PRG1 (6): 多相型

脇田建

2018.10.26

findFirstString

```
def findFirstString(ss: Array[String], key: String): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}
```

- findFirstString: 配列 (ss: Array[String]) のなかから、検索キーの文字列 (key: String) に合致する最初のデータの位置を返す。見つからない場合は -1 を返す。

findFirstNumber

```
def findFirstNumber(ss: Array[Int], key: Int): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}
```

- findFirstNumber: 配列 (ss: Array[Int]) のなかから、検索キーの数 (key: Int) に合致する最初のデータの位置を返す。見つからない場合は -1 を返す。

findFirst{String vs Number}

```
def findFirstString(ss: Array[String], key: String): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}

def findFirstNumber(ss: Array[Int], key: Int): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}
```

findFirst{String vs Number}

```
def findFirstString(ss: Array[String], key: String): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}

def findFirstNumber(ss: Array[Int], key: Int): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}
```

findFirstNumber vs findFirst

```scala
def findFirstNumber(ss: Array[Int], key: Int): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}

def findFirst[A](ss: Array[A], key: Int): Int = {
  def loop(n: Int): Int =
    if (n >= ss.length) -1
    else if (ss(n) == key) n
    else loop(n + 1)

  loop(0)
}
```

型変数 A

- 型変数の導入: def findFirst[A] …
  - 新たな型変数 A を導入する。その変数 A の有効範囲は def の範囲。
    - 型変数 A の意味：「ある型があって、その名前をひとまず A としておこう」、(簡単に)「任意の型Aについて」
- 型構成子の型変数への適用: Array[A]
  - Array[A]: 任意の型 A に関して、型構成子 Array を型Aについて特殊化したもの。
  - Arrayの場合 A はArray が表す配列が要素とするデータの型なので、Array[A] は、A型のデータを要素とする配列の型と読める。

class Array[T]が提供する多相関数

- 型変数に依存しない関数群

- def isEmpty: Boolean
- def length: Int
- def size: Int

class Array[T]が提供する多相関数

- 出現する型変数がTだけで、返り値の型が単純なもの
- def indexOf(T): int
- def forall((T) ⇒ Boolean): Boolean　// ∀
- exists((T) ⇒ Boolean): Boolean　// ∃
- def indexOf(T): int　// Array(1, 2, 3).indexOf(3) => 2
- def count(p: (T) ⇒ Boolean): Int
  // [1,…, 99].count(奇数) => 50

  Range(1, 99).toArray.count((x) => x % 2 == 0) // 実は toArray は不要だけど

class Array[T]が提供する**多相関数**

- **出現する型変数がTだけで、返り値の型がTを含むもの**
- def head: T
- def last: T
- def init: Array[T]　// Array(V^n, v) => Array(V^n)
- def tail: Array[T]　// Array(v, V^n) => Array(V^n)
- def take(Int): Array[T]　// Array(V^k, v, …) => Array(V^k)
- drop(Int): Array[T]　// Array(V^k, v, …) => Array(v, …)

class Array[T]が提供する多相関数

- **型変数Tが引数にも返り値にも出現するもの**

- def filter((T) ⇒ Boolean): Array[T]
  / / Array(1, 2, 3, 4, 5).filter(奇数) => Array(1, 3, 5)

  / / Array(1, 2, 3, 4, 5).filter((n) => n%2 == 1)

- def foldLeft[B](B) ((B, T) ⇒ B): B

trait Set[T] が提供する関数

- isEmpty: Boolean / / 返り値が単相
- empty: Set[A] / / 返り値が多相
- contains(A): Boolean / / 引数が多相
- diff(GenSet[A]): Set[A] / / 引数も返り値も多相 (…[A] => …[A])
- union(GenSet[A]): Set[A]
- map[B]((A) ⇒ B): Set[B]　/ / Array(1, 2, 3).map((x: Int) => x.toString)
- subsets(): Iterator[Set[A]]　/ / Set(1, 2, 3).subsets().foreach(println)

Scala の API マニュアル

- Macユーザへのお薦め: Dash の利用（強力な検索能力）
- そうではない人は、本家のドキュメントをダウンロードして利用。
- http://scala-lang.org/download/all.html を開き、自分が利用している Scala のバージョン(2.12.7)のページを開き、APIDocsのZipファイル(scala-docs-2.12.7.zip)をダウンロードしたあとで展開して利用する。

file:///Users/wakita/Dropbox/scala-2.12.7/api/scala-library/scal

Titech projects Reader.jp github misc dilcolet

Scala 2.12.7 | The Scala Programming Language | Scala Standard Library 2.12.7 - scala

Scala Standard Library 2.12.7 Search

package / class / trait の名前で検索

O, C をクリック

O: Companion Object
C: Class

scala

package **scala**

Core Scala types. They are always available without an explicit import.

Source package.scala

Linear Supertypes

Filter all members

Type Members

```
type ::[A] = scala.collection.immutable.::[A]
```

```
type AbstractMethodError = java.lang.AbstractMethodError
```

```
abstract class Any
```
Class Any is the root of the Scala class hierarchy.

```
abstract class AnyVal
```
AnyVal is the root class of all *value types*, which describe values not implemented as objects in the underlying host system.

Packages

- root
  - scala
    - annotation
    - beans
    - collection
    - compat
    - concurrent
    - io
    - math
    - ref
    - reflect
    - runtine
    - sys
    - text
    - util
    - C Any
    - C AnyVal
    - t App
    - O C Array
    - O C Boolean
    - O C Byte
    - O C Char
    - t Cloneable
    - O Console

クラスとCompanionオブジェクト（連れ合いのオブジェクト）

- class Int vs object Int
- class Array vs object Array
- class List vs object List
- trait Set vs object Set

class Int vs object Int

- Class Int: Int の代数
  - 算術演算子 (+, -, *, /)、比較演算子 (>, <, ==)、ビット毎演算子
  - min, max, signum
- Object Int: Intに関するシステム情報
  - MaxValue, MinValue
  - toString

class Array vs object Array

- class Array: Array[T] への操作
- object Array
  - Array の作成 (empty[T]: Array[T] / emptyIntArray: Array[Int] / ofDim[T](Int, Int): Array[Array[T]] / tabulate[T](T, Int)((T) => T): Array[T])
    - Array.ofDim[Int](3, 4)
    - Array.tabulate(10)((x) => x * x) or Array.tabular(10)(_ * _)
  - Array のデータ設定 (fill[T](Int)(⇒T): Array[T])
    - Array.fill(5)(3), Array.fill(100)(math.random)

object List

- empty[A]: List[A]

- iterate[A](A, int)((A) ⇒ A) / /
  List.iterate(**List.empty[Int]**, 4)((l: List[Int]) => 0::l)

- range[T](T, T) / / List.range(0, 20, 3)