Under the Hood of Scala Implicits

by Alexander Podkhalyuzin

ScalaDays San Francisco 2015
Agenda

- How implicits work
- How IDE can help with them
- Possibilities to improve performance of compilation and IDE responsiveness
Implicits: The Beginning

- Implicits
- Conversions
- Parameters
Implicits: Conversions

- Implicits
- Parameters
- Conversions
  - Type conversions
  - Extension methods
Type conversions

```scala
implicit def b2a(a: B): A = new A
class A; class B
val a: A = new B
```

Compiler will search implicit conversions of type $B \Rightarrow A$. 
Search scopes

1. Resolve scope
   a. Declared implicits
   b. Imported implicits
   c. Implicitly imported implicits

2. Extended scope
   a. Companion objects of parts? of $B \Rightarrow A$ type.
If few implicitss were found...

class Base; object Base {
    implicit def clazz2str(c: Clazz): String = ""
}
class Clazz extends Base; object Clazz {
    implicit def base2str(b: Base): String = ""
}
val c: String = new Clazz
Performance

If all implicits are in the resolve scope

1. Harder to debug problems, when wrong implicit is chosen
2. Complexity to find right one
Performance

Solution:
1. Use “Extended scope” to encapsulate implicits as much as possible
2. Try to split implicits by priority
class A; class B

def toInt(x: B): Int = 123

def goo(x: A) = toInt(new A)

implicit def a2b(x: A) = new B
Extension methods

We can add new methods to existing types

class Type

class TypeExt {
  def foo(x: Int) = 123
}

implicit def extType(a: Type): TypeExt =
  new TypeExt
  (new Type).foo(1)
Algorithm...

1. Find method without implicit conversion.
2. All are not applicable
   a. Find such conversions, which add applicable method
   b. Choose most specific conversion of them
3. If in (1) we found something not applicable, then we can’t use implicit for args
4. We can’t choose most specific method among methods coming from different implicit conversions
Implicit classes

For most cases we want implicit classes:

```
implicit class Foo(s: String) {
  def intValue: Int = s.toInt
}
```
Magnet pattern

- Widely used in Spray routes
- Originally was described in spray.io blog
Parameter Info problem

With magnet pattern “Parameter Info” stops working in IDE, isn’t it?..
Implicit parameters

Compiler is able to fill missing implicit parameter list on the call site:

```scala
implicit val x: Int = 1
def foo(implicit x: Int) = x + 1
foo
```
Recursiveness

The main feature, which gives us a step towards type-level programming:

```scala
class Ordering[T]
implicit val intOrdering: Ordering[Int] = null
implicit def listOrdering[T](
    implicit t: Ordering[T]
): Ordering[List[T]] = null
```
How it works?

Implicit parameter search is completely the same as Implicit conversion search, just we will search not only function types.
SOE...

```
implicit def a(implicit s: String): Int = 1
implicit def b(implicit i: Int): String = "1"
```

Compiler is able to stop recursion. How?
Another example

```scala
implicit def a[T](implicit t: T): List[T] = ???

implicit def b[T](implicit t: T): Option[T] = ???

implicit val s: String = ???

def foo(implicit l: List[Option[List[String]]]) {}

foo //foo(a(b(a(s)))))
```
Complexity of type

Number of different parts of type:

*List[Int]* complexity is 2

*Seq[Int, Option[String]]* complexity is 4

Top level classes are *List* and *Seq*.
Complexity of type

To avoid SOE:

● Have list of types to search
● Do not add new search for some type if
  ○ List contains equivalent type
  ○ List contains type with same top level class and complexity of new type is bigger
Local type inference

It’s not legal in Scala:

```scala
def foo[T](x: T, y: T => String) = y(x)
foo("text", s => s + s)
```
Local type inference

Scala compiler tries to solve type parameters for every single parameter list.

Solution:

```scala
def foo[T](x: T)(y: T => String) = y(x)
foo("text")(s => s + s)
```
Type inference for implicits

So is it the same for implicit parameters?
It's good when everything is ok.

-Xprint:typer can help you to see implicits.
Implicit conversions

In IntelliJ IDEA you can use currently available tool to analyze available implicit conversions.
Implicit parameters

In IntelliJ IDEA

- Full analysis even in bad cases
- It’s not including implicit conversions
What’s next

● Use implicits as it’s most important feature of Scala language
● Now you know almost everything about implicits, so use it safely
● Choose right tools...
Thank you!

twitter: Safela
email: Alexander.Podkhalyuzin@jetbrains.com