Implicits

Pepe García

2020-04-20

Implicits

Implicits provide us the ability to pass arguments to function in an *implicit* way.

The simplest way for understanding them is to see a simple example:

```scala
implicit val name: String = "Pepe"
// name: String = "Pepe"


def printImplicitName(implicit name: String): Unit =
  println("Hello " + name)


printImplicitName
// Hello Pepe
```

Implicits

But, how does it work?

When finding an implicit parameter, Scala will look for candidates for the given type in:

- The lexical scope, as any other variable
- The implicit scope
- The companion object of the datatype
- The companion object of the typeclass
- In the imports

reference in the Scala docs.

Applications

Implicit conversions

One of the most used features of implicits are implicit conversions. Using implicit conversions, we'll be able to use values of a type as values of other type **if there's an implicit conversion between them**.

We'll declare implicit conversions as `implicit def`.

Implicit conversions

```scala
import java.util.UUID

implicit def uuidAsString(uuid: UUID): String =
  uuid.toString

val id: UUID = UUID.randomUUID
// id: UUID = 6815ac78-4bef-4464-bace-def89968db8e

def printString(str: String): Unit =
  println(str)

printString(id)
// 6815ac78-4bef-4464-bace-def89968db8e
```

Datatype expansion

AKA: Pimp My Library

Datatype expansion

We'll use datatype expansion when we want to add new methods to datatypes we don't control.

As an example... let's copy Ruby's `n.times do...` pattern.

```
5.times {
  println("it worked!")
}
// error: value times is not a member of Int
// 5.times {
// ^^^^^^^
```

Datatype expansion

We can add new methods to a datatype we don't control using
implicit classes:

```scala
implicit class IntTimes(val x: Int) {
  def times(action: => Unit): Unit = {
    (1 to x).foreach(_ => action)
  }
}
```

Datatype expansion

```
5.times {
  println("it worked!")
}
// it worked!
// it worked!
// it worked!
// it worked!
// it worked!
```

Typeclasses

We've already mention typeclasses previously, but let's refresh it.

Other languages such as Haskell, have typeclasses built into the language, but we don't have that in Scala, we need to emulate them. The most common way of using typeclasses in Scala is by passing their instances as implicit parameters.

Typeclasses

Typeclass declaration

```scala
trait ToString[A] {
  def toString(a: A): String
}

object ToString {
  def apply[A](
    implicit TS: ToString[A]
  ): ToString[A] = TS
}
```

Typeclasses

Instance declaration

```scala
implicit val toStringInt: ToString[Int] =
  new ToString[Int] {
    def toString(a: Int): String =
      a.toString
  }
// toStringInt: ToString[Int] = repl.Session$App$$anon$1@54f7acc1

// Scala can also use Single Abstract Method syntax,
// as Java
implicit val toStringFloat: ToString[Float] =
  _.toString
// toStringFloat: ToString[Float] = repl.Session$App$$anonfun$9@2beaabde
```

Typeclasses

Usage

```scala
def print[A: ToString](a: A): Unit =
  println(ToString[A].toString(a))

print(1)
// 1
print(2f)
// 2.0

print(true)
// error: could not find implicit value for evidence parameter of type repl.
// print(true)
// ^^^^^^^^^^^
```