CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Scala 2.12 and Beyond

Adriaan @adriaanm Moors

Scala Team Lead

adriaanm.github.io/reveal.js/scala-next.html


Typesafe

Community!

- Key to Scala's success
- Investments:
  - Simplify process
  - Infrastructure (CI on EC2, standardize build,...)
    - contribute from browser!
  - Modularization (more to come in 2.13)
    - community owned (merge PRs, push tag to release)

Growth!

- Contributors
  - Community: 1/4th of 2.11 commits (**× 3 since 2.10!**)
  - EPFL: 1/4th as well
  - Typesafe: half (team of 4)
- PRs merged
  - 2.11: 51 per month (from 43 in 2.10)

`g sl -sn --no-merges 2.11.x --not 2.10.x`:

- 188 A. P. Marki
- 125 Simon Ochsenreither
- 53 Antoine Gourlay
- 40 Iain McGinniss
- 22 Vladimir Nikolaev
- 19 Michał Pociecha
- 10 Rüdiger Klaehn
- 10 Paolo Giarrusso
- 10 Kenji Yoshida
- 10 Gérard Basler

Thank you!

Have fun!

Let me know how I can help

Scala Roadmap

Overview

2.12

Compiler release

- Java 8 (λ, SAM, `default`)
- New Optimizer (based on Miguel Garcia's work)
- Abide (time permitting)
- Set scene for library cleanups
- Q1 2016

Aida

Library release

- Cleanup, simplify & shrink stdlib
- Invite community modules
- `scala.meta` *// TODO: rewrite all the macros!*
- (±12 months after 2.12.3; 18 month cycle)

Don Giovanni, act 1

Rework back-end (see dotty)

- (keep front-end)
- faster (phase fusion)
- improve binary compat

Don Giovanni, act 2

Rework front-end (see dotty)

- remove deprecated syntax
- simpler foundations
- fewer puzzlers (sorry, Andrew & Nermin)

2.12 <3 Java 8

scala-lang.org/news/2.12-roadmap

Timeline

- Q1 2016: 2.12.0 & last *public* 2.11.x
  - ... after Java 7 public EOL
- 2.11 & 2.12 closely aligned
  - modulo Java 6

INDY-λ

2.11.7, 2.12.0-M2

1. scalac λ bytecode mimics javac's
2. ride JIT perf tuning wave
3. profit

`default` methods

2.12.0-M3

- Java 8 λ assignable-to Scala `FunctionN` (a SAM)
- Define Java 8 API in Scala
- Compile `trait` to `interface`
  - Binary compat ftw
  - Probably opt-in

Single Abstract Method

2.11.5 (`-Xexperimental`)

- Scala λ assignable-to Java 8 λ
- Call Java 8 API from Scala
- TODO
  - Better type inference with Java wildcards...
  - Spec
  - Also use INDY?

java.util.stream.Stream

```
/* https://gist.github.com/adriaanm/892d6063dd485d7dd221
Original Java 8 version:
http://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples

$ scala -Xexperimental                                              */
scala> import java.util.stream.{Stream, IntStream}
scala> import java.util.{Arrays, List}

// List<String> myList= Arrays.asList("a1", "a2", "b1", "c2", "c1");
scala> val myList = Arrays.asList("a1", "a2", "b1", "c2", "c1")
myList: java.util.List[String] = [a1, a2, b1, c2, c1]

/*
myList
    .stream()
    .filter(s -> s.startsWith("c"))
```

SAM Synthesis

```
scala> Stream of ("d2", "a2", "b1", "b3", "c") filter { s =>
         println(s"filter: $s"); true
       }

// Behind the scenes, scalac generated:
scala> Stream of ("d2", "a2", "b1", "b3", "c") filter {
    def test$body(s: String): Boolean = println(s"filter: $s"); true

    new Predicate { def test(s: String): Boolean = test$body(s) }
  }
```

FP >> λs

(A brief rant)

Define, compose and understand small units of abstraction over known set of types

(OO's complement)

Express, Understand & Refactor

- immutable first (val <> var, expr <> stmt, recursion)
- pattern matching (analysis of input determines output)
- local methods (divide & conquer)
- (de)compose safely, guided by types
- type inference (best types are unseen -- e.g., definition-site variance)

Function subtyping

Java: `Function`'s users must encode expected subtyping

```
<R> Stream<R> map(Function<? super T, ? extends R> mapper)
```

Scala: => said how type arguments determine subtyping

```
def map[R](mapper: T => R): Stream[R]
```

Variance

definition	ensuring fun subtyping
`trait =>[-T, +R]`	`=>[Arg, Res]`
`trait F[T, R]`	`F[_ >: Arg, _ <: Res]`

Natural subtyping for functions

```
BetterArg => BetterRes <: Arg => Res
```

<=>

```
BetterArg >: Arg /\ BetterRes <: Res
```

Wildcard subtyping for functions

```
F[_ >: BetterArg, _ <: BetterRes]
<: F[_ >: Arg, _ <: Res]
```

<=>

```
BetterArg >: Arg /\ BetterRes <: Res
```

Aida

scala-lang.org/news/roadmap-next

Simpler Core Library

- Composition over inheritance
- Move out parallel collections & views
- Caveat Inheritor Collectionis
  - 99% of users need not worry, though

Modules

- Lazy collections (interop with Java 8 Streams)
- Parallel collections (multiple impls: scala-blitz & Java 8)
- Validation

Macros: scala.meta

- Prototype, default in Don Giovanni
- Complete rewrite!
- Only use macros when all else fails...

Don Giovanni

scala-lang.org/news/roadmap-next

Refactor compiler

- Inspired by dotty incubator
- Focus on ease of migration, bang for buck

Simpler syntax

- Drop procedure syntax
  - all definitions: `kw name sig [= body]`
- So long, XML literals!
  - `xml""<hello/>"""`
- Anyone ever use early definitions?
  - introducing: trait parameters

Simpler semantics

- Type members model type constructors, existentials,...
- No more lub blow-ups (union types as lazy lubs)
- Built-in support for some of shapeless (`HList`/`HMap`)

Binary Compatibility

- Serialize Typed ASTs
- Recompile dependencies
- Source & binary compatibility coincide
- See Martin's talk

CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Thanks!

Questions!

Rate my talk on the GOTO Guide app?

@adriaanm