Unparsing types the Scaladoc way

© Miguel Garcia, LAMP, EPFL
`http://lamp.epfl.ch/~magarcia`

January 2nd, 2011

Abstract

Unparsing, as implemented by `scala.tools.unparse`, makes explicit the desugaring, code expansion ("*adding synthetics*"), and type inference that the `scalac` compiler routinely performs. A close relative of unparsing, pretty-printing, usually results in less detailed output, as for example inferred types are not made explicit (that's the terminology we follow in this write-up).

A tool that already shows detailed types is *Scaladoc* [1], moreover taking into account any type instantiations and type refinements that might be in effect (unlike Javadoc, where "*if method `C:m` returns type `T`, subclasses of `C` will always [be displayed with that] parent signature, even if they instantiate `T` to a concrete type such as `Integer`*")

We take a look at the "type unparsing" capabilities of Scaladoc. First, by using Scaladoc as a library from within a compiler plugin (our unparser). After that, we duplicate that functionality without running Scaladoc at all.

In spite of its focus, this write-up might also be of interest to those interested in the internal workings of API documentation tools in general (e.g., see also Collaborative Scaladoc, `http://code.google.com/p/collaborative-scaladoc/`), or code browsing tools (e.g., Scala X-Ray, `http://www.scala-lang.org/node/1509`).

Contents

1 Background

Build and run instructions for the compiler plugin described in these notes can be found in Sec. 1 of `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/2010Q4/Unparsing.pdf`

2 A tutorial about types in ASTs, from mailing list posts

2.1 Some terminology

Quoting from `http://www.scala-lang.org/node/7796`

> *Everything in the compiler divides into terms and types.* `TermSymbol` *is the base class for terms and* `TypeSymbol` *for types.* `ClassSymbol` *is a* `TypeSymbol` *...*`BooleanClass` *is one of the symbols set up right at the beginning, it's defined in* `Definitions.scala` *and you can count on it having the type booleans have since it is what defines it.*

```
scala> global.definitions.ScalaValueClasses
res2: List[global.Symbol] = List(class Unit, class Byte, class Short, class Int, class Long, class Char, cl

scala> global.definitions.ScalaValueClasses map (_.tpe)
res3: List[global.Type] = List(Unit, Byte, Short, Int, Long, Char, Float, Double, Boolean)
```

> *Now just like symbols have types, types have symbols:*

```
scala> res3 map (_.typeSymbol)
res4: List[global.Symbol] =
  List(class Unit, class Byte, class Short, class Int, class Long,
       class Char, class Float, class Double, class Boolean)
```

> *In this case with a pretty obvious relationship. Type application:*

```
scala> definitions.optionType(definitions.BooleanClass.tpe)
res11: global.Type = Option[Boolean]
```

> *Notice the typeSymbol is the symbol of the type constructor: types can tell you more than symbols.*

```
scala> res11.typeSymbol
res12: global.Symbol = class Option
```

> *Types have type members, which are symbols:*

```
scala> val listType = global.definitions.getClass("scala.collection.immutable.List").tpe
listType: global.Type = List
scala> listType.member("drop")
res13: global.Symbol = value drop
scala> res13.tpe
res14: global.Type = (n: Int)List (n: Int)scala.collection.LinearSeqOptimized (n: Int)java.lang.Object
```

And etc. etc. It's all pretty discoverable if you use the repl. (Up to a point anyway.)

> Is there any place online where I can find more about the compiler
> source? Maybe other compiler plugins that would be good examples
> in this case?

Everything I know I figured out from the source. Maybe you'll have better luck finding useful bits outside the source. Use power mode in the repl as I have here. See my past messages to scala-internals. Not that I'm the expert (far from it) but it's slim pickings for recent compiler docs.

More details at `http://www.scala-lang.org/node/598`

2.2 `tpe.narrow`

Quoting from `http://www.scala-lang.org/node/8311`

```
% scala29
Welcome to Scala version 2.9.0.r23657-b20101202094630 (Java HotSpot(TM) 64-Bit Server VM, Java 1.6.0_22).
Type in expressions to have them evaluated.
Type :help for more information.
scala> :power
** Power User mode enabled - BEEP BOOP **
** scala.tools.nsc._ has been imported **
** New vals! Try repl, global, power **
** New cmds! :help to discover them **
** New defs! Type power. to reveal **
scala> import global._
import global._
scala> atPhase(currentRun.typerPhase)(definitions.ListModule.tpe)
res0: global.Type = object List
scala> atPhase(currentRun.typerPhase)(definitions.ListModule.tpe.narrow)
res1: global.Type = scala.collection.immutable.List.type
```

