GrammarProposal

[Draft]

Abstract

Grammar is a pure java syntax analyzer library.

Proposal

Grammar library can be use with a simple maven dependency. Grammar are defined with terminals, non-terminals and rules. For example an expression is a non-terminal, a number is a terminal and a rule is 'expression -> expression + number'. You write the rule content in a java file, and you can attach some data to terminals, non-terminals. Data attached to terminals are provided by some lexer, whereas data attached to non-terminals are generated using a method attached to rule. All terminals, non-terminals and rules are java object you will give to one of provided grammars.

Background

Most of java compilation tools ask to define your grammar is a special file. Then you will run the tool on this special file and you will obtain a java source file. Usually the special file contains some piece of pseudo java code (with special token like $n to access to information attached to the n token of the rule). However SableCC propose another approach : no pseudo java code in special file, you will complete generated files after. I think that this approach is cleaner.

Rationale

Because of pre-compilation grammar are not enough use in refractor tools or analyze tools, we prefer to use regular expression. Regular expression are very powerful however they becomes really difficult to understand and maintain when we want to recognize some complex informations.

With a dynamically generated grammar I think refractor tools or analyze tools could be enhanced (if the tools provide you predefined terminals and ask you to define what you need to recognize).

Moreover dynamically generated grammar allow some rules reusability. For example (if then else), (while), (for) can be part of grammar reusable from one language to another. The condition to allow the reusability is to accept different terminals for block start (a '{' in java or 'then' in lua), block stop ... We can also share grammar construction for boolean or arithmetic expressions.

With dynamically generated we can also imagine dynamically grammar transformer, for example the end user decide if he want multiplication having higher priority than addition or not then the grammar is generated (without pre-compilation issues).

In last, the code is not so hard to understand. Java brings Set, List and Map implementations and they are widely use to perform grammar automate construction.

Current Status

Look for a community.

Meritocracy

TODO

Community

A single developer.

Core Developers

Gael Lalire starts to write the code.

Alignment

Apache provide widely used utilities, this project may add a missing utility.

Known Risks

Orphaned Projects

Because there is only one committer, there is a risk of being orphaned.

Inexperience With Open Source

TODO

Reliance On Salaried Developers

The project development occurs in volunteer time. No corporations is sponsoring it.

Relationships with Other Apache Products

As an utility project, grammar should not have many dependencies. In initial code there is no dependency (only JDK 5 classes). Grammar could use Log4J.

Initial Source

http://commons-grammar.googlecode.com/svn/trunk/

Required Resources

Mailing lists, Subversion Directory and Issue Tracking if this project should be a TLP. However commons project could be the right place for such project.

Initial Committers

Gael Lalire (gael dot lalire at gmail dot com)

Sponsors

Volunteers, please.

Champion

Volunteers, please.

Mentors

Volunteers, please.

Sponsoring Entity

TODO