JavaCC、解析树和 XQuery 语法，第 1 部分

使用 BNF 和 JavaCC 构建定制的解析器

Howard Katz (howardk@fatdog.com), 业主, Fatdog Software

简介： 在简要讨论了语法、解析器和 BNF 后，本文将介绍 JavaCC，这是一个流行的解析器生成器工具。您将开发使用 JavaCC 的样本代码来构建定制的解析器，先从语法的 BNF 描述开始。第 2 部分接着将演示如何使用辅助工具 — JJTree 来构建同一解析的解析树表示，以及如何在运行时遍历该树，以发现其状态信息。文章将以开发构建和遍历解析树的样本代码作为结束，该解析树是您为一小部分 XQuery 语法生成的。

发布日期： 2002 年 12 月 01 日
级别： 初级

要完成最简单的日常解析任务，您不需要使用象自动化解析器生成器那样复杂的任何东西。例如，同“梳理”CSV（逗号分割的值，Comma-Separated-Value）文件的各部分同样简单的编程练习需要了解文件的结构，可能还需要了解如何使用 Java `StringTokenizer` 。另外，CSV 练习还需要了解很少的解析理论知识或者将自动化工具应用于任务的需求。

但是，一旦正式描述它的某种语言和语法变得复杂，那么语言中的有效表达式数量将迅速增加。而能够手工处理将任意表达式解析成其组成部分（这或多或少是解析更简明的定义）所需的代码将变得越来越困难。自动化解析器生成器减轻了这种困难。其他程序员或许也可以将您生成的解析器用于他们自己的用途。

从 BNF 开始

复杂语言的语法通常都是使用 BNF（巴科斯-诺尔范式，Backus-Naur Form）表示法或者其“近亲”— EBNF（扩展的 BNF）描述的。自动化工具可以使用那些描述（我将使用通用的术语 *BNF*来指代这两种变体）或与它们近似的描述来为您生成解析代码。本文就描述了这样的一种解析器-生成器工具，称为 JavaCC。我将简要地研究一下 JavaCC 的基本知识，并在结束的时候花些时间研究一下它的一个辅助工具 — JJTree，但是在讨论中不会介绍太多的理论知识，以免偏离主题。本文力图阐明我的理念：您并不需要了解很多有关正规的解析理论就能进行解析！

为什么使用 JavaCC 呢？有几个原因：我对 XQuery 有着强烈的兴趣，而 W3C 的 XML Query 工作组恰好使用 JavaCC 来构建并测试 XQuery 语法的版本，并且构建和测试它与 XSL 组共享的 XPath 语法。我还使用 JavaCC 来提供 XQEngine 中的查询-解析代码，XQEngine 是我自己的开放源码 XQuery 实现（请参阅参考资料）。

最后一点（但不是最不重要的），价格是完全合适的：尽管 JavaCC 不是开放源码，但它是完全免费的。（请参阅 参考资料以了解有关如何获得 JavaCC 的信息）。

解析 101

在我开始介绍一些实际的 XQuery 语法之前，让我们先从一个非常简单的 BNF 开始，它描述了一种语言，该语言仅由两个只对整数进行运算的算术运算符构成。我称这种语言为 `SimpleLang` ：

```
simpleLang     ::=   integerLiteral ( ( "+" | "-" ) integerLiteral )?
integerLiteral ::=   [ 0-9 ]+
```

该语法中的每个规则都是一个 **结果（production）**，其中左边的项（结果的名称）是依据语法中的其它结果描述的。最上面的结果 simpleLang 表明，该语言中有效的（或合法的）表达式是这样构成的，一个整数值，可以任意选择其后跟一个加号（+）或减号（-）以及另一个整数值或不跟任何东西。按照这种语法，单个整数“42”是有效的，同样，表达式“42 + 1”也是有效的。第二个结果以类 regex 的方式更特定地描述了一个整数值看上去象什么：一个或多个数字的连续序列。

该语法描述了两个结果 simpleLang 和 integerLiteral 之间存在的抽象关系。它还详细描述了三个 **记号**（加号、减号和整数）组合的具体项，解析器在扫描整个输入流时希望遇到这些项。解析器中负责该任务的部件称为 **扫描器（scanner）**或 **记号赋予器（tokenizer）** 一点也不稀奇。在该语言中， simpleLang 是 *非终端（non-terminal）* 符号的一个示例，它对其它结果进行引用；另一方面，规则 integerLiteral 描述了 *终端（terminal）* 符号：这是一种不能进一步分解成其它结果的符号。

