有不少变动，这些变动会导致 AST 结构等的变化。这时，就需要根据扩展需求，定义扩展的 AST 并构造扩展的语法分析器。

Polyglot 提供 PPG(Polyglot parser generator)帮助快速建立语法分析器，它相当于 CUP 的预处理器，其代码在 ppg 包中。PPG 提供文法继承，用户只需将扩展语言相对于基语言文法的变化集定义成.ppg 文件输入到 PPG 运行，即可得到描述该扩展语言完整文法的.cup 文件。PPG 文法是对 CUP 文法的扩展，增加的命令见表 1。根据扩展的 AST，对 PPG 生成的.cup 文件中的语义动作等进行修改，再将该文件输入到 CUP 中运行，即可得扩展的语法分析器。

表 1 PPG 文法格式中扩展的命令

命令	含义
include “filename”	它出现在 ppg 文件的第 1 行，用于指定被继承的文法文件(是.cup 或.ppg 文件)
precedence[left | right | nonassoc] tokenlist;	指定新的优先级规则，含义同 CUP
precedence;	删除被继承文法中的所有优先级规则
drop { symbol}	删除指定的终结符或非终结符 symbol
drop {S ::= <productions>; }	删除指定的产生式 productions
override S ::= <productions>;	用指定的产生式覆盖 S 的产生式
extend S ::= <productions>;	将指定的产生式加入到 S 的定义中
transfer S to A_1 { rhs_1 } to A_n { rhs_n }	将非终结符S的产生式分支rhs_i传递到非终结符A_i中
start with S_1 $func_1$ … start with S_n $func_n$	支持文法中有多个开始符，每个开始符S_i对应的启动分析函数是$func_i$

对扩展语言的编译可通过修改基语言的遍调度器来实现，这可能要增加、重排、修改或删除遍。各个遍使用类型

基金项目： Intel 公司研究基金资助项目
作者简介： 张 昱(1972－)，女，副教授，主研方向：程序设计语言理论和实现技术，XML 数据管理
收稿日期： 2006-10-12 **E-mail：** yuzhang@ustc.edu.cn

系统(type system)、节点工厂(node factory)等对象。其中，类型系统是类型对象等的工厂，提供类型检查功能；节点工厂用于构造 AST 节点。

这样在用 Polyglot 实现语言扩展前，需要了解其中的：(1)编译流程及类体系；(2)AST 结构及扩展方法；(3)已有的遍及遍调度器。

2 Polyglot 的深入剖析

为解答上述问题，笔者剖析 2006 年 5 月发布的 Polyglot 1.3.3 源码。表 2 是其中 polyglot 包的包结构，为简便起见，下文省去包前缀"polyglot."。

表 2 polyglot 包的包结构

包名	含义
main	包含主类 Main 和选项处理类 Options 等
ast	包含各种 AST 节点的接口(基类是 Node)、扩展接口 Ext 和委托接口 Del 及其工厂接口 NodeFactory, ExtFactory 和 DelFactory
types	包含各种类型接口(基类是 Type)、表示编译时值的类型对象接口(基类是 TypeObject)及类型系统接口 TypeSystem
lex	包含 Lexer 接口及实现各种词法记号的类(基类是 Token)
parse	包含实用的语法分析器类 BaseParser
frontend	包含用于前端分析的各种 helper 类
visit	包含一些标准的访问者类
ext	安放各种扩展语言
ext.jl	为基语言 Java1.4 实现的可扩展编译器
ext.coffer	支持信息流安全的扩展
ext.pao	支持将简单类型当作对象(Primitives as Objects)
ext.param	实现参数化类的抽象扩展
ext.skel	语言扩展的骨架，填充内容可实现语言扩展

2.1 Polyglot 的主要类结构

图 2 是 Polyglot 的类结构。一个 Job 对象负责管理一个源文件所执行的编译任务，其中 passes 域保存对应的各个遍；每个 Job 对象会有 Node 型域 ast 指向对应的 AST 的根。Node 声明节点需要实现的公共接口。

