

第2章 最初の例

最初の例として、Scala でのクイックソートの実装を示します。

```
def sort(xs: Array[Int]) {
  def swap(i: Int, j: Int) {
    val t = xs(i); xs(i) = xs(j); xs(j) = t
  }
  def sort1(l: Int, r: Int) {
    val pivot = xs((l + r) / 2)
    var i = l; var j = r
    while (i <= j) {
      while (xs(i) < pivot) i += 1
      while (xs(j) > pivot) j -= 1
      if (i <= j) {
        swap(i, j)
        i += 1
        j -= 1
      }
    }
    if (l < j) sort1(l, j)
    if (j < r) sort1(i, r)
  }
  sort1(0, xs.length - 1)
}
```

この実装は Java や C で書くものと大変よく似ています。Scala では同じ演算子、似た制御構造を使います。ただし、少しばかり構文的な違いがあります。具体的には、

- 定義は予約語で始まります。関数定義は `def` で始まり、変数定義は `var` で始まり、値 (すなわち読み出し専用の変数) の定義は `val` で始まります。
- シンボルの型宣言は、シンボルとコロンの後に書きます。型宣言は省略できることもあります。コンパイラが文脈から推論するからです。
- 配列の型は `T[]` ではなく `Array[T]` と書き、配列の指定は `a[i]` ではなく `a(i)` と書きます。
- 関数は他の関数の内側に入れ子にできます。入れ子になった関数は、それを包む関数のパラメータやローカル変数にアクセスできます。次の例では、配列 `xs` は関数 `swap` と `sort1` から見えるため、それらにパラメータとして渡す必要はありません。

今までのところ、Scala は構文に少し変わった点があるものの、かなり普通の言語のように見えます。実際、プログラムを書くのは、普通の命令型(手続き型)でもオブジェクト指向でも可能です。これは重要なことです。なぜならこれは、Scala のコンポーネント(部品)を、Java、C#、Visual Basic などの主流言語で書かれたコンポーネントと組み合わせることを容易にする仕掛けの一つだからです。

しかし、プログラムをまったく違うスタイルでも書けます。クイックソートを再び、今度は関数型プログラミング風を書いてみます。

```
def sort(xs: Array[Int]): Array[Int] = {
  if (xs.length <= 1) xs
  else {
    val pivot = xs(xs.length / 2)
    Array.concat(
      sort(xs filter (pivot >)),
      xs filter (pivot ==),
      sort(xs filter (pivot <))
    )
  }
}
```

関数型プログラミングはクイックソート・アルゴリズムの本質を簡潔にとらえています。

- 配列が空あるいは要素が1つなら、すでにソートされているので、直ちにそれを返します。
- 配列が空でなければ、配列の真ん中の要素をピボットとして選びます。
- 配列を副配列に分割し、一つにはピボットより小さな要素を、もう一つにはピボットより大きな要素を、そして三つ目の副配列にはピボットと等しい要素を入れます。
- 初めの2つの副配列を `sort` 関数の再帰呼び出しでソートします(*1)。
- これら3つの副配列をひとつに結合して、結果を得ます。

(*1) これは命令型アルゴリズムのすることと全く同じという訳ではありません。後者では元の配列を、要素がピボットに対して小

さい、あるいは、大きいか等しい、に分けて2つの副配列へ分割します。

命令型と関数型の実装のどちらも同じ漸近的計算量 --- 平均的な場合は $O(N \log(N))$ 、最悪の場合は $O(N^2)$ になります。しかし命令型実装が引数の配列自体を変更するのに対して、関数型実装はソートされた新しい配列を返し、引数の配列を変更しません。ですから、関数型実装は命令型実装よりも一時的なメモリを多く必要とします。

この関数型実装によって Scala が配列への関数的操作に特化した言語の様に見えます。実際は違います。この例で用いた操作は、Scala の標準ライブラリの一部である **シーケンス** クラス `Seq[T]` のたんなるライブラリメソッドであり、そのライブラリ自身も Scala で実装されています。そして、配列は `Seq` のインスタンスなのでシーケンスクラスのメソッドはすべて使用できるのです。

とりわけ、**述語関数** (predicate function) を引数にとるメソッド `filter` があります。この述語関数は、配列の各要素に対して真偽値を返す必要があります。filter の結果は配列であり、元の配列の要素で述語関数が真を返すもの全てからなります。Array[T] 型オブジェクトの filter メソッドは、次のような書き方をします。

```
def filter(p: T => Boolean): Array[T]
```

