Type-Level Programming in Scala 3

張瑋修 Walter Chang

@weihsiu / weihsiu@gmail.com

Scala Taiwan

Scala Taiwan gitter channel

Scala Taiwan FB group

Scala Taiwan meetup

Agenda 1/2

- What is Scala 3?
- What is type-level programming?
- Quick words on Scala types
- Match types
- Firsts.scala
- SKICalculus.scala
- Nats.scala
- Vects.scala

Agenda 2/2

- scala.Tuple
- Tuples.scala
- scala.compiletime
- Math.scala
- Sorts.scala
- HMaps.scala
- TicTacToe.scala
- Q&A

What is Scala 3?

- A upcomping version of Scala
- Easier to learn than Scala 2
- More intuitive
- More powerful

What is type-level programming?

- Programming with types, not values
- Runs during compilation, not runtime
- Utilizes Scala's powerful type system
- Catch coding error before execution
- Put singleton types to good use

Quick words on Scala types

```
val one: 1 /* type */ = 1 // value
val trueV: true /* type */ = true // value
val hello: "hello" /* type */ = "hello" //v value

trait CelestialBody

case object Sun extends CelestialBody
val sun: Sun.type /* type */ = Sun // value
val body1: CelestialBody /* type */ = sun // value

case class Planet() extends CelestialBody
val mars: Planet /* type */ = Planet() // value
val body2: CelestialBody /* type */ = mars // value
```

Match types

- The foundation type-level programming construct
- Acts like "function" with types as "arguments" and returning different types as result

```
// from dotty doc
type Elem[X] = X match {
  case String => Char
  case Array[t] => t
  case Iterable[t] => t
}

summon[Elem[String] =:= Char]
summon[Elem[Array[Int]] =:= Int]
summon[Elem[List[Float]] =:= Float]
summon[Elem[Nil.type] =:= Nothing]
```

Firsts.scala

- A function that returns first of everything
- Integrates type-level and term-level definitions
- A dependent-type function with a succinct implementation

SKICalculus.scala

- A proof that Scala 3 type system is turing-complete
  - (I, x) = x
  - ((K, x), y) = x
  - (((S, x), y), z) = ((x, z), (y, z))

Nats.scala

- Canonical type-level programming example
  - Plus operation
  - Multiply operation
  - Many more are possible

Vects.scala

- A list whose length is encoded in its type
  - length()
  - map()
  - zip()
  - concat()

```
trait Vect[N <: Nat, +A]
```

scala.Tuple

- Scala 3 standard package
- https://github.com/lampepfl/dotty/blob/master/library/src/scala/Tuple.scala
- A tuple implemenation that gets rid of the 22-arity limit of Scala 2
- Extensive use of type-level programming
- A lightweight Shapeless

```
val tuple1: (Int, Char, Boolean) = (1, 'a', true)
val tuple2: Int *: Char *: Boolean *: EmptyTuple = (1, 'a', true)
assert(tuple1 == tuple2)
```

Tuples.scala

```
case class User(name: String, age: Int)
val user1 = User("walter", 18)
val userT = Tuple.fromProductTyped(user1)
val user2 = User.tupled(userT)
assert(user1 == user2)
```

scala.compiletime

- Scala 3 standard package
- A new package that has lots of type-level goodies
- scala.compiletime.ops._
  - Int, Boolean and String singleton type operations in types

```
val i: 1 + 2 = 3
val b: 1 > 2 = false
val s: "a" + "b" = "ab"
```

Math.scala

- Some cool math examples in types
  - Greatest common divisor
  - Fibonacci number
  - Odd/Even

Sorts.scala

- Insertion sort of tuple element types
- Sorting order can be altered by using different type predicates

HMaps.scala

- A hashmap implementation with heterogeneous value types

```
val hm = HMap[String]
  .put[String]("name", "walter")
  .put[Int]("age", 18)
  .put[Int]("age", 20)
  .put[String]("age", "twenty")
hm[String]("name")
hm[Int]("age")
val hm1 = hm.remove("age")
hm1[String]("age")
```

TicTacToe.scala

- Tic Tac Toe game at type-level
- Order of play is enforced by the compiler
- Invalid moves are detected by the compiler
- Winning condition is detected by the compiler

Q&A

That's all and thank you for your attention