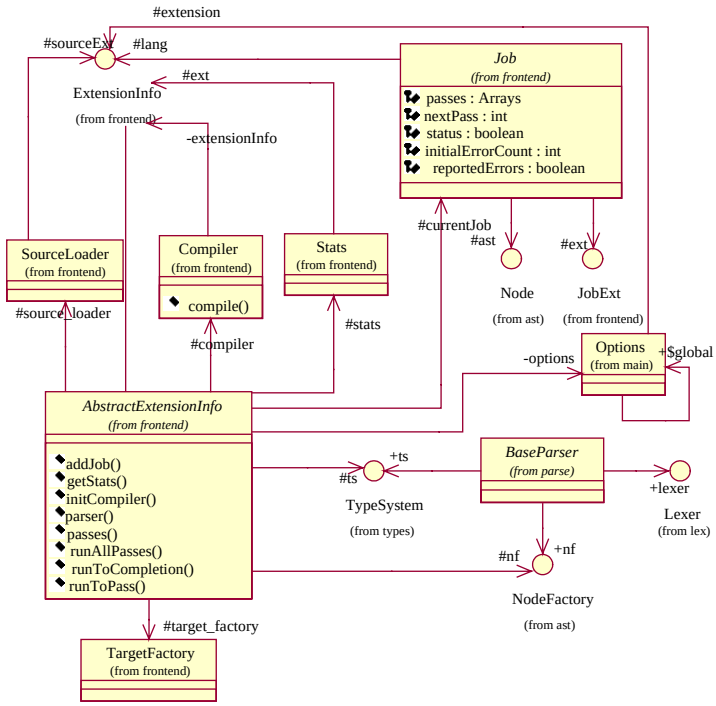

图 2 Polyglot 的主要类结构

扩展语言需实现 frontend.ExtensionInfo 接口定义语言的类型系统、节点工厂、分析器和其他参数等。AbstractExtensionInfo 是实现该接口的抽象类，它包含初始化编译器(initCompiler)、构造 Job 对象(addJob)、执行 Job(runToCompletion)等的通用实现。这样，扩展语言只需从它派生自己的 ExtensionInfo 类，在类中实现如 passes(定义遍序列)、parser(创建分析器对象)、createNodeFactory、createTypeSystem 等简单的方法。

Compiler 类负责管理输入和输出文件并启动扩展语言的编译过程，其 compile 方法负责调用扩展语言 ExtensionInfo 对象中的方法为每一源文件构造 Job 对象并执行所有的 Job。

2.2 实现可升级扩展的问题

AST 和遍是 Polyglot 的核心，其结构设计将直接影响 Polyglot 能否支持可升级扩展。当扩展语言时，可能需要：(1)对某些 AST 节点类及其子类统一增加域或方法；(2)增加新的 AST 节点类；(3)增加在 AST 上的行为，即新遍。

对于问题(1)，可通过子类化扩展这些具有多个子类的节点类得到新的节点类，这时要复制各个子类，并更新各子类复本中的父类名。这种做法不满足可升级扩展性，因为影响多个类的一个变化会导致大量子类代码的复制。对于问题(2)和问题(3)。遍通常采用 Visitor 模式实现对 AST 的行为，每个 Visitor 需要为其涉及的每一节点类实现一个访问方法。Visitor 模式允许新遍(问题(3))的可升级增加，但不支持 AST 节点类的可升级增加。因为当增加新节点类时(问题(2))，需要修改所有现有 Visitor 以插入对新节点类的访问方法。

2.3 节点的可升级扩展

为支持节点的可升级扩展，Polyglot 在节点类中增加 ext 域，它指向节点的扩展对象。通过扩展对象可以实现为节点类及其子类增加新的方法和域(问题(1))，而不必采用子类化扩展。扩展的方法和域可通过强制 ext 为实际扩展对象类型来访问。如果有多个 AST 节点类需扩展相同的成员，则这些节点类可以使用同一扩展对象类，避免了代码复制。

当基语言被多次扩展时，节点的方法及其扩展对象中的方法可能会被多次重写(override)。这时需要有一种机制能让节点或其扩展对象调用正确的方法实现。为解决此问题，Polyglot 将委托(delegation)机制引入到节点结构中(见图 3)。

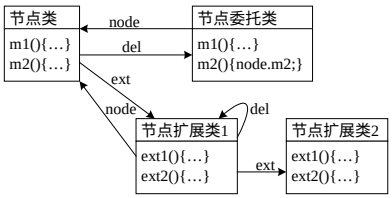

图 3 节点和扩展对象上的委托机制

以节点方法的重写与正确调用为例，Polyglot 在节点类中增加委托域 del，它指向实现 Node 接口的委托对象，缺省指向节点本身。为重写节点的方法，语言实现者会创建一个新的委托类，它可以自己或分派实现要重写的方法，也可以通过 node 域调用节点或节点的扩展对象中的方法来实现。为确保调用正确的节点方法实现，应通过节点的 del 域调用节点的方法。类似地，可以在扩展对象类中增加委托域 del、扩展对象域 ext 和关联的节点域 node 实现扩展方法的重写和正确调用。

2.4 遍的可升级扩展

frontend.Pass 声明遍要实现的接口，包括计时及 run 等方法。AbstractPass 是实现 Pass 的抽象类，给出了计时的实现。

从它派生各种遍，有分析遍 ParserPass，根据指定 Visitor 对 AST 检查或变换的 VisitorPass，产生新 Job 并在当前上下文执行该 Job 的 SpawnPass，同步遍 BarrierPass 以及代码生成遍 OutputPass 等，它们需要实现各自的 run 方法。

绝大多数遍都是 VisitorPass 的实例。VisitorPass 中增加 Job 型域 job 和 NodeVisitor 型域 v，表示对 job 的 AST 按 v 进行访问。NodeVisitor 是各种访问者的基类。不同的访问者虽然有不同的行为，但是都需遍历 AST。为消除实现新遍时复制按节点访问其孩子的代码，Polyglot 将这部分代码分离出来放到节点的 visitChildren 方法中。VisitorPass 的 run 流程如图 4。

