[ARRAYDEQUE]

Scala 2.13 Collections

* https://english.stackexchange.com/questions/226954/how-is-deque-commonly-pronounced

ABOUT ME

- Pathikrit Bhowmick
- github.com/pathikrit
- Coatue Management
- Scala for 5 years
- Data structures ❤ FP

SCALA 2.13 COLLECTIONS

https://www.scala-lang.org/blog/2018/06/13/scala-213-collections.html

17:30-18:15 - Wednesday

Metropolitan Ballroom - Foundations

Stefan Zeiger
@StefanZeiger

Julien Richard-Foy
@julienrf

Migrating to Scala 2.13

~~CANBUILDFROM~~

```
def map[B, That](f: A => B)(implicit bf: CanBuildFrom[Repr, B, That]): That
```

https://www.scala-lang.org/blog/2017/05/30/tribulations-canbuildfrom.html

~~CANBUILDFROM~~

```scala
def map[B, That](f: A => B)(implicit bf: CanBuildFrom[Repr, B, That]): That
```

```scala
trait IterableOps[A, CC[_]] {
  def map[B](f: A => B): CC[B]
}
```

https://www.scala-lang.org/blog/2017/05/30/tribulations-canbuildfrom.html

NEW APIS

NEW APIS

```
def namesByAge(users: Seq[User]): Map[Int, Seq[String]] =
  users.groupBy(_.age).mapValues(users => users.map(_.name))
```

NEW APIS

```
def namesByAge(users: Seq[User]): Map[Int, Seq[String]] =
  users.groupBy(_.age).mapValues(users => users.map(_.name))
```

```
def namesByAge(users: Seq[User]): Map[Int, Seq[String]] =
  users.groupMap(_.age)(_.name)
```

IN PLACE MUTABLE API

IN PLACE MUTABLE API

```
val users: mutable.ArrayBuffer[User] = ???

users
  .filterInPlace(user => !user.name.startsWith("J"))
  .mapInPlace(user => user.copy(age = user.age + 1))
```

MORE

- ~~Stream~~ vs LazyList
- Better Views
- Iterable vs. ~~Traversable/Iterator~~
- New collections

ARRAYDEQUE

ARRAYDEQUE

- New collection in Scala 2.13
- Also known as CircularBuffer

ARRAYDEQUE

- New collection in Scala 2.13
- Also known as CircularBuffer
- Replacement for most mutable collections
- Faster than ArrayBuffer when used as an array
- Faster than LinkedList when used as a linked list

ARRAYDEQUE

- New collection in Scala 2.13
- Also known as CircularBuffer
- Replacement for most mutable collections
- Faster than ArrayBuffer when used as an array
- Faster than LinkedList when used as a linked list
- *My first contribution to Scala!*

DATA STRUCTURES 101

Data Structures	get(idx)	update(idx, e)	append(e)	prepend(e)	deleteFirst()	deleteLast()	insertAt(idx, e)	deleteAt(idx)
mutable.ArrayDeque	**O(1)**	**O(1)**	**O(1)**	**O(1)**	**O(1)**	**O(1)**	**O(min(i, n-i))**	**O(min(i, n-i))**
mutable.ArrayBuffer	O(1)	O(1)	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)
mutable.Stack	O(n)	O(n)	O(1) (push)	O(n)	O(n)	O(1) (pop)	O(n)	O(n)
mutable.Queue	O(n)	O(n)	O(1) (enque)	O(n)	O(1) (deque)	O(n)	O(n)	O(n)
mutable.LinkedList	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)	O(n)	O(n)
java.util.ArrayList	O(1)	O(1)	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)
java.util.ArrayDeque	N/A	O(1)	O(1)	O(1)	O(1)	O(1)	O(n)	O(n)

ARRAY

DOUBLY

ENDED

QUEUE

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)
}
```

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)

  def append(a: A): Unit = {
    array(end) = a
    end = (end + 1)%n
  }
}
```

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)

  def append(a: A): Unit = {
    array(end) = a
    end = (end + 1)%n
  }

  def prepend(a: A): Unit = {
    start = (start - 1)%n
    array(start) = a
  }
}
```

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)

  def append(a: A): Unit = {
    array(end) = a
    end = (end + 1)%n
  }

  def prepend(a: A): Unit = {
    start = (start - 1)%n
    array(start) = a
  }

  def deleteLast(): Unit =
    end = (end - 1)%n
}
```

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)

  def append(a: A): Unit = {
    array(end) = a
    end = (end + 1)%n
  }

  def prepend(a: A): Unit = {
    start = (start - 1)%n
    array(start) = a
  }

  def deleteLast(): Unit =
    end = (end - 1)%n

  def deleteFirst(): Unit =
    start = (start + 1)%n
}
```

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)

  def append(a: A): Unit = {
    array(end) = a
    end = (end + 1)%n
  }

  def prepend(a: A): Unit = {
    start = (start - 1)%n
    array(start) = a
  }

  def deleteLast(): Unit =
    end = (end - 1)%n

  def deleteFirst(): Unit =
    start = (start + 1)%n

  def clear(): Unit =
    start = end
}
```

