

# LINQExpand4Java

## A compiler plugin for AST rewriting

If the designers of query languages have worked hard to make them readable, why not let developers include verbatim LINQ query snippets in Java?

A compiler plugin (based on JSR-269) can intercept a Java parse tree, modify it, and let the Java compiler process the updated tree.

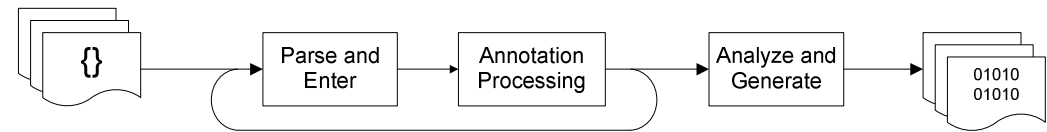
### Advantages of *Lightweight Language Nesting:*

the original syntax of LINQ is supported without extending the Java grammar

far less complex than bytecode manipulation. In fact, not even necessary to build AST nodes yourself: just let the parser run again

the same error reporting mechanism of the Java compiler is re-used for ill-formed LINQ queries

applicable to other query languages (OQL, XQuery, JPQL, SQL)



before ... ... after

|  |  |
|--|--|
| <pre> 1 package pluginTester; 2 import plugin.marker.LINQ; 3 public class Example 4 { 5     public Example() { 6         super(); 7         System.out.println("FOO"); 8         System.out.println("bar"); 9     } 10    public static void main(String[] args) 11    { 12        System.out.println("bar"); 13    } 14 } 15 @plugin.LINQ 16 public void thisContainsMyQuery() 17 { 18     System.out.println("FOO"); 19     LINQ.expand("from x1 in e1 select y"); 20 } 21 } 22 }           </pre> | <pre> 1 package pluginTester; 2 3 public class Example { 4 5     public Example() { 6         super(); 7         System.out.println("FOO"); 8         System.out.println("bar"); 9     } 10 11    public static void main(String[] args) { 12        System.out.println("bar"); 13    } 14 15    @plugin.LINQ 16    public linqtextual.MethodCall thisContainsMyQuery() { 17        System.out.println("FOO"); 18        LINQ.expanded("from x1 in e1 select y!"); 19        linqtextual.impl.LinqtextualFactoryImpl factory = new linqtextual.impl.LinqtextualFactoryImpl(); 20        linqtextual.Literal literal0 = factory.createLiteral(); 21        literal0.setKind(linqtextual.LiteralKind.ID_LITERAL); 22        literal0.setNonDateTimeLit("e1"); 23        linqtextual.Literal literal1 = factory.createLiteral(); 24        literal1.setKind(linqtextual.LiteralKind.ID_LITERAL); 25        literal1.setNonDateTimeLit("y"); 26        linqtextual.LambdaExpr lambda0 = factory.createLambdaExpr(); 27        lambda0.getParams().add("x1"); 28        lambda0.setBody(literal1); 29        linqtextual.MethodCall select0 = factory.createMethodCall(); 30        select0.setReceiver("select"); 31        select0.getArgs().add(literal0); 32        select0.getArgs().add(lambda0); 33        return select0; 34    } 35 }           </pre> |
|--|--|

Now that we are talking about compiler plugins.

Did you know that the Scala compiler supports a richer extension model than its Java counterparts?

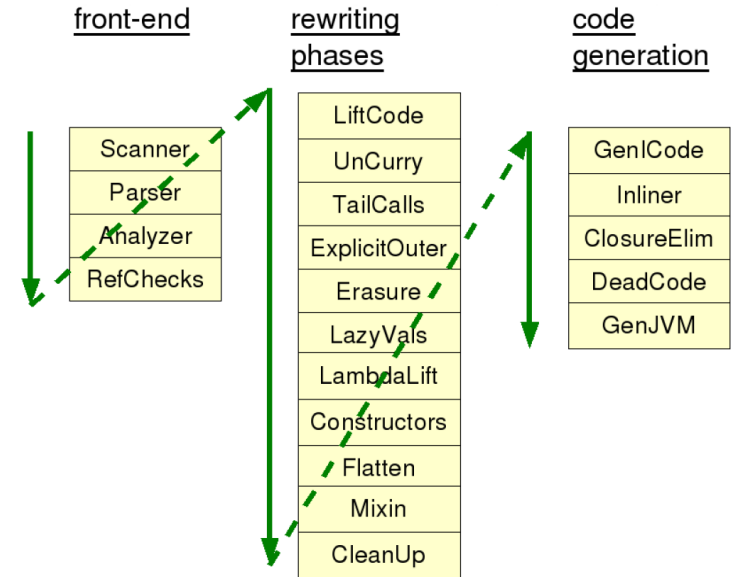
Essentially the same kind of AST transformations:

```

(a) T1: inline query continuation
    from x1 in e1 ... into x2 ...
    → from x2 in (from x1 in e1 ...) ...
-----
(b) function T1 ( q : QueryExp ) : QueryExp
    when q.qbody.qcont ≠ ∅
-----
    new QueryExp {
      from = new FromClause {
        type = q.qbody.qcont.type,
        var = q.qbody.qcont.var,
        in = new QueryExp {
          from = q.from
          qbody = new QueryBody {
            qbclauses = q.qbody.qbclauses,
            sel_gby = q.qbody.sel_gby
          }
        }
      }
      qbody = q.qbody.qcont.qbody
    }
  
```

Only that the output is Scala syntax (in general, for further processing, as part of *staged rewriting*)

|       |   |
|-------|---|
| T10   | From followed by a join with into followed by a select  |
| LINQ  | <code>from x1 in e1<br/>join x2 in e2 on k1 equals k2 into g<br/>select e3</code>   |
| SQO   | <code>e1.GroupJoin( e2,<br/>                x1 =&gt; k1, x2 =&gt; k2,<br/>                (x1, g) =&gt; e3 )</code>           |
| Scala | <code>for ( x1 &lt;- e1; val outerKey = k1;<br/>      val g = e2 filter { x2 =&gt; outerKey == k2 }<br/>    ) yield e3</code> |



Work in progress:  
*Transforming Scala into a full-fledged DB Programming Language*

Yes, we're after  
*Orthogonal Persistence*,  
 in the spirit of PJama or Oberon-D

Only that using a modern, optimizing,  
 database backend  
 (>1 candidate)

Further info:  
<http://www.sts.tu-harburg.de/people/mi.garcia>