Theory: Introduction to collections

13 minutes 0 / 3 problems solved

Most objects or concepts we face in life don't come single: you buy five apples, there are three unique answers to the question "Do you know Scala?" (yes, no, and what), we see thirty slots in a calendar for each day – you get the idea. It makes sense that in our programs we also have to describe such structures known in programming as **collections**. Scala has a nice selection of types that allow to work with practically any kind of elements.

§1. Lists

A simple example of a Scala collection is a **list**: an ordered collection with a defined length.

```
scala> val list = List(1, 3, 8)
list: List[Int] = List(1, 3, 8)
scala> list.length
res1: Int = 3
scala> list(0)
res2: Int = 1
scala> list(2)
res3: Int = 8
scala> list(3)

java.lang.IndexOutOfBoundsException: 3
```

We use `length` to get a count of elements and `(index)` to get an element with the specified index starting from 0 and ending at length minus 1. If we try to get an element with an index greater than length – 1 or negative, we'll get an error.

There is a special symbol to define an empty list in Scala – `Nil`. It is the same as `List()` or `List.empty`. We can construct new lists with the help of `Nil` and concatenation `::`:

```
scala> val characters = 'c' :: 'h' :: 'a' :: 'r' :: Nil
characters: List[Char] = List(c, h, a, r)
scala> characters.isEmpty
res4: Boolean = false
scala> characters.head
res5: Char = c
scala> characters.tail
res6: List[Char] = List(h, a, r)
```

Above, we've used some convenient methods for lists: `isEmpty` checks for emptiness, `head` gets the first element and `tail` gets the list without the first element.

§2. Arrays

An **array** is a collection of elements each identified by index. You may ask: then what is the difference between arrays and lists? In general, the difference is not huge: arrays are just more affiliated with Java code and have low-level representation. For example, a union of arguments for the main function is an array:

```
object Main {
  def main(args: Array[String]): Unit = {
    println(args(0))
  }
}
```

We're speaking of arrays mostly to build a bridge between Java and Scala; you may encounter this type of collection somewhere in your programming exercises. Practically, lists are more convenient as they have more methods.

Current topic:

Introduction to collections

Topic depends on:

× Values and variables

Topic is required for:

Tuples

Table of contents:

The basic operations with arrays are mostly the same as described above for the lists:

```
scala> val array = Array(0.0, 0.5, 3.5, 4)
array: Array[Double] = Array(0.0, 0.5, 3.5, 4.0)
scala> array.length
res7: Int = 4
scala> array(1)
res8: Double = 0.5
```

§3. Sets

A **set** is a collection with no duplicate elements. We can check if an element is part of a set or not:

```
scala> val set = Set(1, 2, 3)
set: scala.collection.immutable.Set[Int] = Set(1, 2, 3)
scala> set(2)
res9: Boolean = true
scala> set.contains(4)
res10: Boolean = false
```

Note that we can do the check just by the `(element)` or with the help of `contains`.

A reasonable question is why do we need sets if we have lists, especially since lists have the method `contains`, too. The answer is simple: not everything allows repetition. If we had a set of students in a group, it would be strange if we had some person twice there, or if we had two Aprils in a year. So if you want to store unique elements in a collection, `Set` is your choice.

§4. Maps

Maps are collections of key and value pairs. With the help of a key, we can get a value:

```
scala> val map = Map(1 -> 'a', 2 -> 'b', 5 -> 'a')
map: scala.collection.immutable.Map[Int,Char] = Map(1 -> a, 2 -> b, 5 -
> a)
scala> map(2)
res11: Char = b
scala> map(3)
java.util.NoSuchElementException: key not found: 3
```

Above we defined `Map` with `Int` keys and `Character` values. We get the value with `(key)`; if we don't have any, we get an error.

Maps are structures that help us match one piece of data with another, for example a name of a person with their age, or a calendar year with a list of holidays in it. Note that keys for the map form a set, so we don't have duplicates.

```
scala> map.contains(2)
res12: Boolean = true
scala> map.keys
res13: Iterable[Int] = Set(1, 2, 5)
scala> map.values
res14: Iterable[Char] = MapLike.DefaultValuesIterable(a, b, a)
```

Above we used `contains` to check for a key existing in `Map`, keys to get the key-`Set`, and values to get a list of all values (it is possible to have duplicates in this list).

§5. Conclusion

We've covered the main collection types in Scala. With the help of a `List`, we can define ordered elements, each of them with an index. With a `Set` we can do the same, but without duplicates. With a `Map`, we can define matches between sets of keys and a list of values.

We mentioned only the basic methods for collection types above, but there are others. Don't hesitate to play with your code and explore it by yourself. This is an introduction topic, so later we'll return to collections to learn more.

Report a typo

12 users liked this theory. 1 didn't like it. What about you?

Start practicing

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