如果解析器在其扫描期间发现了除这三个记号外的任何 *其它*记号，则认为它正在扫描的表达式是无效的。解析器的主要工作之一就是确定您传递给它的任何表达式的有效性，并且让您知道。一旦认为某个表达式是有效的，则它的第二项工作是将输入流分解成其组件块，并以某个有用的方式将它们提供给您。

从 BNF 到 JavaCC

让我们看看如何使用 JavaCC 实现该语法。JavaCC 使用称为 .jj 的文件。该文件中的语法描述是使用非常类似于 BNF 的表示法编写的，这样从一种形式转换到另一种形式通常就相当容易。（该表示法有自己的语法，从而使其在 JavaCC 中是可表达的。）

JavaCC .jj 文件语法和标准的 BNF 之间的主要区别在于：利用 JavaCC 版本，您可以在语法中嵌入操作。一旦成功遍历了语法中的那些部分，则执行这些操作。操作都是 Java 语句，它们是解析器 Java 源代码的一部分，该部分作为解析器生成过程的一部分产生。

（注：除了一条 Java println() 语句外， 清单 1并不包含您需要用来对该语言中的表达式实际求值的嵌入式 Java 代码。当您研究过 JJTree 及其解析树表示后，我将对此做更详细的研究。）

清单 1. 编码 SimpleLang 语法的完整 .jj 脚本

```
PARSER_BEGIN( Parser_1 )
package examples.example_1;
public class Parser_1 {}
PARSER_END( Parser_1 )
void simpleLang() : {}              { integerLiteral()
                                    ( ( "+" | "-" ) integerLiteral() )? <EOF> }
void integerLiteral() : {Token t;} { t=<INT>
                                    { System.out.println("integer = "+t.image); }}
SKIP    : { " " | "\t" | "\n" | "\r" }
TOKEN   : { < INT : ( ["0" - "9"] )+ > }
```

请注意有关该文件的下述情形：

- PARSER_BEGIN 和 PARSER_END 伪指令指定了要生成的 Java 解析器的名称（Parser_1.java），并提供一个位置以便将 Java 语句插入该类。在这个案例中，您正将一个 Java package 语句放置在文件的顶部。该 package 语句也放置在 Java 助手类文件的顶部，该文件是作为生成过程一部分产生的（请参阅 JavaCC 编译过程 ）。尽管我在本示例中没有这样做，但是这也是一个声明实例变量的好场所，该实例变量将由您结果中的 Java 语句引用。如果您喜欢，甚至可以在这里插入 Java main() 过程，并且使用它来构建独立的应用程序，以启动和测试您正在生成的解析器。
- JavaCC 语法看上去象一个过程，而结果看上去非常象进行方法调用。这并非偶然。在 JavaCC 编译这个脚本时产生的 Java 源代码包含与这些结果具有相同名称的方法；这些方法在运行时按照它们在 .jj 脚本中调用的顺序执行。我将在 遍历解析代码中为您演示那是如何工作的。（这里的术语“编译”也不是偶然的 — 解析器生成器通常也称为“编译器的编译器”。）

- 花括号（{ 和 }) 内描述了结果的主体，并且排除了任何您正在嵌入的 Java 操作。请注意 integerLiteral 规则中用花括号包括的 System.out.println() 语句。该操作作为方法 Parser_1.integerLiteral() 的一部分产生。每当解析器遇到整数时，都执行该操作。
- 文件结尾的 SKIP 语句表明，在记号之间可以出现空白（空格、跳格、回车和换行），空白将被忽略。
- TOKEN 语句包含类似 regex 的表达式，该表达式描述了整数记号看起来象什么。在前面的结果中，对这些记号的引用是用尖括号括起来的。
- 第二个结果 integerLiteral() 声明了类型 Token （JavaCC 的内置类）的局部变量 t 。当在输入流中遇到整数时会 *触发* 该规则，该整数（象文本一样）的值被赋给实例变量 t.image 。另一个 Token 字段 t.kind 被赋值为一个枚举（enum），表明这个特殊的记号是一个整数，而不是解析器所知的另一种类型的记号。最后，在解析器中生成的 Java System.out.println() 代码可以在解析时在那个记号的内部使用 t.image 进行访问并且打印其文本值。

