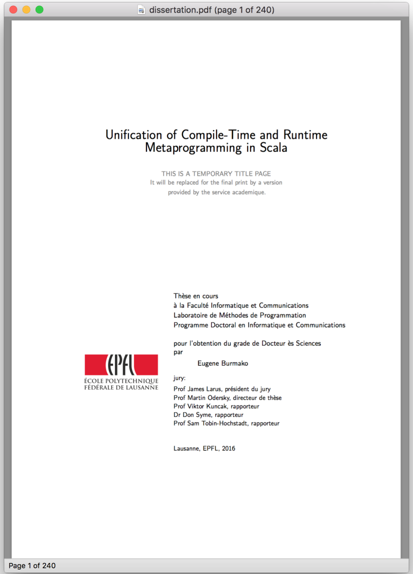
Scala.meta semantic API

Eugene Burmako (@xeno_by)

2 March 2017

What is scala.meta?

scala.meta is a cutting edge research



Unification of Compile-Time and Runtime Metaprogramming in Scala

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version provided by the service academique.

Thèse en cours
à la Faculté Informatique et Communications
Laboratoire de Méthodes de Programmation
Programme Doctoral en Informatique et Communications

pour l'obtention du grade de Docteur ès Sciences
par

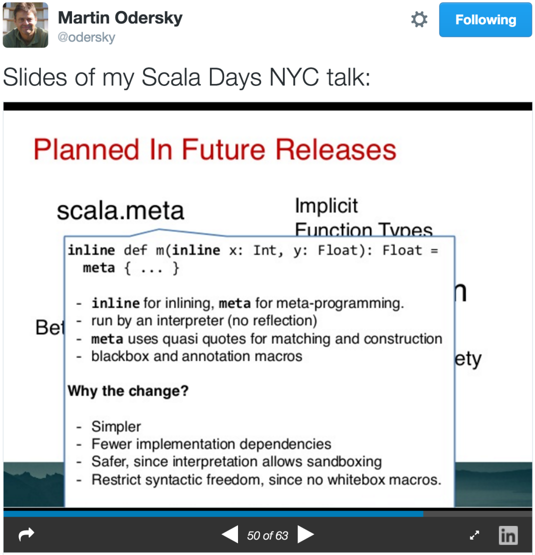
Eugene Burmako

jury:
Prof James Larus, président du jury
Prof Martin Odersky, directeur de thèse
Prof Viktor Kuncak, rapporteur
Dr Don Syme, rapporteur
Prof Sam Tobin-Hochstadt, rapporteur

Lausanne, EPFL, 2016

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

scala.meta is officially endorsed (EPFL)


Martin Odersky
@odersky

Following

Slides of my Scala Days NYC talk:

Planned In Future Releases

scala.meta

Implicit
Function Types

Bet

n

ety

```
inline def m(inline x: Int, y: Float): Float =
  meta { ... }
```

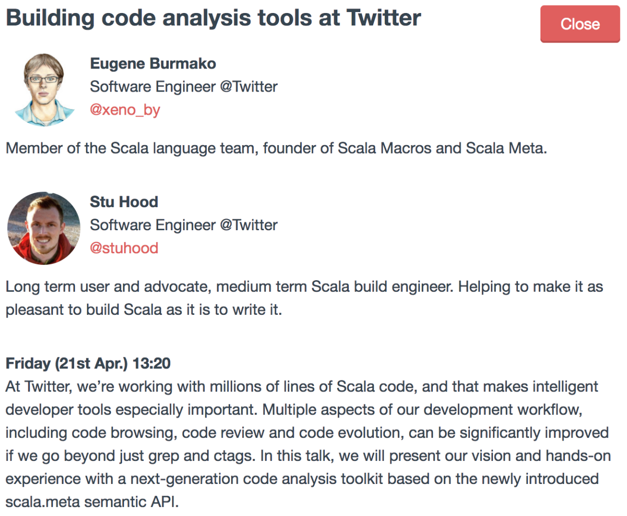
- **inline** for inlining, **meta** for meta-programming.
- run by an interpreter (no reflection)
- **meta** uses quasi quotes for matching and construction
- blackbox and annotation macros

Why the change?

- Simpler
- Fewer implementation dependencies
- Safer, since interpretation allows sandboxing
- Restrict syntactic freedom, since no whitebox macros.