ここで `T => Boolean` は関数の型を表わし、その関数は型 `T` の配列要素を引数とし Boolean 値を返します。filter のように、他の関数を引数としたり結果として返す関数は、**高階関数** と呼ばれます。

Scala は識別子と演算子の名前を区別しません。識別子は「文字で始まる、文字と数字からなる列」でも、「+」、「*」、「:」のような特殊文字からなる列」でも構いません。Scala ではすべての識別子は中置演算子として使えます。二項演算 `E op E'` は常に、メソッド呼び出し `E.op(E')` として解釈されます。これは文字で始まる中置二項演算子にもあてはまります。したがって、式 `xs filter (pivot >)` は、メソッド呼び出し `xs.filter(pivot >)` と等価です。

先のクイックソートプログラムで、filter は無名関数の引数に3回適用されています。最初の引数 `pivot >` は、引数 `x` をとり、値 `pivot > x` を返す関数を表しています。これは **部分適用された関数** の例です。この関数を見えていない引数を明示して `x => pivot > x` と書いても同じです。この関数は無名、つまり名前を付けて定義されていません。引数 `x` の型が省略されているのは、Scala コンパイラが、関数の使用されている文脈から自動的に推論できるからです。まとめると `xs.filter(pivot >)` は、リスト `xs` の要素で `pivot` より小さいもの全てからなるリストを返します。

最初のクイックソートの命令型実装をよく見ると、二つ目の例で使われている多くの構文が、隠された形ではあるものの、出てきていることが分かります。

たとえば、`+`、`-`、`<` のような「普通の」二項演算子が特別扱いかいされていないことが分かります。append などと同じく、その左側にあるオペランドのメソッドです。したがって、式 `i + 1` は整数値 `i` のメソッド `+` の呼び出し `i.+(1)` とみなされます。もちろんコンパイラは、整数引数に対するメソッド `+` の呼び出しを特例とみなし、効率のよいインラインコードを生成できます (もし、ほどほどに賢いコンパイラなら、そうすることが期待されます)。

効率と、より良いエラー診断ができるように、while ループは Scala の組み込み構文となっています。しかし原理的には、たんなる事前定義された関数であってもよいのです。次は考えうる実装です。

```
def While (p: => Boolean) (s: => Unit) {
  if (p) { s ; While(p)(s) }
}
```

この While 関数は最初のパラメータとしてテスト関数を取り、そのテスト関数はパラメータをとらずブール値を返します。二番目のパラメータとしてコマンド関数を取り、そのコマンド関数もパラメータをとらず、Unit 型の結果値を返します。While 関数はテスト関数が真を返す限り、コマンド関数を呼び出します。

Scala の Unit 型は概ね Java の void に相当し、関数が特定の結果(戻り値)を返さない場合に使います。実際のところ、Scala は式指向の言語なので、すべての関数は何かしら結果を返します。もし明示的な戻り値が与えられなければ、値 `()` --- "unit" と発音します --- が肩代わりします。この値は Unit 型です。Unit を返す関数もまた、手続き (procedure) と呼ばれます。クイックソートの最初の実装中の swap 関数をさらに「式指向」な形にして、これを明示します。

```
def swap(i: Int, j: Int) {
  val t = xs(i); xs(i) = xs(j); xs(j) = t
  ()
}
```

この関数の結果(戻り値)は単に最後の式です。キーワード return は必要ありません。注意点として、明示的に値を返す関数は常に、「=」を本体あるいは定義式の前に必要とします。

[前ページ](#) [2章](#) [目次](#) [次ページ](#)

----- 修正案 by ryugate

この述語関数は配列の要素をブール値に写像させる必要があります。 (filterの引数である) この述語関数は配列の要素をブール

値に写像させる必要があります。

filter の結果は与えられた述語関数が真となる元の配列のすべての要素からなる配列です filter の結果は元の配列の要素のなか
で与えられた述語関数が真となるすべての要素からなる配列です

- ご指摘どうもありがとうございます > ryugateさん。 -- tmiya (2008-05-14 06:59:09)
- おお。コメント欄があったんですね。直接書きちゃってすみません。気づいたら、またコメントさせていただきます。 -- ryugate (2008-05-19 21:53:05)
- いえ、直接本文に追記して頂いてOKです。wikiに慣れてない人の為のコメント欄です。 -- tmiya (2008-05-25 23:26:44)
- sort関数の一番外の{}は無くてもOKでは？ ifの単文なので。 -- Lyc (2010-01-10 06:43:43)
- repl環境だとエラーになりますね -- tskk (2010-07-25 02:20:32)

----- HELP 上の「文字で始まる文字と数字からなる列」の修正を求めます。

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