ARRAY DOUBLY ENDED QUEUE

```
class ArrayDeque[A] {
  val n = 64
  val array = Array.ofDim[A](n)
  var start, end = 0

  def update(i: Int, a: A): Unit =
    array((start + i)%n) = a

  def apply(i: Int): A =
    array((start + i)%n)

  def append(a: A): Unit = {
    array(end) = a
    end = (end + 1)%n
  }

  def prepend(a: A): Unit = {
    start = (start - 1)%n
    array(start) = a
  }

  def deleteLast(): Unit =
    end = (end - 1)%n

  def deleteFirst(): Unit =
    start = (start + 1)%n

  def clear(): Unit =
    start = end

  def size: Int =
    (end - start)%n
}
```

DEMO

SCALA 2.13

`scala.collection.mutable.ArrayDeque`

PERFORMACE

PERFORMACE

- Array.copy (memcpy)
  - insertAt(idx), deleteAt(idx), remove(idx)
  - clone()
  - slice()
  - Pre-emptive allocations:
    - insertAll(), prependAll()

PERFORMACE

- Array.copy (memcpy)
  - insertAt(idx), deleteAt(idx), remove(idx)
  - clone()
  - slice()
  - Pre-emptive allocations:
    - insertAll(), prependAll()
- Bit hack if n (array.length) = 2^k
  - `i%n == i&(n - 1)`

BENCHMARKS

Operation	ArrayBuffer	ArrayDeque	Speedup
Insert lots of items	2473.36 ms	956.76 ms	2.5x
Drop some items from an head index	7.65 ms	1.25 ms	5x
Drop some items from a tail index	2.54 ms	0.28 ms	10x
Append lots of items one by one	3576.63 ms	2222.13 ms	1.5x
Prepend few items one by one	8699.13 ms	1.33 ms	O(n)
Prepend lots of items at once	2124.02 ms	462.76 ms	5x
Random indexing	81.62 ms	84.02 ms	-
Insert items near head	2980.46 ms	1429.52 ms	2x
Reversal	491.46 ms	378.69 ms	1.5x
Insert items near tail	8588.98 ms	2504.20 ms	3x
Sliding	1591.47 ms	157.25 ms	10x
toArray	194.55 ms	181.07 ms	-
Clear lots of items	48.34 ms	28.62 ms	2x

GITHUB.COM/STANCH/REFTREE

GITHUB.COM/STANCH/REFTREE

```
import reftree.core._
import reftree.render._
import reftree.diagram._
import reftree.util.Reflection._

implicit def renderArrayDeque: ToRefTree[ArrayDeque[Char]] = ToRefTree {ds =>
  val array = ds.privateField[Array[AnyRef]]("array")
  val start = ds.privateField[Int]("start")
  val end = ds.privateField[Int]("end")

  val arrayRef = {
    val arrayFields = array.zipWithIndex map { case (a, i) =>
      val fieldName = {
        var s = i.toString
        if (i == start) s = '↳' + s
        if (i == end) s = s + '↲'
        s
      }
      val refTree = Option(a) match {
        case Some(c) => RefTree.Val(c.asInstanceOf[Char]).withHighlight(true)
        case None => RefTree.Null().withHighlight(i == end)
      }
      refTree.toField.withName(fieldName)
    }
    RefTree.Ref(array, arrayFields).rename(s"char[${array.length}]")
  }

  RefTree.Ref(ds, Seq(
    start.refTree.withHighlight(true).toField.withName("start"),
    end.refTree.withHighlight(true).toField.withName("end"),
    arrayRef.toField.withName("array")
  ) ++ ds.toArray.zipWithIndex.map({case (a, i) => a.refTree.toField.withName(i.toString)}))
}
```

TAKE AWAYS

- Please contribute

TAKE AWAYS

- Please contribute
  - Contributing to Scala is not scary

TAKE AWAYS

- ArrayDeques are cool
- Please contribute
  - Contributing to Scala is not scary
  - Data structures > Algorithms
  - Rope, SkipList, Zipper, Heap

TAKE AWAYS

- ArrayDeques are cool
- Please contribute
  - Contributing to Scala is not scary
  - Data structures > Algorithms
  - Rope, SkipList, Zipper, Heap
- Visualize your data structures

TAKE AWAYS

- ArrayDeques are cool
- Please contribute
  - Contributing to Scala is not scary
  - Data structures > Algorithms
  - Rope, SkipList, Zipper, Heap
- Visualize your data structures
- Scala 2.13 awaits!

THANK YOU

github.com/pathikrit/arraydeque-talk

WE ARE HIRING

- **Who we are:**
  - Coatue Management
- **What we do:**
  - Quant Trading
  - Data Science
  - NLP
- **What we love:**
  - Functional Programming
  - Algorithms
  - Statistics
  - Data
- **Tech stack:**
  - Scala
  - Spark
  - PostgreSQL
  - Tableau
  - R
  - Python
  - Docker
  - AWS
- **Where are we:**
  - NYC
  - SF

pbhowmick@coatue.com