50 of 63

scala.meta is officially endorsed (Twitter)



Building code analysis tools at Twitter

Eugene Burmako
Software Engineer @Twitter
@xeno_by

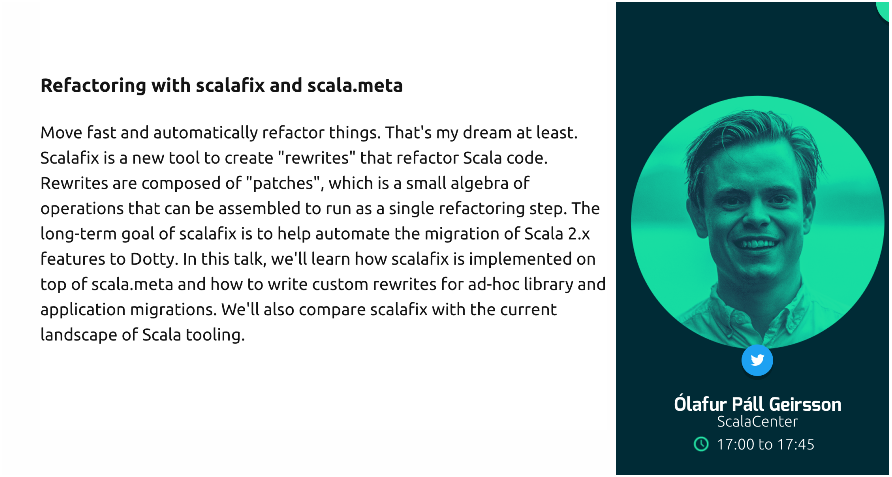
Member of the Scala language team, founder of Scala Macros and Scala Meta.

Stu Hood
Software Engineer @Twitter
@stuhood

Long term user and advocate, medium term Scala build engineer. Helping to make it as pleasant to build Scala as it is to write it.

Friday (21st Apr.) 13:20
At Twitter, we're working with millions of lines of Scala code, and that makes intelligent developer tools especially important. Multiple aspects of our development workflow, including code browsing, code review and code evolution, can be significantly improved if we go beyond just grep and ctags. In this talk, we will present our vision and hands-on experience with a next-generation code analysis toolkit based on the newly introduced scala.meta semantic API.

scala.meta is officially endorsed (Scala Center)

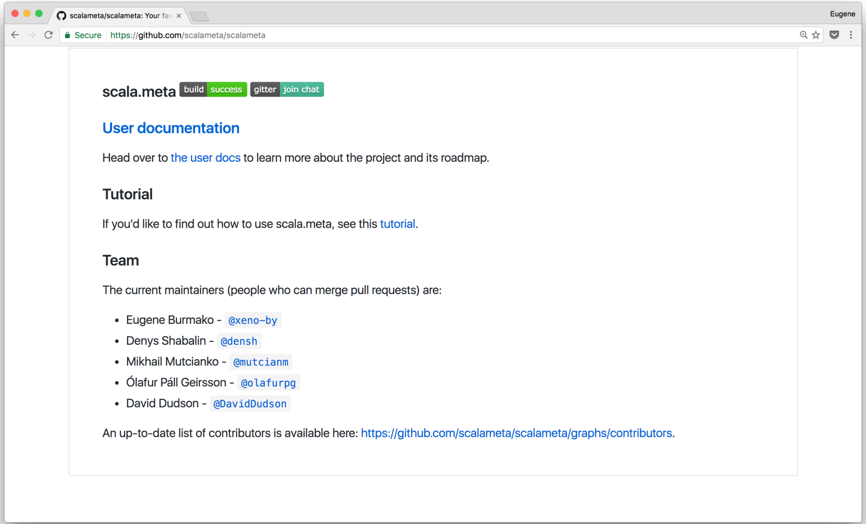


Refactoring with scalafix and scala.meta

Move fast and automatically refactor things. That's my dream at least. Scalafix is a new tool to create "rewrites" that refactor Scala code. Rewrites are composed of "patches", which is a small algebra of operations that can be assembled to run as a single refactoring step. The long-term goal of scalafix is to help automate the migration of Scala 2.x features to Dotty. In this talk, we'll learn how scalafix is implemented on top of scala.meta and how to write custom rewrites for ad-hoc library and application migrations. We'll also compare scalafix with the current landscape of Scala tooling.

Ólafur Páll Geirsson
ScalaCenter
17:00 to 17:45

scala.meta is an active project


scalameta/scalameta: Your fav
Eugene
Secure
https://github.com/scalameta/scalameta

scala.meta build success gitter join chat

User documentation

Head over to the user docs to learn more about the project and its roadmap.

