

Scala Collections



cerenode.io

Collections in Scala

- Collections are the container of things which contains random number of elements
- Nothing but Classes found in the package `scala.collection`
- There are mutable and immutable collections
- Mainly there are three types of collection classes
 - Sequence
 - Set
 - Map




Mutable Collection

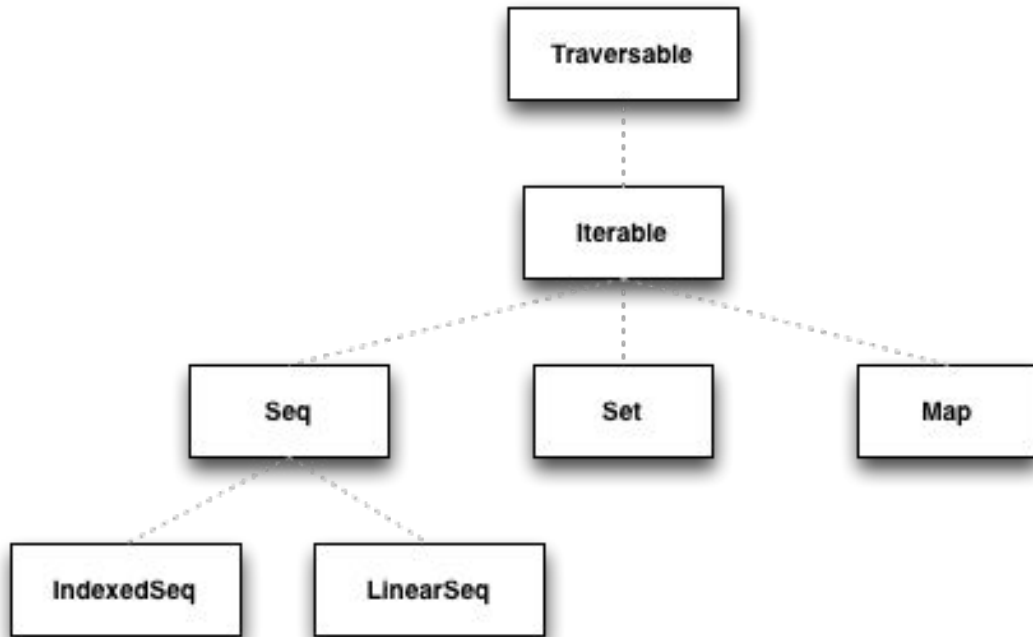
This type of collection is changed after it is created. All Mutable collection classes are found in the package `scala.collection.mutable`.

Immutable Collection

The elements in **most** immutable collections will not change after it is created. All immutable collection classes are found in the package `scala.collection.immutable`.

A decorative graphic in the bottom right corner consisting of several overlapping triangles in shades of pink and magenta.