2.3 Contexts

Quoting from `http://comments.gmane.org/gmane.comp.lang.scala/21087`

```
scala> :power
scala> import global._
scala> power.mkContext()
res0: power.compiler.analyzer.Context = [...]

scala> res0.imports
res1: List[power.compiler.analyzer.ImportInfo] = List(import scala.Predef._, import scala._, import java.lang._

scala> res1 flatMap (_.allImportedSymbols) sortBy (_.toString)
res2: List[power.compiler.analyzer.global.Symbol] =
       List(class <byname>, class <equals>, class <repeated...>, class <repeated>,
            class AbstractMethodError, class AbstractStringBuilder, class Annotation, class Any, class AnyVal,
            class Appendable, class Application, class ApplicationShutdownHooks, class ApplicationShutdownHooks$
            class ArithmeticException, class Array, class Array$, class Array$$anon$2,
            class Array$$anonfun$apply$1, class Array$$anonfun$apply$10, class Array$$anonfun$apply$2,
            class Array$$anonfun$apply$3, class Array$$anonfun$apply$4, class Array$$anonfun$apply$5,
            class Array$$anonfun$apply$6, class Array$$anonfun$apply$7, class Array$$anonfun$apply$8,
            class Array$$anonfun$apply$9, class Array$$anonfun$concat$1,
            class Array$$anonfun$concat$2, class Array$$anonfun$fill$1, ...

scala> res2 filter (x => x.isImplicit && x.tpe.resultType <:< definitions.IntClass.tpe) foreach println
method byte2int
method char2int
method short2int
```

3 Running Scaladoc as a library

Quoting from "*Scaladoc 2 Design*", `http://lampsvn.epfl.ch/trac/scala/wiki/Scaladoc/Design`:

> *The model extractor transforms the symbol table left by the compiler into a new representation of the program's structure, called the "model". The model is similar to the symbol table in that it represents all elements (classes, objects, methods, values, types, etc.) of a program, which it encodes as entities.*

Moreover, after invoking Scaladoc as a library (Sec. 3.1), we can query its *template cache*: we provide a template symbol and get back (see also Figure 1):

- a `doc.model.Trait`:

  ```
  trait Trait extends DocTemplateEntity with HigherKinded
  ```

  - which has as subtype:

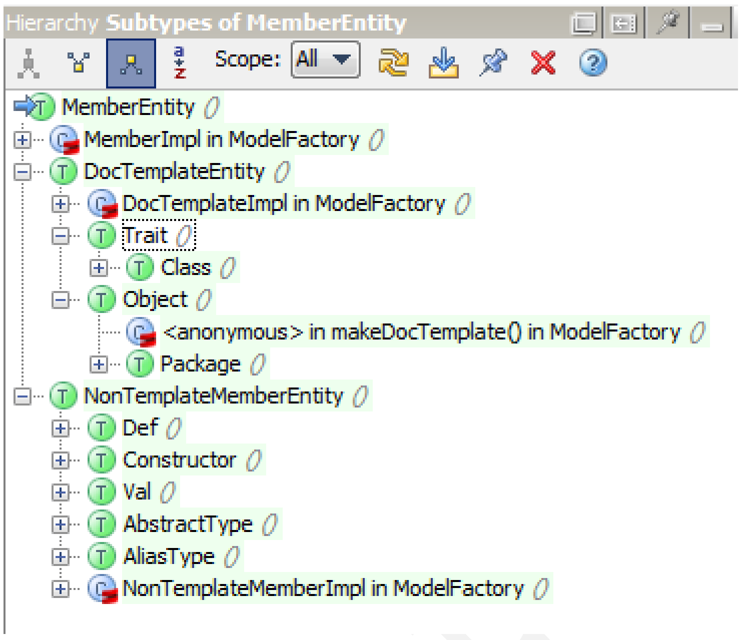
    ```
    /** A ''documentable'' class. */
    trait Class extends Trait with HigherKinded {
      def primaryConstructor: Option[Constructor]
      def constructors: List[Constructor]
      def valueParams: List[List[ValueParam]]
    }
    ```

- a `doc.model.Object`:

  ```
  trait Object extends DocTemplateEntity
  ```

  - which has as subtype:

    ```
    /** A package that contains at least one ''documentable'' class, trait,
      * object or package. */
    trait Package extends Object {
      def inTemplate: Package
      def toRoot: List[Package]
      def packages: List[Package]
    }
    ```



```
MemberEntity
├─ MemberImpl in ModelFactory
├─ DocTemplateEntity
│  ├─ DocTemplateImpl in ModelFactory
│  ├─ Trait
│  │  └─ Class
│  └─ Object
│     ├─ <anonymous> in makeDocTemplate() in ModelFactory
│     └─ Package
└─ NonTemplateMemberEntity
   ├─ Def
   ├─ Constructor
   ├─ Val
   ├─ AbstractType
   ├─ AliasType
   └─ NonTemplateMemberImpl in ModelFactory
```

