Introduction to scalameta-based macro annotations

Oleksandr Olgashko

What is that and what can be done

- @foo object bar { ... }
- Works on every expression, transforms it somehow.

What is that and what can be done

- Manipulations with AST, maps definitions to definitions.

```
Term.Apply(Term.Name("println"),
Seq(Lit("hello!”)))
=>
Term.Apply(Term.Name("println"),
Seq(Lit("hello, world!")))
```

What is that and what can be done

- How AST looks like?

```
def foo = 42
=>
Defn.Def(Nil, Term.Name("foo"), Nil, Nil, None, Lit(42))
```

- See scala.meta.Trees

What is that and what can be done

- How annotation looks like?

```
class main extends scala.annotation.StaticAnnotation {
  inline def apply(defn: Any): Any = meta {
    val q"object $name { ..$stats }" = defn
    val main = q"def main(args: Array[String]): Unit = { ..
$stats }
    q"object $name { $main }"
  }
}
```

What is that and what can be done

- Typechecking is delayed until expansion.

```
q"some really meaningless text"
=>
Term.Select = (some really meaningless).text
```

- No invalid code can be produced after expansion.

How it works

- AST on input, AST on output.
- See “inline/meta” SIP, Eugene Burmako’s dissertation for implementation details.

What are restrictions?

- By design:
 * No access to caller’s AST.
 * No access to parent’s AST.
 * No semantic API (left for “def macros”).

What are restrictions?

- Is not implemented yet:
  * Semantic API
    (typechecking, name resolution, implicits, ...)
  * Separate compilation
  * Runtime execution
  * Several top-level definitions
    (@ann class C => class D)

What are restrictions?

- Bugs:
  * https://github.com/scalameta/paradise/issues
  * https://github.com/scalameta/scalameta/issues

How to write

- import scala.meta._
  * scalac -cp <path_to_scalameta.jar>

- include paradise plugin
  * -Xplugin:<path_to_paradise.jar>

- https://github.com/scalameta/scalameta/blob/master/notes/quasiquotes.md

- scala.meta.Trees

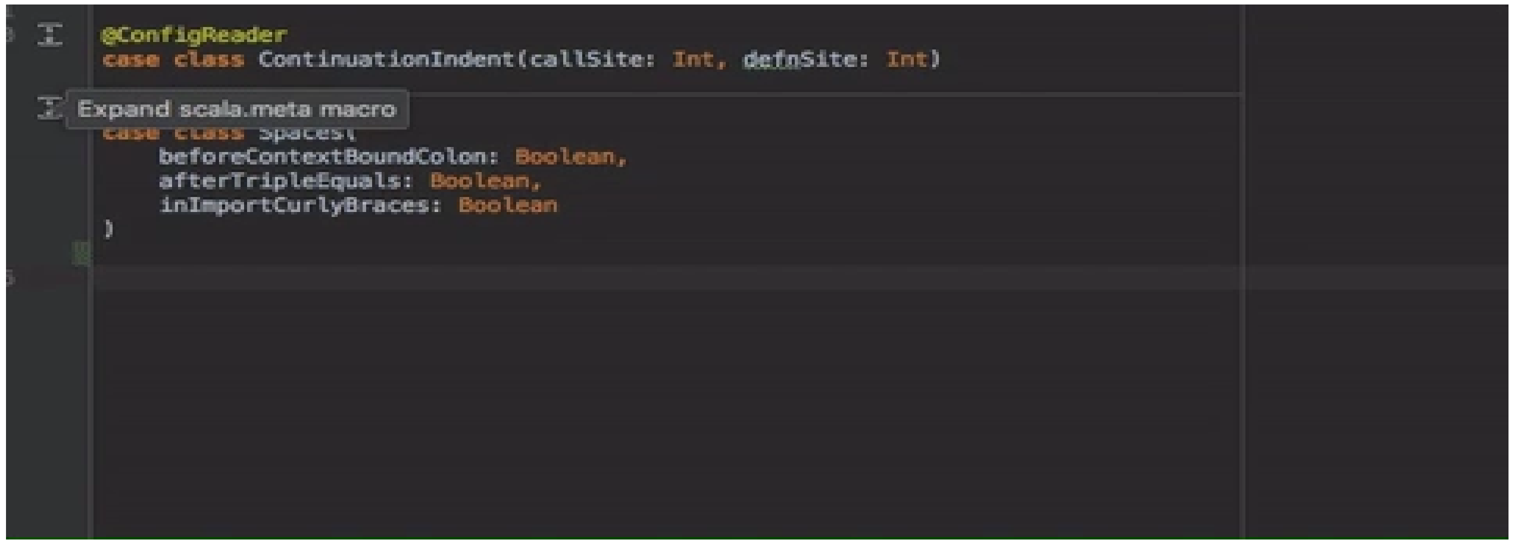
How to debug

- -Dquasiquote.debug
- scalac -print <>
- println
  * tree.show[Structure]
  * tree.show[Syntax]
- Don't forget to clean *.class files.

Analogs

- scalareflect-based macro annotations

- won’t be supported in Dotty
- wont change APIs if compiler internals changed
- less friendlier from metaprogrammer's point of view
- no IDE support



```
@ConfigReader
case class ContinuationIndent(callSite: Int, defnSite: Int)

case class Spaces(
    beforeContextBoundColon: Boolean,
    afterTripleEquals: Boolean,
    inImportCurlyBraces: Boolean
)
```

+ less API (see “restrictions”)

Analogs

- Lombok

- significantly less friendlier from metaprogrammer's point of view
- impossible to read annotation’s source code for programmer-user
- conflicts with other Java Annotation Processor -based libraries
- tons of hacks to get into compiler internals

+ better IDE support (for now…)
+ less API (see “restrictions”)

Analogs

- Python’s decorators

  hard to compare, different implementations, but same ideas.

DEMO

(see https://github.com/dveim/scala_meta_example_paradise for completed example)