Scala 3 Reference / Dropped Features / Dropped: Weak Conformance

LEARN INSTALL PLAYGROUND FIND A LIBRARY COMMUNITY BLOG

Dropped: Weak Conformance

Edit this page on GitHub

In some situations, Scala used a *weak conformance* relation when testing type compatibility or computing the least upper bound of a set of types. The principal motivation behind weak conformance was to make an expression like this have type `List[Double]`:

```scala
List(1.0, math.sqrt(3.0), 0, -3.3) // : List[Double]
```

It's "obvious" that this should be a `List[Double]`. However, without some special provision, the least upper bound of the lists's element types `(Double, Double, Int, Double)` would be `AnyVal`, hence the list expression would be given type `List[AnyVal]`.

A less obvious example is the following one, which was also typed as a `List[Double]`, using the weak conformance relation.

```scala
val n: Int = 3
val c: Char = 'X'
val d: Double = math.sqrt(3.0)
List(n, c, d) // used to be: List[Double], now: List[AnyVal]
```

Here, it is less clear why the type should be widened to `List[Double]`, a `List[AnyVal]` seems to be an equally valid -- and more principled -- choice.

Weak conformance applies to all "numeric" types (including `Char`), and independently of whether the expressions are literals or not. However, in hindsight, the only intended use case is for *integer literals* to be adapted to the type of the other expressions. Other types of numerics have an explicit type annotation embedded in their syntax (`f`, `d`, `.`, `L` or `'` for `Char`s) which ensures that their author really meant them to have that specific type).

Therefore, Scala 3 drops the general notion of weak conformance, and instead keeps one rule: `Int` literals are adapted to other numeric types if necessary.

More details

‹ Dropp... Dropp... ›