遍历解析器代码

让我们非常简要地了解一下您生成的解析器的内部原理。出于下面两个原因，稍微了解由特殊的 .jj 语法生成的方法以及其在运行时的执行顺序是很有用的：

- 有时候（特别是当您第一次做的时候），解析器似乎返回了不同的结果，而您认为是 .jj 文件指示它这样做的。您可以在运行时单步遍历产生的解析器代码，以便查看它到底在做什么，并相应地调整语法文件。我现在还经常这样做。
- 如果您知道结果 / 方法的执行顺序，那么您将对在脚本中的什么地方嵌入 Java 操作以获得特定的结果有更好的理解。在第 2 部分谈论有关 JJTree 工具和解析树表示时，我将回过头来更详细地讨论这一内容。

尽管深入研究已生成解析器的详细信息超越了本文的范围，但是 清单 2 还是显示了为方法 Parser_1.integerLiteral() 生成的代码。这可能会让您对最终代码看起来象什么有一些了解。特别需要注意方法中的最后一条语句： System.out.println("integer = "+t.image) 。该语句作为嵌入 .jj 脚本的 Java 操作发挥作用。

清单 2. Parser_1 中生成的方法

```
static final public void integerLiteral() throws ParseException {
                          Token t;
  t = jj_consume_token(INT);
  System.out.println( "integer = "+t.image);
}
```

以下高级、详尽的描述说明了这个解析器将做什么：

1. 最上面的方法 simpleLang() 调用 integerLiteral() 。
2. integerLiteral() 希望在输入流中立即遇到一个整数，否则该表达式将无效。为了验证这一点，它调用记号赋予器（Tokenizer.java）以返回输入流中的下一个记号。记号赋予器穿过输入流，每次检查一个字符，直到它遇到一个整数或者直至文件结束。如果是前者，则以 <INT> 记号将值“包”起来；如果是后者，则当作 <EOF> ；并将记号返回给 integerLiteral() 做进一步处理。如果记号赋予器未遇到这两个记号，则返回词法错误。
3. 如果记号赋予器返回的记号不是整数记号或 <EOF> ，那么 integerLiteral() 抛出 ParseException ，同时解析完成。
4. 如果它是整数记号，表达式仍然可能是有效的， integerLiteral() 再次调用记号赋予器以返回下一个记号。如果返回 <EOF> ，则由单个整数构成的整个表达式都是有效的，解析器将控制返还给调用应用程序。
5. 如果记号赋予器返回加号或减号记号，则表达式仍然是有效的， integerLiteral() 将最后一次调用记号赋予器，以寻找另一个整数。如果遇到一个整数，则表达式是有效的，解析器将完成工作。如果下一个记号不是整数，则解析器抛出异常。

注：如果解析器失败了，则抛出 ParseException 或 TokenMgrError 。任何一种异常都表明您的表达式是无效的。

这里的 要点 是，只有当解析器成功地遍历了嵌入 Java 操作的那部分结果后，才能执行嵌入到这两个结果

中的任何 Java 操作。如果将表达式“42 + 1”传递给该解析器，则语句 integer = 42 将被打印到控制台，后跟 integer = 1 。如果运行无效的表达式“42 + abc”，则产生消息 integer = 42 ，后跟 catch 块消息 a Token Manager error! 。在后一种情形中，解析器只成功地遍历了 simpleLang 结果中的第一个 integerLiteral() 项，而未成功遍历第二项：

```
void simpleLang() : {} { integerLiteral() ( ("+" | "-") integerLiteral() )? <EOF> }
```

换而言之，第二个 integerLiteral() 方法未被执行，因为未遇到希望的整数标记。

JavaCC 编译过程

当您对 .jj 文件运行 JavaCC 时，它会生成许多 Java 源文件。其中一个是主解析代码 Parser_1.java，当您有一个要解析的表达式时，您将从您的应用程序调用该代码。JavaCC 还创建了其它六个由解析器使用的辅助文件。JavaCC 总共生成了以下七个 Java 文件。前三个是特定于这个特殊语法的；后四个是通用的助手类 Java 文件，无论语法是怎么样的，都会生成这几个文件。

- Parser_1.java
- Parser_1Constants.java
- Parser_1TokenManager.java
- ParseException.java
- SimpleCharStream.java
- Token.java
- TokenMgrError.java

一旦 JavaCC 生成了这七个 Java 源文件，则可以编译它们并将它们链接到您的 Java 应用程序中。然后可以从您的应用程序代码调用新的解析器，从而将表达式传递给它进行求值。下面是一个样本应用程序，它实例化您的解析器，并且为它提供了一个硬连接在应用程序顶部的表达式。

