Scala Collections How many ways are there to say "Multiple Things"

Julian Bieber

December 13, 2018

Collections Trivia Type Hierarchy Views Benchmar

Collections Trivia



Sequence

val sequence = Seq(1, 2, 3, 4, 5)
println(sequence)

val sequence = Seq(1, 2, 3, 4, 5)println(sequence)

List(1, 2, 3, 4, 5)

Stream

```
val stream = Stream(1, 2, 3, 4, 5)
println(stream)
val seq: Seq[Int] = stream.toSeq
seq.map(println)
```

```
val stream = Stream(1, 2, 3, 4, 5)
println(stream)
val seq: Seq[Int] = stream.toSeq
seq.map(println)
```

Stream(1, ?)

```
val stream = Stream(1, 2, 3)
stream.foreach(println)
val streamPlusOne = stream.map( + 1)
println(stream.size)
streamPlusOne.foreach(println)
```

```
val stream = Stream(1, 2, 3)
stream.foreach(println)
val streamPlusOne = stream.map( + 1)
println(stream.size)
streamPlusOne.foreach(println)
```

```
Seq(1, 2, 3).map\{ i = > 
   println("method1", i)
map{i =>
   println("method2", i)
```

Sequence Order of Execution

```
Seq(1, 2, 3).map{ i =>
    println("method1", i)
    i
}.map{ i =>
    println("method2", i)
    i
}
```

```
(method1,1)
(method1,2)
(method1,3)
(method2,1)
(method2,2)
```

(method2,3)

Stream Order of Execution

```
Stream(1, 2, 3).map\{i = >
                                        println("method1", i)
\mbox{} \mbo
                                        println("method2", i)
```

(method 1, 1) (method 2, 1)

Stream Order of Execution

```
Stream(1, 2, 3).map\{i = >
                                        println("method1", i)
\mbox{} \mbo
                                        println("method2", i)
```

(method 1, 1) (method 2, 1)

Stream.force?

```
Stream(1, 2, 3).map{ i =>
  println("method1", i)
  i
}.map{ i =>
  println("method2", i)
  i
}.force
```

Stream.force?

```
Stream(1, 2, 3).map\{ i = > 
  println("method1", i)
\mbox{\normalfont } .map{ i => }
  println("method2", i)
}.force
```

```
(method 1, 1)
(method2,1)
(method1,2)
(method2,2)
```

(method1,3)

(method2,3)

Infinite Streams

```
val stream = Stream.from(1).map{ i =>
  println("method1", i)
  i
}.map{ i =>
  println("method2", i)
  i
}
println(stream.sum)
```

Infinite Streams

```
val stream = Stream.from(1).map{ i =>
  println("method1", i)
  i
}.map{ i =>
  println("method2", i)
  i
}
println(stream.sum)
```

```
...
```

(method1,2517132) (method2,2517132)

 ${\sf Exception: java.lang.OutOfMemoryError}$

Infinite Iterators

```
val iterator = Stream.from(1).map{ i =>
  println("method1", i)
  i
}.map{ i =>
  println("method2", i)
  i
}.tolterator

println(iterator.sum)
```

Infinite Iterators

```
val iterator = Stream.from(1).map{i =>}
                           println("method1", i)
\mbox{} \mbo
                           println("method2", i)
  }.tolterator
     println(iterator.sum)
```

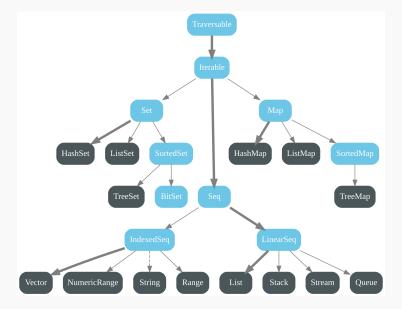
(method1,2517132) (method2,2517132) ections Trivia Type Hierarchy Views Benchmarks

Type Hierarchy



ections Trivia Type Hierarchy Views Benchmark

Immutabe



Array

- ► slow prepend/append
- ightharpoonup very fast random access (O(1))
- contiguous locality
- strict

Vector

- ightharpoonup fast random access (O(c))
- ightharpoonup fast append/prepend (O(c))
- good but not contiguous locality
- ► strict

- \triangleright very fast prepend (O(1))
- ightharpoonup very fast head access (O(1))
- ightharpoonup single linked \Rightarrow slow random access (O(n))
- bad locality
- strict

Stream

- ► List with lazy tail
- while something holds the head it can not be GC'ed
- will not be consumed by iterating (while something hold the head)
- not strict

Concept of a View

- calling view on a collection makes it non strict
- modifications can be applied as usual (map, filter, ...)
- however they are not evaluated yet

Concept of a View

- calling view on a collection makes it non strict
- modifications can be applied as usual (map, filter, ...)
- however they are not evaluated yet
- force applies the changes
- better memory foodprint

Scala SeqView

- ➤ Signature: SeqView[+A, +Coll]
- ► Represents a view to Coll[+A]
- chaining maps without creating new collections
- ▶ ⇒ less memory consumption

Scala SeqView

- ➤ Signature: SeqView[+A, +Coll]
- ► Represents a view to Coll[+A]
- chaining maps without creating new collections
- ▶ ⇒ less memory consumption
- ▶ ⇒ less GC activity

Scala SeqView

- ► Signature: SeqView[+A, +Coll]
- Represents a view to Coll[+A]
- chaining maps without creating new collections
- ▶ ⇒ less memory consumption
- ightharpoonup \Rightarrow less GC activity
- ▶ ⇒ faster

lections Trivia Type Hierarchy Views Benchmarks

Benchmarks

