scala.meta

Semester projects for Fall 2015

Eugene Burmako (@xeno_by)

École Polytechnique Fédérale de Lausanne
http://scalameta.org/

19 May 2014

What do we do?

Metaprogramming is...

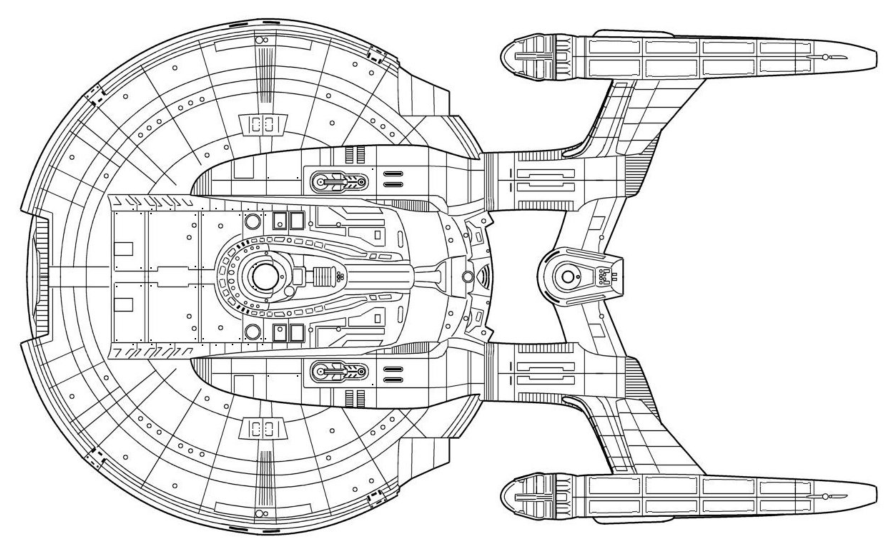
> *Metaprogramming is the writing of computer programs that write or manipulate other programs or themselves as their data.*
>
> —Wikipedia

Notable metaprograms

- Compilers
- Code generators
- IDEs

Before Scala 2.10

We are building a better tech



Why would you care?

Reason #1: Innovate in programming languages

```
val usersMatching = query[String, (Int, String)](
  "select id, name from users where name = ?")
usersMatching("John")
```

- Database queries can be written in SQL

Reason #1: Innovate in programming languages

```
val usersMatching = query[String, (Int, String)](
  "select id, name from users where name = ?")
usersMatching("John")

case class User(id: Column[Int], name: Column[String])
users.filter(_.name === "John")
```

- Database queries can be written in SQL
- They can also be written in a DSL, at times slightly awkward

Reason #1: Innovate in programming languages

```
val usersMatching = query[String, (Int, String)](
  "select id, name from users where name = ?")
usersMatching("John")

case class User(id: Column[Int], name: Column[String])
users.filter(_.name === "John")

case class User(id: Int, name: String)
users.filter(_.name == "John")
```

- Database queries can be written in SQL
- They can also be written in a DSL, at times slightly awkward
- Or they can be written in Scala and virtualized by a macro

Reason #1: Innovate in programming languages

```
trait Query[T] {
  def filter(p: T => Boolean): Query[T] = macro ...
}

val users: Query[User] = ...
users.filter(_.name == "John")
```

```
Query(Filter(users, Equals(Ref("name"), Literal("John"))))
```

- The `filter` macro takes an AST corresponding to the predicate
- This AST is then analyzed and transformed into a query fragment
- C# and F# need dedicated language features for this, we don't!

Reason #2: Implement stuff that people will use

```
ClassDef(
  NoMods,
  newTypeName("D"),
  Nil,
  Template(
    List(Ident(newTypeName("C"))),
    emptyValDef,
    List(DefDef(NoMods, nme.CONSTRUCTOR, ...))))
```

- Once, the most reliable way to do ASTs was via vanilla datatypes
- Above you can see what it took to say `class D extends C`
- Needless to say, only brave souls cared to use such an API

Reason #2: Implement stuff that people will use

```
val tree = q"class D extends C"
val q"class $_ extends ..$parents" = tree
```

- Then we introduced quasiquotes, a template-based approach to ASTs
- They became wildly popular even before an official release
- And the biggest feature of Scala 2.11
- A fun fact: quasiquotes were a semester project at LAMP.
  Thanks, Denys, for your hard work!

Reason #2: Implement stuff that people will use

- Another great example is work done by Martin Duhem
- When he started tinkering with SBT, macro support was pretty subpar
- A couple months in, Martin managed to make things much better and his improvements were already included in SBT 0.13.5

Reason #3: Influence future developments in Scala

- AST persistence was once a cute little idea
- We saw and appreciated its potential and decided to dig in
- Of course, as a semester project here at LAMP!

Reason #3: Influence future developments in Scala

- Adrien Ghosn & Mathieu Demarne approached the project in a very creative way
- Thanks to awesome prototype and great presentation, they've managed to convince people that AST persistence is practical
- The seeds were planted, and in the autumn we had a breakthrough!

Reason #3: Influence future developments in Scala

scala-internals ›

Attacking binary compatibility at the root with typed trees

37 posts by 17 authors

martin Oct 9

Scala has suffered from binary incompatibility problems ever since it has become moderately popular. Sure, we have made progress. There's now a tool (MiMa) to check for compatibility violations and a policy to maintain strict binary compatibility for minor versions of Scala. At the same time, the burden of maintaining binary compatibility makes bug fixes and library and compiler evolution much more cumbersome than before. It can now take 2 years or more to get a bugfix in which would break binary compatibility. And an increasing proportion of our effort going into a fix or improvement is spent on questions of maintaining binary compatibility. So, we are between a rock and a hard place: not binary compatible enough for the public at large yet already greatly slowed down by our own efforts to maintain the level of bc that we have achieved so far.

What projects do we have?

Summary

- AST interpreter
- Obey: nextgen style checker and rewriting
- Metadoc: documentation tool from the future