

4.14

Tour of the Scala API

Brief interlude on partial functions

- A partial function is a function which is defined for a subset of possible input values.

trait *PartialFunction* extends *Function1*:

```
trait PartialFunction[-A, +B] extends (A) => B
abstract def isDefinedAt(x: A): Boolean
```

- Some examples:

```
scala> val fraction = new Function[Int, Int] {
|   def apply(d: Int) = 1 / d
| }
fraction: Function[Int,Int] = <function1>
scala> List(1,0) map fraction
java.lang.ArithmeticException: / by zero
at $anon$1.apply$mcII$sp(<console>:11)
at $anon$1.apply(<console>:11)
at $anon$1.apply(<console>:10)
at scala.collection.immutable.List.map(List.scala:277)
... 33 elided

scala> val fraction = new PartialFunction[Int, Int] {
|   def apply(d: Int) = 1 / d
|   def isDefinedAt(d: Int) = d != 0
| }
fraction: PartialFunction[Int,Int] = <function1>
scala> List(1,0) collect fraction
res1: List[Int] = List(1)
scala> List(1, "1") collect { case i: Int => i + 1 }
res2: List[Int] = List(2)
```

Collections (1)





- *Iterable[A]*

- Methods defined (in *IterableLike*, *IterableOnce*, etc.):
 - (iteration) *foreach*, *iterator*, *grouped*, *slice*, *sliding*
 - (concatenation) *++* and *++:* append two iterables together.
 - (monad) *map*, *flatMap*, *filter/Not* and *collect*: *collect* takes a partial function
 - (conversions) *toArray*, *toList*, *toIterable*, *toSeq*, *toIndexedSeq*, *toLazyList*, *toSet*, *toMap*: all convert (but only if necessary) to the appropriate type
 - (copying) *copyToBuffer*, *copyToArray*
 - (size) *isEmpty*, *nonEmpty*, *size*, and *hasDefiniteSize*
 - (element retrieval) *head*, *last*, *headOption*, *lastOption*, and *find*

caters to infinite collections
like LazyList

may not be meaningful if
collection is not ordered.

Collections (2)

- Iterable continued...
- Continuing other methods defined:
 - (sub-collection retrieval) *tail, init, slice, take, drop, takeWhile, dropWhile, filter, filterNot, withFilter*  These are similar: see below
 - (subdivision) *splitAt, span, partition, groupBy*
 - element tests (by predicate) *exists, forall, count*
 - (accumulating) *foldLeft, foldRight, reduceLeft, reduceRight, scan, scanLeft, scanRight*  works if underlying type is *Numeric* or *Ordered*
 - (specific folds) *sum, product, min, max*  optional
 - (string operations) *mkString(start, sep, end, addString, stringPrefix)* 
 - (views) *view, view(from, to)*

```
partition(p: A => Boolean) => (T[A], T[A])
span(p: A => Boolean) => (T[A], T[A])
splitAt(n: Int) => (T[A], T[A])
groupBy(f: A => K) => Map[K, T[A]]
```

Collections (3)

- *Iterable* extends *Iterable*

- Provides an iterator:


```
def foreach(f: Elem => Unit): Unit = {  
  val it = iterator  
  while (it.hasNext) f(it.next())  
}
```

- (other iterators) *grouped*, *sliding*
- (*GenIterable*):
 - *zip*

may be overridden by subclasses;
an *Iterator* is not itself a collection
but can be a generator in for-comp



Collections (4)

- Sequence traits:
 - *Seq* (extends *Iterable*), *IndexedSeq*, *LinearSeq* (both extend *Seq*)
 - (indexing and length) *apply*, *isDefinedAt*, *length*, *indices*, and *lengthCompare*
 - (index searches) *indexOf*, *lastIndexOf*, *indexOfSlice*, *lastIndexOfSlice*, *indexWhere*, *lastIndexWhere*, *segmentLength*, *prefixLength*
 - (element addition) *+:*, *:+*, *padTo*
 - (updates) *updated*, *patch*
 - (sorting) *sorted*, *sortWith*, *sortBy*
 - (reversal) *reverse*, *reverseIterator*, *reverseMap*
 - (comparison) *startsWith*, *endsWith*, *contains*, *containsSlice*, *corresponds*
 - (multiset) *intersect*, *diff*, *union*, *distinct*
- extends *PartialFunction[Int]*. Methods all the same but efficiency of operations varies
- 

Collections (5)

- Other traits/types:
 - *List, Map, Set, Array**
 - Immutable collections
 - Mutable collections
 - See it all here: <http://docs.scala-lang.org/overviews/collections/introduction.html>
 - And look up individual types/methods here: <http://www.scala-lang.org/api/2.12.5/#package>

* *Array[T]* has two implicit conversions: to *ArrayOps[T]* and *WrappedArray[T]* which extends *Seq[T]*.