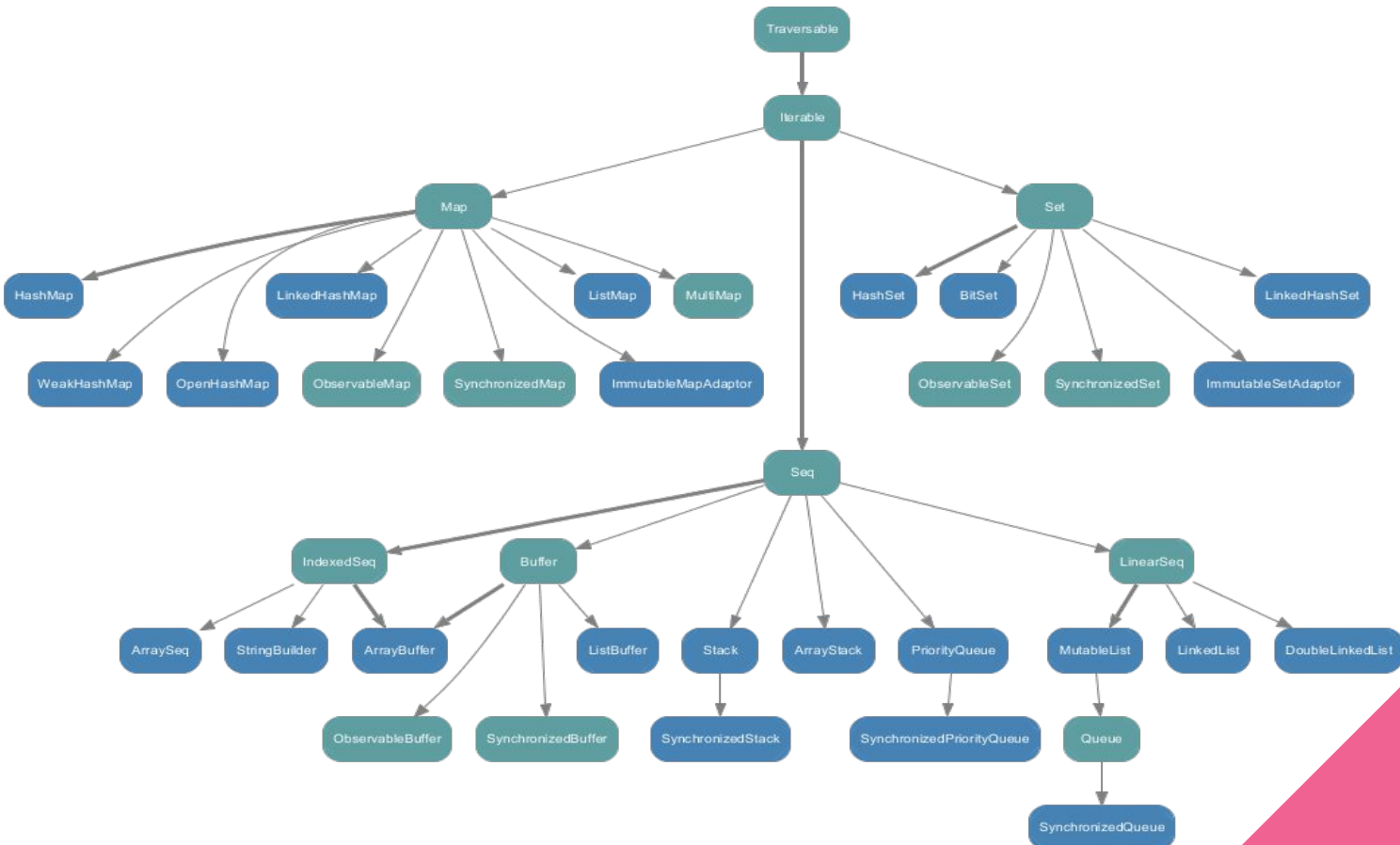
Scala Collections Hierarchy



All collections in package `scala.collection.immutable`



Collections in package `scala.collection.mutable`



Important Methods of Traversable Trait



```
def head: A
```

It returns the first element of collection.

```
def init: Traversable[A]
```

It returns all elements except last one.

```
def isEmpty: Boolean
```

It checks whether the collection is empty or not. It returns either true or false.

```
def last: A
```

It returns the last element of this collection.

```
def max: A
```

It returns the largest element of this collection.

```
def min: A
```

It returns smallest element of this collection



Important Methods of Traversable Trait (contd.)



```
def size: Int
```

It is used to get size of this traversable and returns a number of elements present in this traversable.

```
def sum: A
```

It returns sum of all elements of this collection.

```
def tail: Traversable[A]
```

It returns all elements except first.

```
def toArray: Array[A]
```

It converts this collection to an array.

```
def toList: List[A]
```

It converts this collection to a list.

```
def toSeq: Seq[A]
```

It converts this collection to a sequence.

```
def toSet[B >: A]: immutable.Set[B]
```

It converts this collection to a set.



Methods In Trait Iterable

`xs` **iterator**

An iterator that yields every element in `xs`, in the same order as `foreach` traverses elements.

`xs` **grouped** `size`

An iterator that yields fixed-sized “chunks” of this collection.

`xs` **sliding** `size`

An iterator that yields a sliding fixed-sized window of elements in this collection.

`xs` **takeRight** `n`

A collection consisting of the last `n` elements of `xs` (or, some arbitrary `n` elements, if no order is defined).



`xs dropRight n`

The rest of the collection except `xs takeRight n`.

`xs zip ys`

An iterable of pairs of corresponding elements from `xs` and `ys`.

`xs zipAll (ys, x, y)`

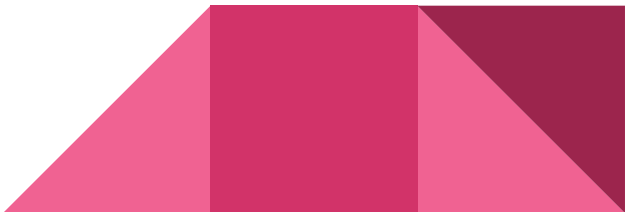
An iterable of pairs of corresponding elements from `xs` and `ys`, where the shorter sequence is extended to match the longer one by appending elements `x` or `y`.

`xs zipWithIndex`

An iterable of pairs of elements from `xs` with their indices.