图 4　VisitorPass 的执行流程

各 NodeVisitor 子类只需实现 begin、enter、leave、finish 和 override 方法，而 visitChildren 由节点类实现，独立于访问者。

在 visit 包中有近 30 种 Visitor(图 5)，每一层类都有各自的功能。以数据流分析为例，超类 HaltingVisitor 提供 bypass(Node n)用于跳过对以 n 为根的子树的遍历；ErrorHandlingVisitor 类增加了统一的错误处理；DataFlow 实现数据流分析的框架，这样其子类只需定义内部类 DataFlowItem 描述数据流项，并实现汇合(confluence)和流动(flow)方法。

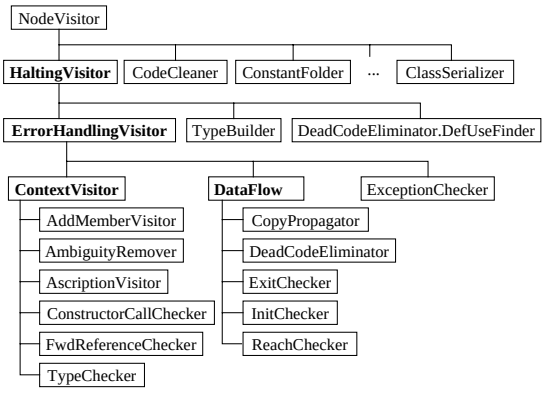

图 5　Polyglot 中 Visitor 类的层次结构

3　Polyglot 的应用

3.1　应用步骤

假设扩展语言名为 XX，其构建步骤如下：

(1)选择基语言，分析扩展语言和它的差异；

(2)选择基本 AST 节点类，在 ext.XX.ast 包中扩展定义自己的节点类，给出各工厂类的实现；

(3)如需扩展节点的方法，则在 ext.XX.extension 包中扩展 ast.Ext 接口得到命名为 XXExt 的接口并给出实现 XXExt_c；根据需要给出特殊节点的扩展类实现；

(4)如需改变类型系统，在 ext.XX.types 包中给出接口 XXTypeSystem 及实现 XXTypeSystem_c；

(5)给出语言的词法文件，用 JFlex 生成 Lexer_c；定义语言的 ppg 文件并经 PPG 和 CUP 处理生成分析器类 Grm 和符号类 Sym，将它们加到 ext.XX.parse 包中；

(6)在 ext.XX.visit 包中定义新增的访问者类；

(7) 定义 ext.XX.ExtensionInfo 类，重写其中的 createNodeFactory, createTypeSystem, parser, passes, compilerName, defaultFileExtension 方法。

3.2　基于 Polyglot 的项目

在 Polyglot 主页上链接有许多基于 Polyglot 的项目。这些项目的分类如下：

(1)Java 程序分析和优化：如 Soot 是 Java 优化框架，它利用 Polyglot 提供的 AST 完成到其它中间形式的转换；Jedd 是以二叉决策图为基础实现指针分析的一种 Java 扩展语言，它们均由 McGill 大学的 Sable 研究组研制。

(2)并行处理：如 jCilk 项目是针对 Java 不支持将异常或返回值从一个线程传回其父线程而做的扩展，它提供多线程的调用-返回语义，集成了多线程的异常处理；AtomJava 是 Washington 大学正在进行的通过软件事务进行可升级并行抽象的研究项目。

(3)信息流安全：例如 Jif 通过增加表达如何使用信息的限制标签来扩展 Java，是支持静态信息流控制的类型安全的编程语言，用以保护计算机系统所处理信息的机密性(confidentiality)和完整性(integrity)。

(4)函数式范例：如 Jmatch 提供抽象的可迭代的模式匹配，其迭代器可中断，DJ 通过将函数作为一等公民加到 Java 中来混合函数式范例和面向对象范例。

(5)新的面向对象特征：如 Jx/J&提供嵌套继承以便于软件复用和合成，Sable 组实现了支持 Java 5 的 Polyglot。

(6)逻辑式范例：如 J-LO 用于支持运行时的时序断言检查，它基于 Java 5 注解和 Haskell 原型。

4　结束语

Polyglot 具有清晰的包结构和类体系，其可升级扩展性为实现语言扩展提供了便利。近年来随着多核技术的发展，许多研究组织和公司开展可升级扩展的新型并行编程语言的研究，以适应在多核平台上的软件研制需求。Polyglot 为探索以 Java 为基础的可升级并行新特性和范例的研究提供了良好的基础平台。笔者期望通过本文的介绍能给国内从事相关工作的人员以帮助，进而吸引更多的研究人员参加与编译器有关的实践和研究，推动国内在这方面的发展。

参考文献

1 Nystrom N, Clarkson M R, Myers A C. Polyglot: An Extensible Compiler Framework for Java[C]//Proc. of the 12th International Conference on Compiler Construction, Warsaw, Poland. 2003.

2 Hudson S E. CUP Parser Generator for Java (v0.10k)[Z]. (1999-09). http://www.cs.princeton.edu/~appel/modern/java/CUP.