Implicit Conversions

An *implicit conversion*, also called *view*, is a conversion that is applied by the compiler in several situations:

1. When an expression `e` of type `T` is encountered, but the compiler needs an expression of type `S`.
2. When an expression `e.m` where `e` has type `T` but `T` defines no member `m` is encountered.

In those cases, the compiler looks in the implicit scope for a conversion that can convert an expression of type `T` to an expression of type `S` (or to a type that defines a member `m` in the second case).

This conversion can be either:

1. An `implicit def` of type `T ⇒ S` or `(⇒ T) ⇒ S`
2. An implicit value of type `scala.Conversion[T, S]`

Defining an implicit conversion will emit a warning unless the import `scala.language.implicitConversions` is in scope, or the flag `-language:implicitConversions` is given to the compiler.

Examples

The first example is taken from `scala.Predef`. Thanks to this implicit conversion, it is possible to pass a `scala.Int` to a Java method that expects a `java.lang.Integer`

```
import scala.language.implicitConversions
implicit def int2Integer(x: Int): java.lang.Integer =
  x.asInstanceOf[java.lang.Integer]
```

The second example shows how to use `Conversion` to define an `Ordering` for an

arbitrary type, given existing `Ordering`s for other types:

```
import scala.language.implicitConversions
implicit def ordT[T, S](
    implicit conv: Conversion[T, S],
             ordS: Ordering[S]
   ): Ordering[T] =
   // `ordS` compares values of type `S`, but we can convert from `T` to `S`
  (x: T, y: T) => ordS.compare(x, y)

class A(val x: Int) // The type for which we want an `Ordering`

// Convert `A` to a type for which an `Ordering` is available:
implicit val AToInt: Conversion[A, Int] = _.x

implicitly[Ordering[Int]] // Ok, exists in the standard library
implicitly[Ordering[A]] // Ok, will use the implicit conversion from
                        // `A` to `Int` and the `Ordering` for `Int`.
```

More details

‹ Chang... Implici... ›