`xs sameElements ys`

A test whether `xs` and `ys` contain the same elements in the same order



Sequence

- Collections of elements may be indexed or linear (as linked list)
- An IndexedSeq indicates that random access of elements is efficient

```
scala> val x = IndexedSeq(1, 2, 3)
x: IndexedSeq[Int] = Vector(1, 2, 3)
```

- A LinearSeq implies that the collection can be efficiently split into head and tail components, and it's common to work with them using the head, tail and isEmpty methods

```
scala> val seq = scala.collection.immutable.LinearSeq(1, 2, 3)
seq: scala.collection.immutable.LinearSeq[Int] = List(1, 2, 3)
```



Range

- Represent integer values in a range with non-zero step value
- Two ways to implement range
 - `val k: Range[Char] = 'a' to 'c' // elements a, b, c`
 - `val m: Range[Int] = 1 until 3 // elements 1, 2`
- Commonly used in for loops



List

- Data structure to represent a collection of values of the same type
- Immutable linked lists representing ordered collections of elements of type A
 - `val k: List[Int] = List(1, 2, 3, 4)`
- Optimal for stack-like access patterns
- Most commonly used data structure in Scala



Vector

- Provides random access and updates in constant time
- Fast append and prepend
 - `val k: Vector[Int] = Vector(1, 2, 3, 4)`
- Addresses the inefficiency for random access on Lists
- Implemented as 32-way trees



Array

- Array is a special kind of collection in Scala
- Even though it's an immutable collection, we can change the value of its element
- We can create up to five dimensional array in scala
- They store elements of a single type in a linear order
- “class” Array is Java’s Array, which is more of a memory storage method than a class



Methods In Trait Seq



Indexing and length apply, isDefinedAt, length, indices, and lengthCompare

Index search operations indexOf, lastIndexOf, indexOfSlice, lastIndexOfSlice, indexWhere, lastIndexWhere, segmentLength, prefixLength, which return the index of an element equal to a given value or matching some predicate.

Addition operations +:, :+, padTo, which return new sequences obtained by adding elements at the front or the end of a sequence.

Update operations updated, patch, which return a new sequence obtained by replacing some elements of the original sequence.

Sorting operations sorted, sortWith, sortBy, which sort sequence elements according to various criteria.

Reversal operations reverse, reverseliterator, reverseMap, which yield or process sequence elements in reverse order.

Comparisons startsWith, endsWith, contains, containsSlice, corresponds, which relate two sequences or search an element in a sequence.

Multiset operations intersect, diff, union, distinct, which perform set-like operations on the elements of two sequences or remove duplicates.



Sets

- Scala Set is a collection of unique elements

```
scala> val set = Set(1, 2, 3)  
set: scala.collection.immutable.Set[Int] = Set(1, 2, 3)
```



Methods In Sets

- **Lookup** operations contains, apply, subsetOf
- **Additions** + and ++, which add one or more elements to a set, yielding a new set.
- **Removals** -, --, which remove one or more elements from a set, yielding a new set.
- **Set operations** for union, intersection, and set difference. Each of these operations exists in two forms: alphabetic and symbolic. The alphabetic versions are intersect, union, and diff



Maps

- Scala Map is a collection of key/value pairs, where all the keys must be unique.

```
scala> val week = Map(1-> "Sun", 2 -> "Mon",3 ->"Tue" , 4 -> "Wen", 5 ->
"Thu" , 6 -> "Fri", 7-> "Sat")
week: scala.collection.immutable.Map[Int,String] = Map(5 -> Thu, 1 -> Sun,
6 -> Fri, 2 -> Mon, 7 -> Sat, 3 -> Tue, 4 -> Wen)
```

- Values can be retrieved using keys
 - `map(1) // gives the String "Sun"`



Methods In Maps

- **Lookup** operations `apply`, `get`, `getOrElse`, `contains`, and `isDefinedAt`. These turn maps into partial functions from keys to values. The fundamental lookup method for a map is: `def get(key): Option[Value]`. The operation “`m get key`” tests whether the map contains an association for the given key. If so, it returns the associated value in a `Some`. If no key is defined in the map, `get` returns `None`.
- Maps also define an `apply` method that returns the value associated with a given key directly, without wrapping it in an `Option`. If the key is not defined in the map, an exception is raised.
- **Additions and updates** `+`, `++`, `updated`, which let you add new bindings to a map or change existing bindings.
- **Removals** `-`, `--`, which remove bindings from a map.
- **Subcollection producers** `keys`, `keySet`, `keysIterator`, `values`, `valuesIterator`, which return a map's keys and values separately in various forms.
- **Transformations** `filterKeys` and `mapValues`, which produce a new map by filtering and transforming bindings of an existing map.