Tutorial

If you'd like to find out how to use scala.meta, see this tutorial.

Team

The current maintainers (people who can merge pull requests) are:

- Eugene Burmako - `@xeno-by`
- Denys Shabalin - `@densh`
- Mikhail Mutcianko - `@mutcianm`
- Ólafur Páll Geirsson - `@olafurpg`
- David Dudson - `@DavidDudson`

An up-to-date list of contributors is available here: https://github.com/scalameta/scalameta/graphs/contributors.

Scala.meta API

Syntactic API

```
scala> import scala.meta._
import scala.meta._

scala> "println(List(1, 2, 3))".parse[Term].get
res0: scala.meta.Term = println(List(1, 2, 3))

scala> res0.structure
res1: String = Term.Apply(
  Term.Name("println"),
  Seq(Term.Apply(
    Term.Name("List"),
    Seq(Lit(1), Lit(2), Lit(3)))))

scala> res0.tokens
res2: scala.meta.tokens.Tokens =
Tokens(, println, (, List, (, 1, ,,  , 2, ,,  , 3, ), ), )
```

Semantic API

- What does a name resolve to?
- What type does an expression have?
- What does an expression desugar to?

Semantic API

- What does a name resolve to?
- What type does an expression have?
- What does an expression desugar to?

```
println(List(1, 2, 3))
```

- `println` resolves to `scala.Predef.println`
- `List` resolves to `scala.List`

Semantic API

- What does a name resolve to?
- What type does an expression have?
- What does an expression desugar to?

`println(List(1, 2, 3))`

- `println` has type `(Any)Unit`
- `List` has type `List.type`
- 1, 2 and 3 have type `Int`
- `List(1, 2, 3)` has type `List`
- `println(List(1, 2, 3))` has type `Unit`

Semantic API

- What does a name resolve to?
- What type does an expression have?
- What does an expression desugar to?

```
println(List(1, 2, 3))
```

- `List(...)` desugars to `List.apply(...)`
- `List.apply(...)` desugars to `List.apply[Int](...)`

Our research shows that...

- These three questions can be easily answered using compiler internals
- Answering them robustly and portably is very hard
- Comprehensive solutions most likely require person-years to implement
- ...and maintain

Rethinking our strategy

- Incrementally ship semantic APIs in bite-sized portions
- Implementation simplicity trumps comprehensiveness
- Portability is still a fundamental goal of the project

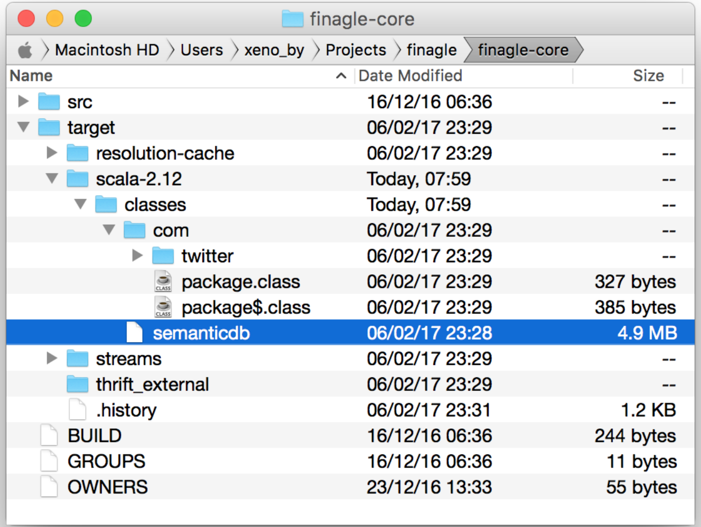
In this talk

```
scala> q"println(List(1, 2, 3))"
res0: scala.meta.Term = println(List(1, 2, 3))

scala> res0.structure
res1: String = Term.Apply(
  Term.Name("println"),
  Seq(Term.Apply(
    Term.Name("List"),
    Seq(Lit(1), Lit(2), Lit(3)))))
```

Semantic databases

Semantic databases


finagle-core
Macintosh HD › Users › xeno_by › Projects › finagle › finagle-core

Name	Date Modified	Size
src	16/12/16 06:36	--
target	06/02/17 23:29	--
resolution-cache	06/02/17 23:29	--
scala-2.12	Today, 07:59	--
classes	Today, 07:59	--
com	06/02/17 23:29	--
twitter	06/02/17 23:29	--
package.class	06/02/17 23:29	327 bytes
package$.class	06/02/17 23:29	385 bytes
semanticdb	06/02/17 23:28	4.9 MB
streams	06/02/17 23:29	--
thrift_external	06/02/17 23:29	--
.history	06/02/17 23:31	1.2 KB
BUILD	16/12/16 06:36	244 bytes
GROUPS	16/12/16 06:36	11 bytes
OWNERS	23/12/16 13:33	55 bytes

