

第 8 章 ジェネリックな型とメソッド (Generic Types and Methods)

Scala のクラスは型パラメータを持てます。型パラメータの使い方を関数型のスタックを例として示します。さて、整数のスタックのデータ型を、メソッド `push`, `top`, `pop`, `isEmpty` とともに書きたいとしましょう。それは次のようなクラス階層になります。

```
abstract class IntStack {
  def push(x: Int): IntStack = new IntNonEmptyStack(x, this)
  def isEmpty: Boolean
  def top: Int
  def pop: IntStack
}
class IntEmptyStack extends IntStack {
  def isEmpty = true
  def top = error("EmptyStack.top")
  def pop = error("EmptyStack.pop")
}
class IntNonEmptyStack(elem: Int, rest: IntStack) extends IntStack {
  def isEmpty = false
  def top = elem
  def pop = rest
}
```

もちろん、同様に、文字列スタックの抽象化を定義するのもよいでしょう。そのために、`IntStack` の既存の抽象化を、`StringStack` と名前を変えると同時に、型 `Int` の出現をすべて `String` に置き換えます。

もっとよい方法は、コードの複製を引き起こさない方法であり、スタック定義を要素の型でパラメータ化することです。パラメータ化によって問題の特定のインスタンスをより一般的なものにできます。今まで、値についてだけパラメータ化してきましたが、型もできます。Stack の **ジェネリック** なバージョンへたどり着けるよう、型パラメータを装備しましょう。

```
abstract class Stack[A] {
  def push(x: A): Stack[A] = new NonEmptyStack[A](x, this)
  def isEmpty: Boolean
  def top: A
  def pop: Stack[A]
}
class EmptyStack[A] extends Stack[A] {
  def isEmpty = true
  def top = error("EmptyStack.top")
  def pop = error("EmptyStack.pop")
}
class NonEmptyStack[A](elem: A, rest: Stack[A]) extends Stack[A] {
  def isEmpty = false
  def top = elem
  def pop = rest
}
```

この定義で、`'A'` はクラス `Stack` とそのサブクラスの **型パラメータ** です。型パラメータは任意の名前でよく、値引数と容易に区別できるように、丸括弧ではなく角括弧で囲みます。次はジェネリックなクラスの使い方の例です。

```
val x = new EmptyStack[Int]
val y = x.push(1).push(2)
println(y.pop.top)
```

最初の行は `Int` のスタックを作成します。形式上の型パラメータ `A` を置き換える実際の型引数 `[Int]` に注意して下さい。

メソッドも型でパラメータ化できます。次は、あるスタックが他のスタックのプレフィックスであるか判定する、ジェネリックなメソッドの例です。

```
def isPrefix[A](p: Stack[A], s: Stack[A]): Boolean = {
  p.isEmpty ||
  p.top == s.top && isPrefix[A](p.pop, s.pop)
}
```

このメソッドのパラメータは **多相的** (polimorphic) と言われます。ジェネリックメソッドも同じく **多相的** と言われます。この用語はギリシャ語由来で「多くの形を持つ」という意味です。isPrefix のような多相的メソッドを適用するには、型パラメータを値パラメータと一緒に渡します。たとえば

```
val s1 = new EmptyStack[String].push("abc")
val s2 = new EmptyStack[String].push("abx").push(s1.top)
println(isPrefix[String](s1, s2))
```

局所的な型推論 [Int] や [String] のような型パラメータをいつも渡すのは、ジェネリック関数を多用するアプリケーションにおいては退屈です。多くの場合、型パラメータの情報は冗長です。なぜなら、正しいパラメータ型は、関数の値パラメータや期待される結果型(戻り値型)から決定できるからです。例として、式 isPrefix[String](s1, s2) を考えると、値パラメータは両方とも Stack[String] であることが分かっており、したがって型パラメータは String に違いないと推論できます。このような状況下で、多相的関数とコンストラクタの型パラメータの省略を許す、かなり強力な型推論器が Scala にはあります。先の例では isPrefix(s1, s2) と書け、見えていない型引数 [String] は型推論器によって挿入されます。

- [8.1 型パラメータの境界 \(Type Parameter Bounds\)](#)
- [8.2 変位指定アノテーション \(Variance Annotations\)](#)
- [8.3 下限境界 \(Lower Bounds\)](#)
- [8.4 最下層の型 \(Least Types\)](#)
- [8.5 タプル \(Tuples\)](#)
- [8.6 関数](#)

[前ページ](#) [8章](#) [目次](#) [次ページ](#)

- 「型パラメータタ」は「型パラメータ」のtypoと思われます。 -- ryugate (2008-05-24 01:23:10)

名前:	<input type="text"/>
コメント:	<input type="text"/>