Pattern Bindings

Edit this page on GitHub

In Scala 2, pattern bindings in `val` definitions and `for` expressions are loosely typed. Potentially failing matches are still accepted at compile-time, but may influence the program's runtime behavior. From Scala 3.1 on, type checking rules will be tightened so that warnings are reported at compile-time instead.

Bindings in Pattern Definitions

```
val xs: List[Any] = List(1, 2, 3)
val (x: String) :: _ = xs   // error: pattern's type String is more specialized
                            // than the right-hand side expression's type Any
```

This code gives a compile-time warning in Scala 3.1 (and also in Scala 3.0 under the `-source future` setting) whereas it will fail at runtime with a `ClassCastException` in Scala 2. In Scala 3.1, a pattern binding is only allowed if the pattern is *irrefutable*, that is, if the right-hand side's type conforms to the pattern's type. For instance, the following is OK:

```
val pair = (1, true)
val (x, y) = pair
```

Sometimes one wants to decompose data anyway, even though the pattern is refutable. For instance, if at some point one knows that a list `elems` is non-empty one might want to decompose it like this:

```
val first :: rest = elems   // error
```

This works in Scala 2. In fact it is a typical use case for Scala 2's rules. But in Scala 3.1 it will give a warning. One can avoid the warning by marking the right-hand side with an `@unchecked` annotation:

```
val first :: rest = elems: @unchecked   // OK
```

This will make the compiler accept the pattern binding. It might give an error at runtime instead, if the underlying assumption that `elems` can never be empty is wrong.

Pattern Bindings in `for` Expressions

Analogous changes apply to patterns in `for` expressions. For instance:

```
val elems: List[Any] = List((1, 2), "hello", (3, 4))
for (x, y) <- elems yield (y, x) // error: pattern's type (Any, Any) is more sp
                                 // than the right-hand side expression's
type Any
```

This code gives a compile-time warning in Scala 3.1 whereas in Scala 2 the list `elems` is filtered to retain only the elements of tuple type that match the pattern `(x, y)`. The filtering functionality can be obtained in Scala 3 by prefixing the pattern with `case`:

```
for case (x, y) <- elems yield (y, x)  // returns List((2, 1), (4, 3))
```

Syntax Changes

Generators in for expressions may be prefixed with `case`.

```
Generator      ::=  [‘case’] Pattern1 ‘<-’ Expr
```

Migration

The new syntax is supported in Scala 3.0. However, to enable smooth cross compilation between Scala 2 and Scala 3, the changed behavior and additional type checks are only enabled under the `-source future` setting. They will be enabled by default in version 3.1 of the language.

‹ Vararg ... Option... ›

Scaladoc