Figure 1: `MemberEntity` hierarchy, Sec. 3

3.1 Running Scaladoc from within a compiler plugin

We need just one additional helper class (`ModelFactoryUnparse`, Listing 1) as well as the following helper method:

```
def factoryAndUniverse(): Option[(ModelFactoryUnparse, nsc.doc.Universe)] = {
   val sless = new nsc.doc.SourcelessComments {
     val global: ScalaDocModel.this.global.type = ScalaDocModel.this.global
   }
   global.docComments ++= sless.comments
   if (!reporter.hasErrors) {
     val docSettings: nsc.doc.Settings = new nsc.doc.Settings(settings.errorFn)
     val modelFactory = (new ModelFactoryUnparse(docSettings)
             with nsc.doc.model.comment.CommentFactory with nsc.doc.model.TreeFactory)
     val madeModel = modelFactory.makeModel
     modelFactory.digStructureOutOfRefinements()
     Some(modelFactory, madeModel)
   }
   else None
 }
```

An invocation for illustration purposes (which otherwise does nothing useful):

```
 private def testScalaDocModel() {
   val Some((modelFactory, theModel)) = factoryAndUniverse()
   val allDocTpls = modelFactory.getTemplates.toArray
   val clsTpls     = allDocTpls collect { case c : nsc.doc.model.Class => c }
   for (tpl <- clsTpls) {
     scala.Console.println("Class " + tpl.toString)
     for (m <- tpl.methods) {
       scala.Console.println("\tmethod " + m.toString)
       for (vps <- m.valueParams; vp <- vps) {
```

```
     scala.Console.println("\t\tvp " + vp.name + " resultType = " + vp.resultType)
    }
   }
  }
  val at = modelFactory.findTemplate("A")
  scala.Console.println(at.toString)
 }
```

3.2 What TemplateEntitys are made of

Most `make...` methods in `ModelFactory.scala` expect an

```
inTpl: => PackageImpl
```

which is not available when documenting the `RootPackage`, so in that case a `RootPackageImpl` instance is used by `makeRootPackage`. The `RootPackageEntity` trait would act as a "marker interface" (only) if it were not for a refinement (shown two listings below).

```
abstract class RootPackageImpl(sym: Symbol) extends PackageImpl(sym, null) with RootPackageEntity

// and due to import model.{ RootPackage => RootPackageEntity }

/** A package represent the root of Entities hierarchy */
trait RootPackage extends Package
```

The refinement below prevents a `RootPackageImpl` from behaving just like a `PackageImpl(sym, null)`:

```
 /** Creates a package entity for the given symbol or returns 'None' if the symbol does not denote a package tha
  * contains at least one ''documentable'' class, trait or object. Creating a package entity */
 def makePackage(aSym: Symbol, inTpl: => PackageImpl): Option[PackageImpl] = {
  val bSym = normalizeTemplate(aSym)
  if (templatesCache isDefinedAt (bSym))
   Some(templatesCache(bSym) match {case p: PackageImpl => p})
  else {
   val pack =
    if (bSym == RootPackage)
     new RootPackageImpl(bSym) { /*- start of refinement for RootPackage */
      override val name = "root"
      override def inTemplate = this
      override def toRoot = this :: Nil
      override def qualifiedName = "_root_"
      override def inheritedFrom = Nil
      override def isRootPackage = true
      override protected lazy val memberSyms =
       (bSym.info.members ++ EmptyPackage.info.members) filter { s =>
        s != EmptyPackage && s != RootPackage
       }
     }
    else
     new PackageImpl(bSym, inTpl) {} /*- NO refinement for non-root package */
   if (pack.templates.isEmpty) {
    droppedPackages += 1
    None
   }
   else Some(pack)
  }
```

```
}
```

```
def normalizeTemplate(aSym: Symbol): Symbol = aSym match {
  case null | EmptyPackage | NoSymbol =>
    normalizeTemplate(RootPackage)
  case ScalaObjectClass | ObjectClass =>
    normalizeTemplate(AnyRefClass)
  case _ if aSym.isModuleClass || aSym.isPackageObject =>
    normalizeTemplate(aSym.sourceModule)
  case _ =>
    aSym
}
```

4 Unparsing type references

TODO

References

[1] Gilles Dubochet and Donna Malayeri. Improving API Documentation for Java-like Languages. In *Evaluation and Usability of Programming Languages and Tools*, 2010. http://infoscience.epfl.ch/record/151520/files/plateau10-dubochet.pdf.

