Theory: Tuples

13 minutes 0 / 5 problems solved

You know some basic Scala collections, including Maps. As you remember, a map is a collection of key-value pairs. A pair, in its turn, is a special data structure called a tuple. In this topic, we will learn more about this type of language expression and how it can help us define complex data types.

§1. Pairs

A pair is a basic example of a tuple. In Scala, a **tuple** is a value that contains a fixed number of elements, where each may have a unique type. The ability to have a variety of types is the main difference between tuples and collections, where elements have to be similar. For example, a pair can be two elements with the key `Char` and the value `Int`:

```
scala> 'a' -> 1
res0: (Char, Int) = (a,1)
```

Note, that for pairs we use special syntax with an arrow. This is just syntactic sugar that makes it easier to distinguish programming tuples from tuples in mathematics:

```
scala> val tuple = ('a', 1)
tuple: (Char, Int) = (a,1)
```

Obviously, you can define any combination of types:

```
scala> (1.0, "string")
res1: (Double, String) = (1.0,string)
scala> (true, 4f)
res2: (Boolean, Float) = (true,4.0)
scala> (8L, ())
res3: (Long, Unit) = (8,())
```

After the definition of pair, you can access the first element as `_1` and second as `_2`:

```
scala> tuple._1
res4: Char = a
scala> tuple._2
res5: Int = 1
```

§2. Triples

In some cases, combining two elements (key and value) is not enough, so we need three. This is not a problem, we have triples! Imagine that for some app we need the name, the year of birth and the rating of a person. We can define it like this:

```
scala> val triple = ("Alex", 1982, 3)
triple: (String, Int, Int) = (Alex,1982,3)
```

You can access the elements by `_1`, `_2` and `_3` respectively:

```
scala> triple._3
res6: Int = 3
```

Note that we don't use syntax with an arrow for triples: if we do use it, we define a pair of a pair and element:

```
scala> val nonTriple = "Alex" -> 1982 -> 3
nonTriple: ((String, Int), Int) = ((Alex,1982),3)
```

Current topic:

Tuples

Topic depends on:

- ✕ Values and variables
- ✕ Introduction to collections

Table of contents:

We can keep going and define tuples with more elements: 4, 5, 6... These types don't have special names in Scala like pairs or triples do.

§3. Tuples

Large tuples are similar to database records: we have some data with a type in a place (column). We can represent user data with additional fields and define a collection of tuples:

```
scala> val userList = List(("Alice", 1985, 9, 4, 'a'), ("Scally", 1988, 5, 6, 's')
, ("Bob", 1990, 12, 7, 'b'))
userList: List[(String, Int, Int, Int, Char)] = List((Alice,1985,9,4,a), (Scally,1
988,5,6,s), (Bob,1990,12,7,b))
scala> userList(0)._1
res6: String = Alice
scala> userList(2)._5
res7: Char = b
```

In Scala, tuples have a limitation: it is impossible to define more than 22 elements.

```
scala> (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23)
<console>:1: error: too many elements for tuple: 23, allowed: 22
       (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23)
       ^
```

If you for some reason need to use tuples with more than 22 elements, you can do it with nested tuples or collections:

```
scala> (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,(22,23))

res8: (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int,
Int, Int, Int, Int, Int, (Int, Int)) =
  (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,(22,23))
scala> (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,List(22,23))

res9: (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int, Int,
Int, Int, Int, Int, Int, List[Int]) =
  (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,List(22, 23))
```

Here we substituted two last numbers for a pair and a list respectively. Note that since we have a limitation of 22 elements, the last element you could get from this tuple is `_22`.

§4. Conclusion

For tuples, we start with two elements (a pair), because a tuple with one element is the element itself. Three elements form a triple, and tuples from 4 to 22 exist without special names. With the help of tuples, we can define structured data of various types: database records, registries, maps and many others.

Report a typo

7 users liked this theory. **0** didn't like it. **What about you?**

😍 🙂 😐 ☹️ 😡

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion