WHAT WILL YOU FIND OUT?

- What is a macro?
- How to write a macro?
- So … anybody uses macros?
- What’s “macro paradise”? 
WHAT IS A MACRO?

**Macro (large):** expands into something larger

**Function:** code => code

**Invoked at build/compile-time**
SCALA MACROS

• Written in Scala
• Have access to and can manipulate the AST
• Use compiler/reflection APIs
• Type-safe
MACROS IN OTHER LANGUAGES

C/C++ – preprocessor

- `#define BUFFER_SIZE 1024`
- `#define min(X, Y) ((X) < (Y) ? (X) : (Y))`

Lisp/Clojure, Racket (Scheme)

- code is data (list)
- quoting
- "Since macros are so much harder to use than functions, a good rule of thumb is: don't use defmacro if defun will work fine"

from http://www.apl.jhu.edu/~hall/Lisp-Notes/Macros.html
MOTIVATION TO ADD MACROS TO SCALA

(it’s not a lean language already!)

✓ Remove boilerplate
✓ Replace run-time reflection
✓ Generate type-checked code
✓ Deep embedding of DSLs
✓ Type-check external DSLs
✓ Simplify compiler in the long run
REATIONS TO MACROS

Mixed ;)

Scala Macros: "Oh God Why?" - Jay Kreps
blog.empathybox.com/post/19126121307/
Mar 11, 2012 - This was my reaction to the Scala macros proposal too. Not because there is anything necessarily bad about macros or the proposal, but just ...
ABOUT ME

During the day: coding @ SoftwareMill
SoftwareMill: a great software house!
Afternoon: playgrounds, Duplo, etc.
Evening: blogging, open-source

• Original author of Hibernate Envers
• ElasticMQ, Veripacks, MacWire

http://www.warski.org
“DEF” MACROS

- Available since Scala 2.10 (Jan 2013)
- Only one type of many possible macro types
- Experimental status
WRITING A MACRO STEP-BY-STEP

Goal – transform this:

```plaintext
debug(x*amount)
```

To:

```plaintext
println("x*amount = " + (x*amount))
```

So that it outputs:

```plaintext
x*amount = 10.23
```
DEMO

WRITING A SIMPLE MACRO
WHERE ARE MACROS USED?

- Slick
- Akka
- Async
- MacWire

Examples are mostly from the projects’ websites
SLICK

@table(name="COFFEES") case class Coffee(
    @column(name="NAME") name: String,
    @column(name="PRICE") price: Double
)

val q = Queryable[Coffee]
val r = q.filter(_.price > 3.0).map(_.name)
ASYNC

```scala
val f1 = Future { beSlow(); 19 }
val f2 = Future { beSlow(); 23 }
val futureResult = for {
  v1 <- f1
  v2 <- f2
} yield v1 + v2
```
ASYNC

```scala
val futureResult = async {
    val f1 = async { beSlow(); 19 }
    val f2 = async { beSlow(); 23 }
    await(f1) + await(f2)
}
```
val future = async {
    val f1 = async { beSlow(); true }
    val f2 = async { beSlow(); 42 }
    if (await(f1)) await(f2) else 0
}

• Also possible with Akka Dataflow & CPS (but that’s so ’09)
• Can also use for-comprehensions; but quickly gets tricky
ERRORS

- Cryptic errors?
- Can be, if generated code doesn’t compile
- But we can provide user-friendly errors

```javascript
context.error(
    c.enclosingPosition,
    "You can’t do that")
```
DEMO

MACWIRE
AND OTHERS!

- Scala Pickling
- Akka 2.2 typed channels
- ScalaMock
- Typesafe Logging
- Scala Blitz
- Expecty
- ...

@adamwarski
OTHER TYPES OF MACROS

• Coming in Scala 2.12+

• Also available as a compiler plugin in 2.10/2.11
  • Macro Paradise

based on the examples from http://scalamacros.org/
DEF MACROS

• What we’ve seen so far
• Look like a method invocation
• Generate code basing on:
  • The parameters
  • Enclosing method/class
  • Implicit lookups
IMPLICIT MACROS

• Useful for Type Classes

```scala
trait Showable[T] { def show(x: T): String }

def useShow[T](x: T)(implicit s: Showable[T]) = s.show(x)

implicit object IntShowable {
    def show(x: Int) = x.toString
}
```
IMPLICIT MACROS

We want to provide a “default implementation” for a type

```scala
trait Showable[T] { def show(x: T): String }
object Showable {
  implicit def materialize [T]: Showable[T] =
    macro ...
}
```

We can get access to T at compile-time and generate what’s needed
trait Foo {
    def m1(p: Int): Long
    def m2(p1: String, p2: Date): Double
}

class FooWrapper(@delegate wrapped: Foo) extends Foo {
    def m1(p: Int) = wrapped.m1(p)+1L
}
MACRO ANNOTATIONS

Annotation-drive macros

- Any definitions can be annotated

```java
class delegate extends StaticAnnotation {
    def macroTransform(annottees: Any*) = macro ???
}
```
MACRO ANNOTATIONS

• Annottees is:
  • Annotated class + companion object
  • Method parameter, owner, companion

• Can expand classes
• Can create companion objects
QUASISOURCES

- Similar to string interpolators
- Extract from trees:

```scala
val q"def $name[..$tparams]($vparamss): $tpt
    = $body" = methodTree
```

- Pattern match

```scala
tree match {
  case q"def $name[..$tps]($vps): $tpt
    = $body" =>
}
```
QUASIIQUOTES

Construct

- Terms: \( \texttt{q"future\{ \$body \}} \)
- Types: \( \texttt{tq"Future\[$t]\}} \)
- Cases: \( \texttt{cq"x => x"} \)
- Patterns: \( \texttt{pq"xs @ (hd :: tl)"} \)
POTENTIAL PROBLEMS

- Hard to write
- Code may be harder to understand
- And to debug
WHEN TO WRITE A MACRO?

- Always think it through
- Lots of repeated, boilerplate code
- Unavoidable copy-paste (patterns)
- Library code

macro: power => responsibility
LINKS

- http://scalamacros.org/
- http://slick.typesafe.com/
- https://github.com/scala/async
- https://github.com/adamw/macwire
- https://github.com/adamw/scala-macro-tutorial
I AM PROUD OF MY CODE

#ProudOfMyCode

http://codebrag.com
COME & GET A STICKER

Adam Warski