Listing 1: `ScalaDocModel`

```scala
package scala.tools
package unparse

import scala.collection._

trait ScalaDocModel {

  this: jdk2ikvm.JDK2IKVM with UnparseTreeFolding with ScalaDocModel =>

  import global._

  class ModelFactoryUnparse(docSettings: nsc.doc.Settings) extends nsc.doc.model.ModelFactory(global, docSettings) {

    thisFactory: nsc.doc.model.ModelFactory with nsc.doc.model.comment.CommentFactory with nsc.doc.model.TreeFactory =>

    override val global: ScalaDocModel.this.global.type = ScalaDocModel.this.global

    type ActualTypeOfDef = NonTemplateParamMemberImpl with HigherKindedImpl with nsc.doc.model.Def
    type ActualTypeOfConstructor = NonTemplateParamMemberImpl with nsc.doc.model.Constructor
    type ActualTypeOfValueParam = ParameterImpl with nsc.doc.model.ValueParam
    type ActualTypeOfTypeParam = ParameterImpl with HigherKindedImpl with nsc.doc.model.TypeParam /* with private TypeBoundsImpl */

    val packagesMap = mutable.Map.empty[nsc.Global#Symbol, nsc.doc.model.Package] // includes RootPackage
    val classesMap = mutable.Map.empty[nsc.Global#Symbol, nsc.doc.model.Class] // includes case classes
    val traitsMap  = mutable.Map.empty[nsc.Global#Symbol, nsc.doc.model.Trait]
    val objectsMap = mutable.Map.empty[nsc.Global#Symbol, nsc.doc.model.Object] // TODO are package objects included here?

    val methodsMap     = mutable.Map.empty[nsc.Global#Symbol, ActualTypeOfDef]
    val constructorsMap = mutable.Map.empty[nsc.Global#Symbol, ActualTypeOfConstructor]
    val valueParamsMap = mutable.Map.empty[nsc.Global#Symbol, ActualTypeOfValueParam]
    val typeParamsMap  = mutable.Map.empty[nsc.Global#Symbol, ActualTypeOfTypeParam]

    val valdefsym2vpsym = mutable.Map.empty[nsc.Global#Symbol, nsc.Global#Symbol]

    def digStructureOutOfRefinements() {
      // track symbols for packages, classes, traits, and objects.
      for ((k, v) <- templatesCache) {
        v match {
          case p : nsc.doc.model.Package => packagesMap.update(k, p)
          case c : nsc.doc.model.Class => classesMap.update(k, c)
          case t : nsc.doc.model.Trait => traitsMap.update(k, t) // Trait test must come after Class test
          case o : nsc.doc.model.Object => objectsMap.update(k, o) // Obect test must come after Package test
        }
      }
      // track symbols for methods and constructors
      for (docTpl <- templatesCache.values; mTpl <- docTpl.methods) {
        mTpl match { case m : ActualTypeOfDef => methodsMap.update(m.sym, m) }
      }
      for (cTpl <- classesMap.values; constrTpl <- cTpl.constructors) {
        constrTpl match { case constr : ActualTypeOfConstructor => constructorsMap.update(constr.sym, constr) }
      }
      // track symbols for value params
      for(mTpl <- methodsMap.values; vparams <- mTpl.valueParams; vpTpl <- vparams) {
        vpTpl match { case vp : ActualTypeOfValueParam => valueParamsMap.update(vp.sym, vp) }
      }
      // track symbols for type params
      // TODO ActualTypeOfTypeParam
    }

    /** Scaladoc uses as keys for the value params those from DefDef.sym.paramss.
        However when unparsing we have only the ValDef.symbol for the visited ValDef. */
    def linkTreeSymbols(dd: DefDef) {
      val infoparamss: List[List[Symbol]] = dd.symbol.paramss
      val valdefss: List[List[ValDef]] = dd.vparamss
      for((infoparams, valdefs) <- (infoparamss zip valdefss); (infoparam, valdef) <- (infoparams zip valdefs)) {
        assert(!modelFactory.valdefsym2vpsym.contains(valdef.symbol))
        modelFactory.valdefsym2vpsym.update(valdef.symbol, infoparam)
      }
    }

  }

  . . .

  lazy val Some((modelFactory, theModel)) = factoryAndUniverse()

  def unparseValueParamType(valdefsym: nsc.Global#Symbol) : String = {
    val vpsym = modelFactory.valdefsym2vpsym(valdefsym)
    val vp : ScalaDocModel#ModelFactoryUnparse#ActualTypeOfValueParam = modelFactory.valueParamsMap(vpsym)
    val te : nsc.doc.model.TypeEntity = vp.resultType
    te.toString // TODO print the FQN given the TypeEntity's refEntity
  }

}
```