清单 3：调用第一个解析器

```
package examples.example_1;
import examples.example_1.Parser_1;
import examples.example_1.ParseException;
import java.io.StringReader;
public class Example_1
{
    static String expression = "1 + 42";

    public static void main( String[] args )
    //--------------------------------------
    {
        new Example_1().parse( expression );
    }
    void parse( String expression )
    //-----------------------------
    {
        Parser_1 parser = new Parser_1( new StringReader( expression ));
        try
        {
            parser.simpleLang();
        }
        catch( ParseException pe )  {
            System.out.println( "not a valid expression" );
        }
        catch( TokenMgrError e ) {
            System.out.println( "a Token Manager error!" );
        }
    }
}
```

这里有什么是值得注意的呢？

1. 要调用 Parser_1 中的解析代码，需要调用该类中的方法 simpleLang() 。.jj 文件中的结果顺序通常是无关的，而本案例除外，在本案例中，语法中最顶部的结果名称用于调用解析器。
2. 如果正在传递给解析器代码的表达式不能根据语法合法地构造，则将抛出 ParseException 或 LexicalError 。
3. 如果表达式是有效的，则执行嵌入语法各部分的任何 Java 操作，这些语法部分都被成功遍历，就象遍历解析器代码结尾描述的一样。

结束语

这篇文章结束后还有第 2 部分。您将从类似的样本代码开始着手，学习如何使用 JavaCC 的“伙伴”工具 JJTree 来创建在运行时构建解析的解析树表示的解析器，而不是执行嵌入 .jj 脚本的操作。正如您将看到的，这有很多优点。

参考资料

- 您可以参阅本文在 developerWorks 全球站点上的 英文原文.
- 参与有关本文的 论坛。（您也可以单击文章顶部或底部的 **讨论**以访问论坛。）
- 希望了解更多有关 BNF 的知识吗？请查阅 Wikipedia.org。
- 请查阅免费的（但非开放源码） JavaCC 和 JJTree 分发版。
- 在 XML Query 主页可找到更多有关 W3C 的 XQuery 和 XPath 规范的信息。
- XQEngine是作者开发的基于 Java 的 XQuery 引擎的开放源码实现。
- 在 developerWorks XML和 Java 技术专区可找到本文所涵盖的技术的更多信息。
- IBM WebSphere Studio提供了一组使 XML 开发自动化的工具，可使用 Java 也可使用其它语言。它与 WebSphere Application Server进行了紧密的集成，但是也可以与其它 J2EE 服务器一起使用。
- 了解如何才能成为 XML 和相关技术的 IBM 认证开发人员。

关于作者

Howard Katz 居住在加拿大温哥华，他是 Fatdog Software 的唯一业主，该公司专门致力于开发搜索 XML 文档的软件。在过去的大约 35 年里，他一直是活跃的程序员（一直业绩良好），并且长期为计算机贸易出版机构撰写技术文章。Howard 是 Vancouver XML Developer's Association 的共同主持人，还是 Addison Wesley 即将出版的书籍 *The Experts on XQuery*的编辑，该书由 W3C 的 Query 工作组成员合著，概述了有关 XQuery 的技术前景。他和他的妻子夏天去划船，冬天去边远地区滑雪。可以通过 howardk@fatdog.com与 Howard 联系。

☐ 保持登录。

单击提交则表示您同意developerWorks 的条款和条件。 使用条款

提交 取消

当您初次登录到 developerWorks 时，将会为您创建一份概要信息。**您在 developerWorks 概要信息中选择公开的信息将公开显示给其他人，但您可以随时修改这些信息的显示状态。**您的姓名（除非选择隐藏）和昵称将和您在 developerWorks 发布的内容一同显示。

所有提交的信息确保安全。

关闭 [x]

请选择您的昵称：

当您初次登录到 developerWorks 时，将会为您创建一份概要信息，您需要指定一个昵称。您的昵称将和您在 developerWorks 发布的内容显示在一起。

昵称长度在 3 至 31 个字符之间。 您的昵称在 developerWorks 社区中必须是唯一的，并且出于隐私保护的原因，不能是您的电子邮件地址。

昵称： （长度在 3 至 31 个字符之间）

单击**提交**则表示您同意developerWorks 的条款和条件。 使用条款.

提交 取消

所有提交的信息确保安全。

平均分 (6个评分)

1 星

1 星
2 星

2 星
3 星

3 星
4 星

4 星
5 星

5 星

提交

添加评论：

请 登录 或 注册 后发表评论。

注意：评论中不支持 HTML 语法

□有新评论时提醒我剩余 1000 字符

发布

快来添加第一条评论