TOUR OF SCALA

FOR COMPREHENSIONS

Scala offers a lightweight notation for expressing *sequence comprehensions*. Comprehensions have the form `for (enumerators) yield e`, where `enumerators` refers to a semicolon-separated list of enumerators. An *enumerator* is either a generator which introduces new variables, or it is a filter. A comprehension evaluates the body `e` for each binding generated by the enumerators and returns a sequence of these values.

Here's an example:

```
case class User(name: String, age: Int)

val userBase = List(
  User("Travis", 28),
  User("Kelly", 33),
  User("Jennifer", 44),
  User("Dennis", 23))

val twentySomethings =
  for (user <- userBase if user.age >=20 && user.age < 30)
  yield user.name  // i.e. add this to a list

twentySomethings.foreach(println)  // prints Travis Dennis
```

A `for` loop with a `yield` statement returns a result, the container type of which is determined by the first generator. `user <- userBase` is a `List`, and because we said `yield user.name` where `user.name` is a `String`, the overall result is a `List[String]`. And `if user.age >=20 && user.age < 30` is a guard that filters out users who are not in their twenties.

Here is a more complicated example using two generators. It computes all pairs of numbers between `0` and `n-1` whose sum is equal to a given value `v`:

```
def foo(n: Int, v: Int) =
   for (i <- 0 until n;
        j <- 0 until n if i + j == v)
    yield (i, j)

foo(10, 10) foreach {
  case (i, j) =>
    println(s"($i, $j) ")  // prints (1, 9) (2, 8) (3, 7) (4, 6) (5, 5) (6, 4) (7, 3) (8, 2) (9, 1)
}
```

Here `n == 10` and `v == 10`. On the first iteration, `i == 0` and `j == 0` so `i + j != v` and therefore nothing is yielded. `j` gets incremented 9 more times before `i` gets incremented to `1`. Without the `if` guard, this would simply print the following:

```
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4) (0, 5) (0, 6) (0, 7) (0, 8) (0, 9) (1, 0) ...
```

Note that comprehensions are not restricted to lists. Every datatype that supports the operations `withFilter`, `map`, and `flatMap` (with the proper types) can be used in sequence comprehensions.

You can omit `yield` in a comprehension. In that case, comprehension will return `Unit`. This can be useful in case you need to perform side-effects. Here's a program equivalent to the previous one, but without using `yield`:

```
def foo(n: Int, v: Int) =
   for (i <- 0 until n;
        j <- 0 until n if i + j == v)
   println(s"($i, $j)")

foo(10, 10)
```

More resources

- Other examples of “For comprehension” in the Scala Book

← previous next →

Contributors to this page: