Scala Course

Implicits

47 Deg

2021-02-15

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:

```
implicit val name: String = "John"
// name: String = "John"

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

printImplicitName
// Hello John
```

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

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

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

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

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

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

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.

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 = 9d2eed17-0548-4587-b626-eb5ba4107a54

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

printString(id)
// 9d2eed17-0548-4587-b626-eb5ba4107a54
```

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`:

```
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

Typeclass declaration

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

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

Typeclasses

Instance declaration

```
implicit val toStringInt: ToString[Int] =
  new ToString[Int] {
    def toString(a: Int): String =
      a.toString
  }
// toStringInt: ToString[Int] = repl.MdocSession$App$$an

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

Typeclasses

Usage

```
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 par
// print(true)
// ^^^^^^^^^^
```