Semantic databases

```
$ ls
Library.scala

$ cat Library.scala
object Library {
  def main(args: Array[String]): Unit = {
    println(List(1, 2, 3))
  }
}
```

Semantic databases

```
$ scalac -Xplugin:.../scalahost.jar -Yrangepos Library.scala

$ ls
Library$.class
Library.class
Library.scala
semanticdb
```

Semantic databases

```
$ cat semanticdb
file:/Users/xeno_by/Projects/Meta1x/sandbox/Library.scala
[7..14): Library => _empty_.Library.
[23..27): main => _empty_.Library.main([Ljava/lang/String;)V.
[28..32): args => _empty_.Library.main([Ljava/lang/String;)V.(args)
[34..39): Array => _root_.scala.Array#
[40..46): String => _root_.scala.Predef.String#
[50..54): Unit => _root_.scala.Unit#
[63..70): println => _root_.scala.Predef.println(Ljava/lang/Object;)V.
[71..75): List => _root_.scala.collection.immutable.List.apply(...
```

Discussion

Semantic databases are...

- Portable
- Persistent
- Distributable

Mirrors

Mirrors

```
package scala.meta
package semantic
package v1

trait Mirror {
  def dialect: Dialect
  def sources: Seq[Source]
  def database: Database
  def symbol(ref: Ref): Completed[Symbol]
}
```

Mirrors

```
import scala.meta._

object Test extends App {
  val classpath = "..."
  val sourcepath = "..."
  implicit val mirror = Mirror(classpath, sourcepath)

  println(mirror.database)

  mirror.sources.foreach { source =>
    source.collect {
      case name @ Term.Name("println") =>
        println(name.symbol)
    }
  }
}
```

Discussion

You can create a mirror from:

- An instance of `scala.tools.nsc.Global`
- A classpath and a sourcepath
- An SBT build

Check out https://github.com/scalameta/sbt-semantic-example for a complete example and an accompanying guide.

Implementation details

Challenges

- Attributed trees have platform-dependent representation
- Including undocumented type inference
- Including undocumented desugarings

Strategy #1 (2015)

- Take compiler ASTs
- Try to revert platform-dependent desugarings
- Convert compiler ASTs to platform-independent ASTs

Half a year of work by compiler experts, several thousand lines of code, heavy modifications of the typechecker, almost works.

Strategy #2 (2016)

- Take compiler attributed ASTs
- Take platform-independent unattributed ASTs
- Traverse them together
- Produce platform-independent attributed ASTs

Several months of work by compiler experts, several thousand lines of code, no modifications to the typechecker, barely works.

Strategy #3 (2017)

- Take compiler ASTs
- For every name, locate a corresponding AST
- For every located AST, obtain and persist its symbol
- No platform-independent ASTs involved!

A month of work by compiler experts, several hundred lines of code, moderate modifications to the typechecker, almost works.

Future work

Next-generation tools

- Def macros
- Automatic refactorings (scalafix)
- Intelligent code browsers
- Better code review tools
- ...

More semantic APIs

- `Symbol.tpe`, `Symbol.members` and friends
- `Type.=:=`, `Type.<:<` and friends
- Additional functionality, strictly on per-usecase basis

Check out https://github.com/scalameta/scalameta/issues/604 for the current roadmap and links to individual work items.

Richer semantic databases

- Support for types
- Maybe even desugarings
- Tool-specific information (e.g. unused imports for scalafix)

Summary

Summary

- Scala.meta 1.6.0 ships with v1 of the semantic API
- Currently, the semantic API only includes `Ref.symbol`
- But we plan to iteratively ship more and more functionality in 2017
- This work is based on semantic databases - a major innovation in itself

Call for contributions

We need help with testing our semantic database technology. See https://github.com/scalameta/sbt-semantic-example for details.

To learn more:

- Ping us on Gitter: https://gitter.im/scalameta/scalameta
- Find me at a discussion table tomorrow at 12:00
- Join the scala.meta hackathon tomorrow at 17:00
- Attend our Scala Days talks in Chicago and Copenhagen