Tinytemplate Aspect Generator

Jesper Öqvist

Department of Computer Science
Lund University

May 6, 2014

The Problem

- Writing pretty printing code is tedious
- The resulting code is verbose and uninteresting

The Problem

Typical pretty printing of AST:

```
void ConditionalExpr.prettyPrint(StringBuffer sb) {
  getCondition().prettyPrint(sb);
  sb.append(" ? ");
  getTrueExpr().prettyPrint(sb);
  sb.append(" : ");
  getFalseExpr().prettyPrint(sb);
}

void Modifiers.prettyPrint(StringBuffer sb) {
  for (int i = 0; i < getNumModifier(); i++) {
    getModifier(i).prettyPrint(sb);
    sb.append(" ");
  }
}
```

The Problem

Note the common pattern:

- print child nodes
  `getCondition().prettyPrint(sb);`
- interspersed with operators/keywords
  `sb.append(":");`
- some loops
- some conditionals (for optional child components)

A Solution

Tinytemplate has:

- Variable expansion
- Conditional expansion
- List concatenation

These can be mapped to components in the pretty printing pattern!

aspectgen.jar

```
$ ant jar
$ java -jar aspectgen.jar PrettyPrint.tt
```

Generates a JastAdd aspect from the template definitions.

Template Declarations

Print a simple keyword:

```
BooleanType [[boolean]]
```

Generated code:

```
BooleanType.prettyPrint(PrettyPrinter out) {
  out.print("boolean");
}
```

PrettyPrinter

`PrettyPrinter`

- Simple helper class for printing to some stream
- Can print strings and newlines and things implementing interface `PrettyPrintable`
- Tracks indentation

Printing Children

Printing children:

```
SwitchStmt [[switch ($Expr) $Block]]
```

Generated code:

```
SwitchStmt.prettyPrint(PrettyPrinter out) {
  out.print("switch (");
  out.print(getExpr());
  out.print(") ");
  out.print(getBlock());
}
```

Conditionals

Conditional printing:

```
BreakStmt [[break$if(hasLabel) $Label$endif;]]
```

Generated code:

```
BreakStmt.prettyPrint(PrettyPrinter out) {
  out.print("break");
  if (hasLabel()) {
    out.print(" ");
    out.print(getLabel());
  }
  out.print(";");
}
```

List Concatenation

List concatenation:

```
Modifiers [[$cat(ModifierList," ")]]
ArrayInit [[{ $cat(InitList,", ") }]]
```

Generated code:

```
Modifiers.prettyPrint(PrettyPrinter out) {
  out.cat(getModifierList(), " ");
}
ArrayInit.prettyPrint(PrettyPrinter out) {
  out.print("{ ");
  out.cat(getInitList(), ", ");
  out.print(" }");
}
```

Indentation

```
Block [[
$if(hasStmts)
{
  $cat(StmtList,"\n")
}
$else
{ }
$endif]]
```

Generated code:

```
if (hasStmts()) {
  out.print("{");
  out.println();
  out.indent(1);
  out.cat(getStmtList(), "", "");
  out.println();
  out.print("}");
} else ...
```

Case Study

MiniJ (pretty printing extension of JastAddJ Java4 frontend):

- 144 lines library code in `PrettyPrinter.java`
- 212 lines of helper RAG code
- 176 lines (incl empty lines) of template code
- Generated aspect is 498 lines (760 ish with alternate concat generation)
- `PrettyPrint.jadd` in current JastAddJ is 885 lines

Remaining Problems

- Some things too painful in the limited template syntax
- Generating the aspect for each build is redundant
- Yet another tool in the build process :(